DESIGN AND CONSTRUCTION OF A LABORATORY SYSTEM FOR NEURO-MUSCULAR STIMULATION OF THE LOWER EXTREMITIES DURING CYCLING

By Matthias H. Popp

Submitted to the University of Cape Town in partial fulfilment of the requirements for the degree of Master of Science in Medicine in the field of Biomedical Engineering

August 1986

ABSTRACT

Functional Neuromuscular Stimulation (FNS) is a method by which paralyzed muscles are stimulated electrically in order to produce a useful movement. The design and testing of a laboratory system for the modulated control of the lower extremities during FNS-induced cycling on an exercising device (Paracycle) is described.

The system hardware, which is designed around a standard IBM compatible Personal Computer, features six independent stimulation channels. Waveform characteristics such as pulse frequency, width and amplitude are defined as a function of the crank position of the Paracycle for each channel. An extensive software package allows programmability of the waveform parameters and supports the user in the definition of stimulation sequences.

The effective performance of the complete FNS-controller/Paracycle system has been demonstrated during a controlled case study with two paraplegic subjects.

ACKNOWLEDGEMENTS

I would like to thank the following people for their contributions to this project:

Peter Kolb, for serving as supervisor of my work, for sharing his technical knowledge and for his help with the design of the shaft encoder interface.

Professor G. Jaros, for providing guidance whenever I needed it, and without whose enthusiasm and commitment the project would not have left the ground.

Gisela Hefftner and Peter Kolb for their valuable advice with the writing up of this thesis.

Dr. H. Goldberg, the Deutscher Akademischer Austauschdienst and the University of Cape Town for their financial support.

The Medical Superintendent of Conradie Hospital, Dr. Kleyn and Dr. Shrosbree for allowing us to work with their patients.

Our two subjects for their kind collaboration without which it would not have been possible to test and evaluate the system.

Gisela Hefftner and Dirk Pons, co-workers on the project for their friendship and support.

My parents, for their interest in the project and for their encouragement and support.

All the people who made my stay in this beautiful and interesting country worthwhile.

Matthias H. Popp August 1986, Cape Town

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND SYMBOLS

A	:	ampere
AC	:	Alternating current
ADC	:	Analogue to digital converter
A/D	:	Analogue to digital
AP	:	Action potential
CMOS	:	Complementary metal-oxide semiconductor
CPU	:	Central processing unit
dB	:	decibel
DC	:	Direct current
EMF	:	Electromagnetic force
EMG (emg)	:	Electromyogram
DIN	:	Deutsche Industrie Norm
EEPROM	:	Electrical erasable programmable read only memory
EPROM	:	Erasable programmable read only memory
FET	:	Field effect transistor
FNS	:	Functional neuromuscular stimulation
HEX (Hex)	:	Hexadecimal
Hz	:	hertz (unit of frequency)
I	:	current
IBM	:	International Business Machines Corporation
k	:	kilo (in conjunction with units)
MOS	:	Metal-oxide semiconductor
ms	:	milli-seconds
P	:	power (electric)
PC	:	Personal computer
PPI	:	Programmable parallel interface
PWM	:	Pulse width modulator (modulation)
RAM	:	Random access memory

RPM (rpm)	:	Rounds per minute
S&H	:	Sample and Hold
µs	:	microseconds
V	:	volt or voltage
VR	:	Variable resistor
VMOS	:	Vertical metal-oxide semiconductor

The author places the electrodes for a training session

Training session with the Paracycle / FNS controller system

Chapter 1 : INTRODUCTION

Statistics in the United States of America suggest that the existing population of spinal cord injury (SCI) patients can be estimated to vary from 150.000 to 500.000 persons. Seven to ten thousand new injuries occur each year (Barto et al., 1985) of which more than 50% are due to motor car accidents. Despite the extensive physiotherapy and occupational therapy treatment programs, rudimentary ambulation is achieved in only 25% of the SCI population. Thus approximately 75% of the SCI population are confined to a wheelchair for life. Current research in the use of functional neuromuscular stimulation (FNS) to activate muscles in which nervous control has been lost, suggests that FNS may be a suitable method for restoring ambulation in a selected population of SCI subjects. While preliminary attempts to achieve this aim have resulted in slow and low quality gait patterns, improvements are expected in the future with use of implanted stimulation hardware. Improved stimulation plans include closed loop control using positional and force feedback.

One of the few FNS applications, which has advanced during the last two decades from the experimental stage to wide clinical use, is the stimulation of the peroneal nerve to activate ankle dorsiflexion for the correction of foot-drop. Although, studies have shown that the FNS braces are not significantly superior to their mechanical counterparts, the potential of FNS must be judged and evaluated in areas in which the classic aids have failed (Marsolais and Kobetic, 1983). These include the maintenance of muscle bulk after spinal cord injury, prevention of pressure sores, improvement of the cardio-vascular fitness,the restoration of the standing posture, and finally the improvement of swing and stance phase of gait. After evaluating the positive effects of FNS on muscle force, size and endurance,

many researchers have concentrated on the use of FNS for the restoration of functional movement, particularly walking.

Bajd et al. (1984) described the positive impacts of FNS induced standing. They claimed, that standing can prevent decubiti, contractures and muscle atrophy, improve function of the bladder and that it can also reduce spasticity. Bajd et al. (1983) described the use of a four channel stimulator for synthesis of simple reciprocal gait. Their paraplegic subject controlled the stimulation with two hand switches and was able to walk short distances with help of a walking frame or cruches.

Petrofsky et al. (1984a) reported on a so called 'feedback control system for walking in man'. Five muscle groups per leg were stimulated via surface electrodes depending on the position of the hip, knee and ankle under computer control. They concluded that the system had been successfully tested in a laboratory environment, however, it only provided limited movement and must be improved particularly as regarding the positional feedback system.

Marsolais et al. (1984b) described the use of implanted percutaneous intramuscular electrodes to provide paraplegic subjects with selected ankle, knee and hip control for walking. Stimulation was controlled by a portable microprocessor based 32 channel stimulator. Quadriceps muscle torque and girth of the thigh increased over a period of several months of exercise. Patients were able to walk with a standard walker for distances ranging from 15 to 50 meters. Major problems were poor balance due to lack of functional hip muscles, high energy requirements and non-related medical complications. The authors stressed the need for the application of closed-loop technology to reduce the energy needs, and the development of reliable sensors to provide feedback information for the closed-loop control.

The results of the above-mentioned research activities were encouraging, although the quality of FNS induced gait remained poor. The main reason for this is the lack of suitable sensors (ie. implantable position and force transducers) and control strategies which would make true closed loop control possible. Other problems are related to the electrodes. In general, the use of surface electrodes has proven to be unsatisfactory for FNS outside the laboratory, because of poor selectivity of excitation, inadequate repeatability of the position and limited patient acceptance. Furthermore, some muscle groups (eg. hip flexors) can only be stimulated by intramuscular electrodes (Chizeck et al., 1984). In addition, there are physiological problems related to fitness of muscles, bones, joints and the cardio-vascular system after prolonged paralysis. Because of these extensive problems involved in using paralyzed muscles for free walking, many researchers found it reasonable to apply FNS to a well defined functional movement such as knee extension (Glaser et al., 1983) or cycling (Petrofsky et al., 1983; Petrofsky et al., 1985a).

Glaser et al. (1983) demonstrated the feasibility of using electrically stimulated paralyzed leg muscles to propel a modified conventional wheelchair. Potential benefits of this vehicle were thought to be an improved circulation of blood in the lower extremeties, an improvement of cardiovascular fitness and an increase of size and strength of the exercising muscles.

Petrofsky et al. (1985a) described an aerobic exercise trainer, which can be used by both paraplegic and quadriplegic subjects. The mechanical design was based on a commercially available bicycle ergometer, which has been modified for this application. Stimulation was controlled by a microprocessor,

according to the sensed pedal position. In addition, the system monitored the heart rate, blood pressure and body temperature to make it safer for home applications. Three muscle groups per leg (Hamstring, Gluteus Maximus, Quadriceps) were stimulated. The authors stressed, that with 15 minutes training three times a week over a period of 6 weeks, the endurance of the subjects increased up to 100 fold, and that the influences of this type of exercise on cardio-vascular fitness were favourable (Petrofsky et al., 1985b).

Encouraged by the results of international research groups in the FNS field, the project 'REWALK' was initiated by the Department of Biomedical Engineering at the University of Cape Town and the Department of Electrical Engineering of the University of Pretoria at the beginning of 1984. Ultimate aim of project REWALK is the restoration of gait in paraplegic subjects. To acquire a complete knowledge of the medical and technological problems related to FNS, the project 'PARACYCLE' was set up. Two masters projects were involved in the design and construction of the system, with which it is possible to create a computer controlled cycling movement of the lower extremeties in paraplegic subjects.

One of the projects concerned itself with the design and construction of the mechanical device, called the Paracycle, for exercising paralysed leg muscles and providing spinal cord injury patients with mobility. Important design features of the Paracycle are the following:

- A seat for a reclined body position during cycling.
- An electric motor, which allows passive motion of the legs in order to reduce joint stiffness, and to assist or retard the pedalling motion when necessary.

- Gears, spanning a wide range, to allow paraplegics to move the vehicle forward.
- Braces, to stabilise the ankle and prevent injury.
- Controls, for steering, gear change, braking and stimulation.

The aim of this thesis was the design and construction of a suitable stimulator system to control the cycling movement of paraplegic subjects on the Paracycle. Furthermore, it was intended to use the stimulator system for FNS applications other than cycling. In order to fulfill both these aims, it was decided to use a computer for the control of the stimulation process.

The design of the stimulator system (ie. FNS-controller) is based on a standard IBM compatible Personal Computer for reasons of its known hardware architecture and expandability, the availability of suitable software and low cost. For the ease of change and expandability of the software as well as its speed it was decided to use a compiled high level programming language (TURBO PASCAL) to control the system. In order to fulfil this control task the Personal computer's processor had to be freed from the waveform generation process. Therefore, digitally programmable waveform generator modules were designed to fit into the expansion slots of the computer. For the collection of analogue control signals and data, a 12-bit analogue to digital converter had to be designed. To allow definition of the numerous parameters determining the outcome of the stimulation with ease, a software package had to be developed which provides an user friendly working environment.

Section one of this thesis deals with the physiological background of electrical muscle stimulation and the stimulation systems decribed in the literature. Section two gives an account

of the hardware and software design of the FNS-controller. Section three describes the results of the evaluation of the FNS-controller/ Paracycle system. Conclusions and recommendations for future work with the FNS-controller are found in section four and a detailed account of the hardware and software details is given in the appendices.

SECTION I : PREPARATORY MATERIAL

Chapter 2 : STIMULATION PARAMETERS

2.1. INTRODUCTION

Functional Neuromuscular stimulation (FNS) is a technique in which electrical stimulation is delivered to nerves and/or muscles in a controlled way in order to create functional limb movements. The most suitable type of stimulation is the one in which electrical current is provided in the form of pulses. It is possible to describe pulsed stimulation completely by specifying the stimulation parameters, which include the shape, amplitude and duration of the individual pulses and their repetition frequency. Until recently, the variation of stimulation parameters has been used in an "on-off" fashion by most researchers. ie. stimulation parameters are held constant during each period of stimulation. Some researchers like McNeal and Meadows (1984), Petrofsky et al. (1985a), Thrope et al. (1985) and Vance et al. (1983) have shown that improved performance can be achieved by modulating stimulation to produce a graded response in the stimulated muscle. The success of such a modulated control largely depends on the knowledge of the effects of various stimulus parameters on muscular contraction during stimulation. The modulation of the amplitude and duration of the pulses with time is usually used to grade the level of muscular contraction. This is commonly referred to as the stimulation sequence or envelope.

Many of the influences of the stimulation parameters on the efficiency of FNS can be predicted from a knowledge of the physiology of muscle and nerve stimulation. The physiological basis of electrical stimulation is discussed in this chapter. The various pulsatile waveforms and the influences of different stimulation parameters are reviewed.

2.2. ELECTRO-PHYSIOLOGY OF NEUROMUSCULAR STIMULATION

All excitable cells maintain a steady resting potential across their membrane when they are in their resting state. The resting state of nerve is such that the potential on the inside of the cell membrane is typically -70mV with respect to the surrounding environment. Muscle cells maintain a slightly higher resting potential of approximately -90mV (Ganong, 1977). Any disturbance to the membrane, whether chemical, mechanical or electrical can polarize or depolarize the cell membrane by changing its permeability. If the disturbance to the cell membrane is sufficient to exceed the critical polarization threshold, an action potential (AP) is initiated. The absolute amount of hypopolarization necessary to reach threshold is essentially the same for all nerve fibres, though the actual amount realized by a single stimulus varies with specific characteristics of each cell (Ganong, 1977; Guyton, 1982). By application of an electrical

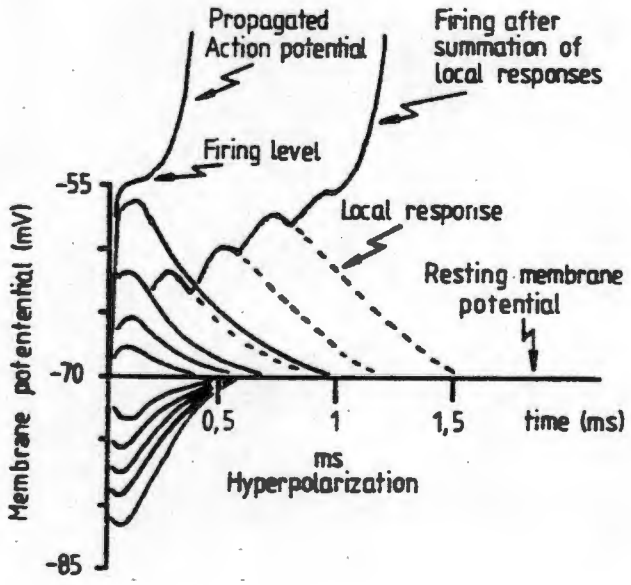


Figure 2.1 ELECTROTONIC POTENTIALS AND LOCAL RESPONSE (adapted from Ganong, 1977) : The changes in the membrane potential of a neuron following application of stimuli of 0.2, 0.4, 0.6, 0.8, and 1.0 times threshold intensity superimposed on the same time scale.

potential across a nerve, a current is passed through the membrane producing transient hypopolarisation and if it is of adequate intensity and duration an action potential may be initiated. Any stimulus that does not cause the muscle or nerve cell in question to reach threshold is said to be a subthreshold stimulus. Although no action potential results, local changes in the membrane potential occur (see Figure 2.1). These 'local potentials' are important as they may actually sum up to initiate an action potential.

The time constant, T which is characteristic of a particular cell membrane, determines the rate of rise and decay of the local potentials following the application or removal of an applied voltage. The time constant of a membrane is dependent on the product of the resistance and capacitance of the membrane. It determines the minimum duration that a certain voltage must be applied before threshold is reached and an action potential initiated. Figure 2.2 is a schematic representation of the rates with which a charge can build up across nerve and muscle cell membranes.

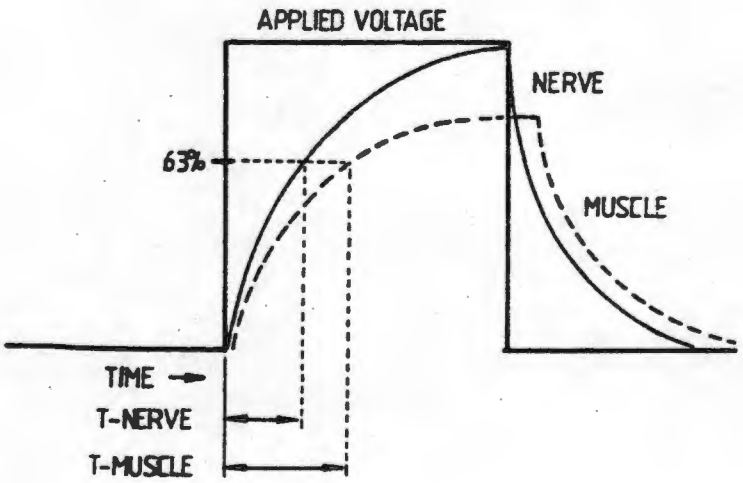


Figure 2.2 RATE OF CHARGING AND DISCHARGING NERVE AND MUSCLE CELL MEMBRANES (adapted from Baker, 1981) : Time constants (T) equal the time it takes the membrane to charge up to approximately 63% of the applied voltage

The capacitance of muscle is approximately ten times greater than that of a nerve cell, which means that for any applied voltage, the muscle membrane takes longer to charge up to the applied voltage than does nerve. Therefore, the nerve will always reach threshold before a muscle, and thus in innervated muscle, electrical stimulation will always occur via the motor nerve. During the action potential a nerve or muscle cell is refractory to stimulation. This refractory period is divided into absolute refractory and relative refractory periods. During the first period no stimulus , whatever its strength, will excite the respective cell again. During the refractory period stronger than normal stimuli can cause excitation. The absolute refractory period of a motor fibres lasts for 0.4 to 1 ms.

The characteristic strength duration curve (see Figure 2.3) demonstrates the relationship between pulse duration and amplitude of current needed to minimally excite nerve or muscle cells. As the duration of the stimulating pulse is shortend, higher and higher voltages are required to reach threshold. There exists a minimum duration, below which no action potential can be initiated at all. Rheobase is the minimum DC current necessary to produce stimulation of an excitable cell. Chronaxie is the minimum pulse duration required to activate excitable tissues at twice the amplitude of rheobase. While rheobase values are similar for nerve (left curve) and muscle (right curve) note the difference in chronaxie values for the two types of excitable tissue.

From Figure 2.3 it can be seen that the chronaxie for muscle is more than two orders of magnitude greater than that of nerve cells. This suggests that the intensity of the applied stimulus can be used to recruit various nerve fibres selectively.

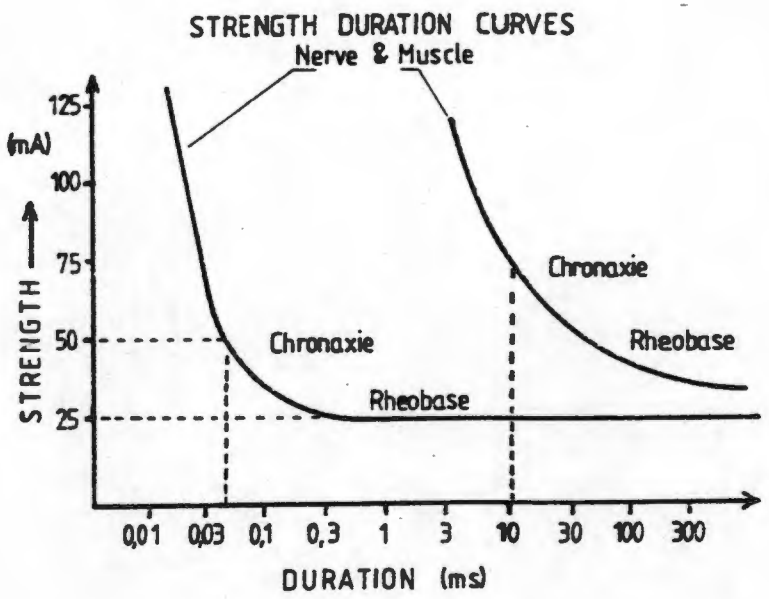


Figure 2.3 Strength duration curve (adapted from Baker,1981)

The amplitude of the current required to excite a peripheral nerve fibre is inversely proportional to the fibre's diameter. This means that large diameter fibres are recruited with the least stimulus amplitude, while fibres of the smallest diameter require the greatest intensity (Ganong, 1977).

The threshold for the initiation of an action potential varies with the rise time of the stimulus. This is known as accommodation. The more slowly the stimulus amplitude rises, the greater the stimulus amplitude required to initiate the action potential. Nerve accommodates very quickly and thus an abrupt rise of the stimulus is required. Muscle accommodates much more slowly and stimuli with a more gradual rate of rise may be used. Figure 2.4 shows the strenghth/duration curve for various nerve and muscle fibres.

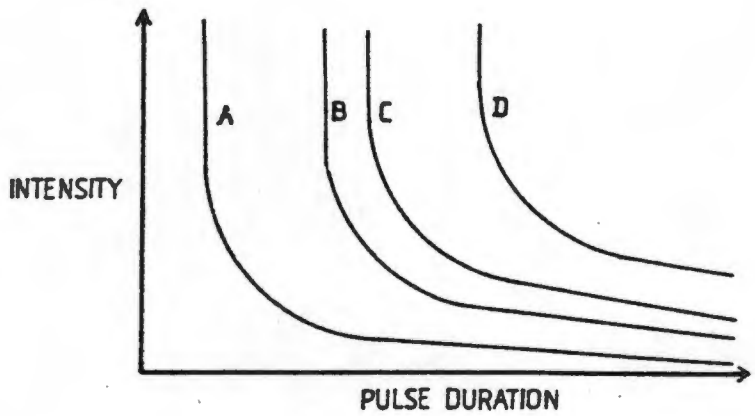


Figure 2.4 **STRENGTH DURATION CURVE FOR DIFFERENT NERVE FIBRES AND MUSCLE (adapted from Baker, 1981) : Plot of the time duration versus intensity of current to excite (A) large-diameter myelinated motor and sensory nerve fibres (A-alpha). (B) small-diameter myelinated sensory nerve fibres (A-delta). (C) small-diameter myelinated nerve fibres (C fibres). (D) denervated muscle.**

It can readily be seen that to selectively activate different populations of nerve fibres, different combinations of pulse duration and amplitude may be used (Baker, 1981).

In addition to selective recruitment of different nerve populations, based on fibre diameter, the intensity of the stimulus can also be used to grade the magnitude of response. This gradual recruitment is in part due to the greater depth to which the current penetrates at higher intensities. The peak-current and not the average-current determines the depth to which an electric impulse penetrates (Benton et al., 1981).

2.3. PHYSIOLOGY OF MUSCLE CONTRACTION

The electrical events after stimulation in skeletal muscle are similar to those in nerve. However, in muscle the action potential lasts longer (2-4ms) and is conducted along the muscle fibre slower (at about 5 meters per second). The absolute refractory period is 1-3 ms long (Ganong, 1977). (See Figure 2.4 for the differences in the strength-duration curve between nerve and muscle.)

THE MUSCLE TWITCH:

As decribed earlier, at a given amplitude the pulse width required to activate muscle directly is approximately two orders of magnitude greater than that used to activate the muscle through the nerve. Therefore, FNS of innervated muscle takes place via the peripheral motor nerves leading to a specific muscle. Each motor nerve leaving the spinal cord usually innervates many muscle fibres, depending on the type of muscle. Fibres innervated by a single motor neuron together form a motor unit. Small muscles which act rapidly and with high accuracy generally have few muscle fibres in each motor unit (eg. in ocular muscles each motor unit may contain as few as 3 muscle fibres). An average Figure for all the muscles in the human body can be considered to be about 150 muscle fibres to the motor unit (Guyton, 1982).

An incoming action potential initiates the depolarization of the muscle fibres at the motor endplate. The action potential is then transmitted via the Sacrotubular system along the muscle fibres and elicits the contractile response. A single incoming action potential causes the muscle to twitch with a defined

duration. The mechanical contractile response starts about 2 ms after the action potential of the motor nerve, and it's duration and amplitude depends on the type of excited muscle fibre. Young and Mayer (1981) classified the single motor units by the contraction time Tc of the muscle twitch and the fatigue index FI, which is a measure for the fatigue resistance. Figure 2.5 illustrates the characteristics of a population of single motor units.

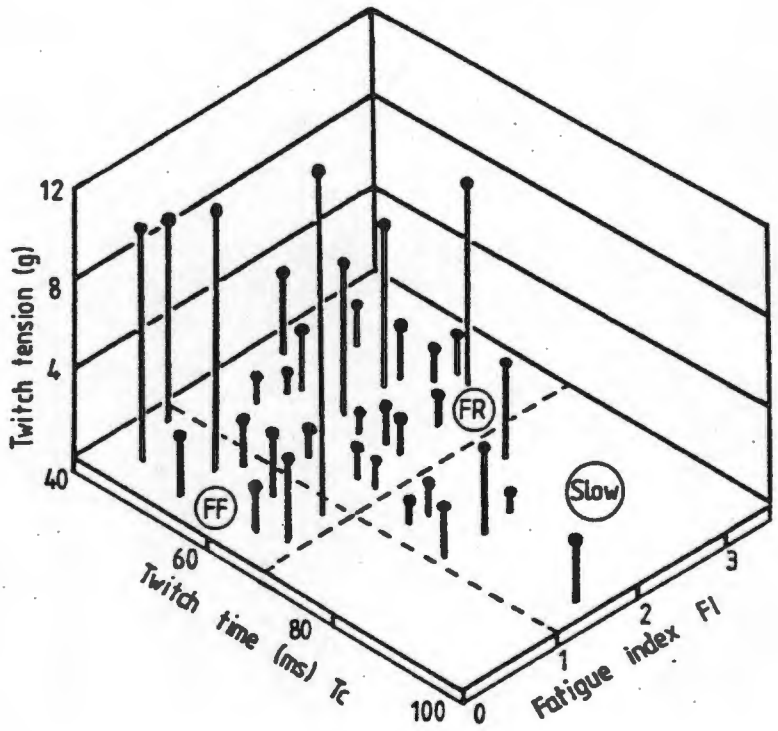


Figure 2.5 **CLASSIFICATION OF A POOL OF SINGLE MOTOR UNITS (adapted from Young and Mayer, 1981) : Three-dimensional plot relating twitch time, twitch tension, and fatigue index (FI) in a population of indentified single motor units in a human first dorsal interosseous.**

On basis of the contraction time Tc alone fast and slow motor units can be identified. Fast motor units having a Tc of less than 70 ms. Slow twitch fibres (S) produce smaller twitch tensions with a high fatigue resistance (FI>=1). The fast motor units can be subdivided in fast twitch, fatigue resistent (FR) and fast twitch, fatiguable (FF). Figure 2.6 gives the isometric

contraction of three different single motor units before and after fatiguing.

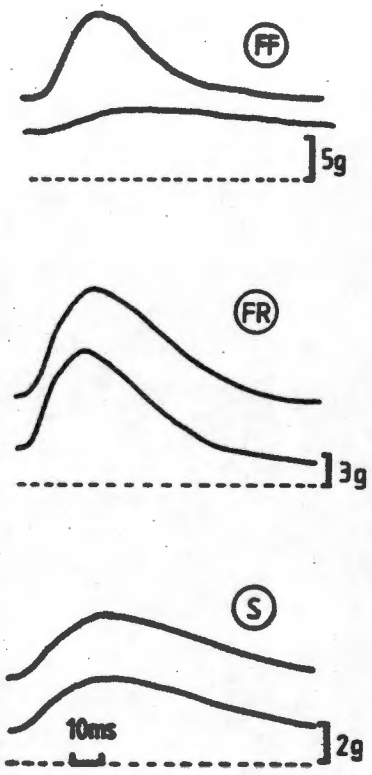


Figure 2.6 FATIGUE OF DIFFERENT SINGLE MOTOR UNITS (adapted from Young and Mayer, 1981) : isometric contractions of single motor units are shown before (upper traces) and after (lower traces) fatigue. (FF = fast twitch, fast fatigue unit; FR = fast twitch, fatigue resistent unit; S = slow twitch unit)

Figure 2.7 illustrates the isometric contractions of three different types of skeletal muscles caused by a single action potential.

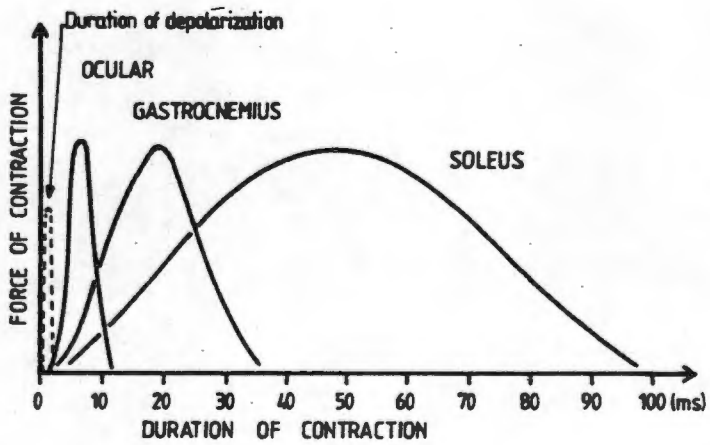


Figure 2.7 Single muscle twitch of skeletal muscles (adapted from Guyton,1982)

It can easily be seen that the durations of the contractions are adapted to the function of each of the respective muscles. For example ocular movements must be extremely rapid to maintain fixation of the eye upon objects.

TETANIZATION

As said earlier, the electrical response of a muscle fibre to stimulation is similar to that of a nerve cell. However the muscle fibre is electrically refractory only during the rising part of the action potential, at which time the contraction initiated by the first stimulus is just beginning. Because the contractile mechanism does not have a refractory period (Baker, 1981; Ganong, 1977; Guyton, 1982) repeated stimulation before relaxation produces additional activation of the contractile elements and a response which is added to the contraction already present. This is known as summation of contractions.

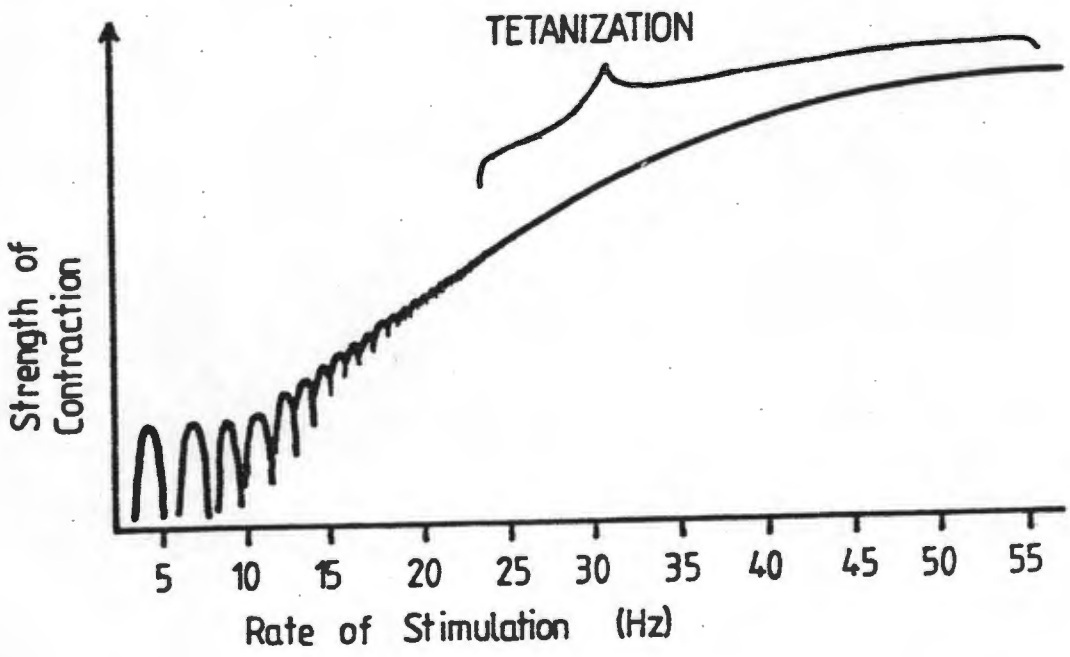


Figure 2.8 Summation and Tetanus (adapted from Guyton, 1982)

The muscle force created during summation is considerably greater than that during a single muscle twitch. If the frequency of the incoming actionpotentials is high enough the individual responses fuse to a smooth steady contraction. This is known as tetanus. Figure 2.8 shows the summation and tetanizing effects with increasing rate of stimulation. It can easily be seen that the tension developed by a tetanic contraction is about 4 times greater than that of a single muscle twitch.

RECRUITMENT

Force modulation of muscular contractions takes place via two mechanisms. Firstly, muscle tension increases with the number of recruited motor units. Secondly, the force is graded by the rate at which individual motor units are stimulated. Note that the recruitment of single motor units takes place in asynchronous fashion, so that even when the individual motor units are recruited at a low frequency, the individual motor unit contractions sum up to a smooth global contraction of the muscle. Generally, the fatigue resistant motor units, innervated by motor nerves with small diameter, are recruited first, followed by larger motor nerves innervating less fatigue resistant fast twitch motor units. This physiological phenomenon causes major problems in the application of FNS, as electrical stimulation in general excites first the larger motor neurons which results in a strong, but fast fatiguing contraction (Benton et al., 1981).

2.4. STIMULATION WAVEFORMS

For these physiological reasons, pulsatile stimulation, (usually a rectangular wave), is currently universally accepted (Bajd et al., 1983; Marsolais et al., 1984a; McNeal and Baker, 1985; Petrofsky et al., 1985a; Thoma et al., 1983); Gorman and Mortimer, 1983. Currents can be delivered in either monophasic or biphasic pulses (see Figure 2.9).

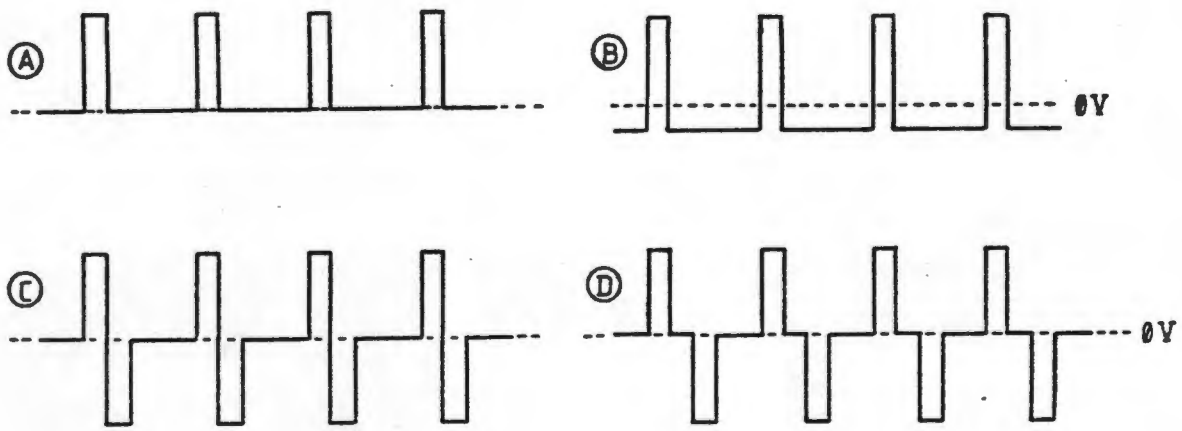


Figure 2.9 **STIMULATION WAVEFORMS : (A) monophasic waveform with unidirectional current flow; (B) Asymmetric biphasic waveform with charge balanced current flow in both directions; (C) Symmetrical waveform with current flow immediately reversed; (D) Symmetrical waveform with time delay before the current flow is reversed.**

A monophasic rectangular wave has the potential disadvantage of causing polarization under the electrodes due to unequal ionic flow, because the current is continually passed through the tissue in one direction. Such ionic flow can lead to electrode deterioration and more seriously to skin irritation, particularly when applied for prolonged periods. A modification of the monophasic, unidirectional rectangular waveform is an asymmetrical, biphasic waveform, which allows equal current flow in both directions to minimize skin ionisation, while providing a

monophasic stimulation effect.

Using monophasic pulses, the active electrode is the cathode, as nerve excitation occurs at the point where the current leaves the nerve. Because the nerve membrane is sensitive to current density, a small cathode will concentrate ions around the target nerve (Baker, 1981).

Symmetrical waveforms allow both electrodes to be active during alternating half cycles. The effect can be particularly useful when large muscle groups, such as quadriceps femoris, are to be stimulated. Baker (1981), and Bowman and Baker (1985) claim that evaluations in non paralysed subjects to determine which type of waveform is most comfortable, indicate that the majority of the subjects preferred biphasic stimuli. However Gracanin and Trnkoczy (1975) who were investigating the sensation of pain during stimulation with different waveforms, found that biphasic stimuli had an "unfavorable effect on optimal stimulation". This shows the importance of further research relating to stimulation waveforms used in patients with incomplete lesions, who have restricted sensation in their extremities.

Stefanowska et al., (1983) compared the influence of sinusoidal and rectangular stimulation to voluntary isometric contraction of the quadriceps muscle group. Figure 2.10 shows the stimuli presented at a frequency of 25 Hz. They concluded in his study that rectangular pulses with a pulse width of 0.3 ms produce greater force than 20 ms bursts of a sine wave with a frequency of 2000 Hz.

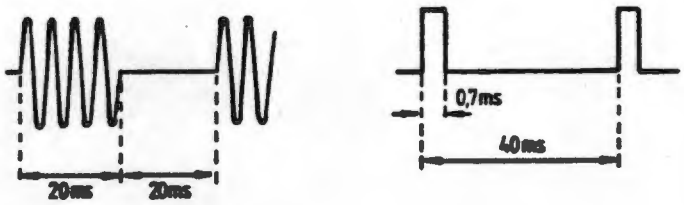


Figure 2.10 Shape of stimulation waveforms used by Stefanovska et al., (1983)

2.5. STIMULUS FREQUENCY

In using electrical stimulation to achieve functionally useful movements, a smooth tetanising contraction of the muscle is usually desired. When a stimulus of one pulse per second is given, there is a definite muscle twitch. As the frequency of the stimulus is increased, the twitching becomes less pronounced and muscle tension increases. Depending on the fibre constituents of the muscle being stimulated, this smoothening of the stimulated contraction occurs between 15 and 25 pulses per second (Hz). As the frequency of the stimulation is increased above 35 Hz little additional force is gained from the muscle contraction and the existing smooth action is not altered. Figure 2.11 shows the effect of the stimulus frequency on the tension developed by a normal human quadriceps. Although the smooth motor response elicited by electrical stimulation is similar to a normal muscle contraction, it is in fact metabolically expensive and fatiguing.

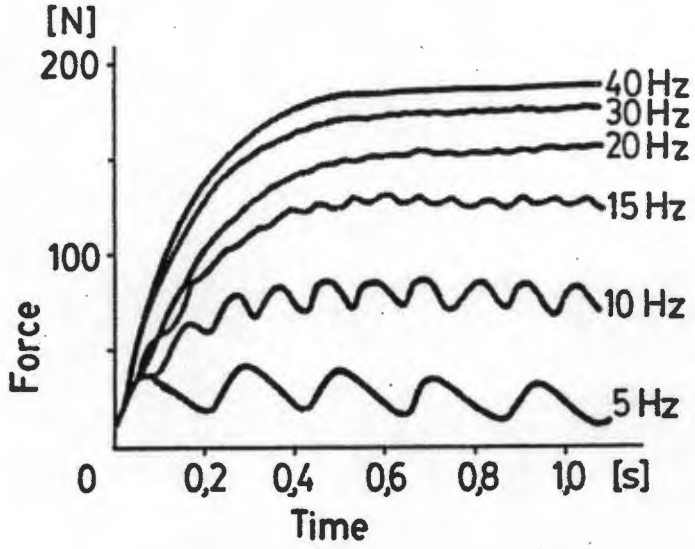


Figure 2.11 Force versus Stimulus frequency (adapted from Edwards et al., 1977)

This results from the synchronized firing of motor units when electrical stimulation is used, while normal physiological activity occurs asynchronously. Thus the stimulated motor action is due to a synchronized firing of a few fibres. The same voluntary contraction may be due to a much larger population of muscle fibres being activated asynchronously and at a low frequency (Baker, 1981).

The effect of increasing the stimulus frequency above the smooth tetanising level is to create muscle fatigue under prolonged stimulation. The time required to fatigue a muscle by electrical stimulation varies according to the amplitude and frequency of the stimulation, the fibre type and composition of the stimulated muscle, and the patient's general state of health. Figure 2.12 shows the fatigue effect during prolonged stimulation of a human quadriceps (Edwards et al., 1977).

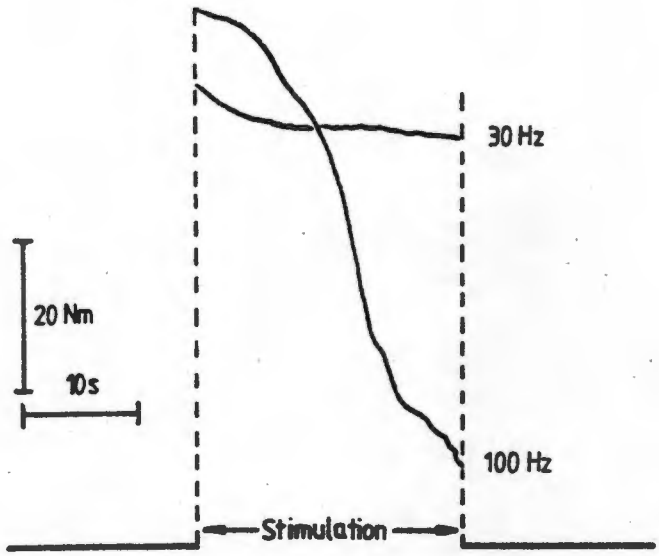


Figure 2.12 **FATIGUE DURING PROLONGED STIMULATION (adapted from Edwards et al., 1977) : Comparison of the maintenance of force during prolonged stimulation of the quadriceps femoris at 30 Hz and 100 Hz**

Thus in general, there is a trade-off between the moderate frequency required to create a smooth tetanizing contraction and a low frequency to minimize fatigue.

2.6. PULSE DURATION AND AMPLITUDE

Most stimulators currently used for FNS deliver an electrical current in discrete pulses. The characteristics of the pulse (ie. its amplitude and duration) determine the type of motor reponse that will occur. The standard strength-duration curve (see Figure 2.3) shows the typical pattern of activation of both nerve and muscle. As said before the pulse duration required to activate muscle directly is approximately two orders of magnitude greater than that used to activate the muscle through the intact nerve (Eichhorn et al., 1984). Thus short pulses (0.1 to 1 ms) regardless of their other characteristics, commonly activate the

nerve. Therefore, intact peripheral nerves are required to activate a muscle with pulses of that duration. Regardless of amplitude, pulse durations less than 1 ms will not be able to activate denervated muscle.

Within the range of 20 µs to approximately 500 µs the duration of the electrical pulse will determine the amplitude required to stimulate a nerve and hence the muscle (Marsolais et al., 1984a). Thus by maintaining the amplitude at a set level and varying the pulse duration eg. from 50 to 300 µs, a muscular contraction can be graded up to near maximal strength. Equally the pulse width could be kept constant and the muscular contraction can be controlled by varying the amplitude from 15 mA to 50 mA (see Figure 2.13).

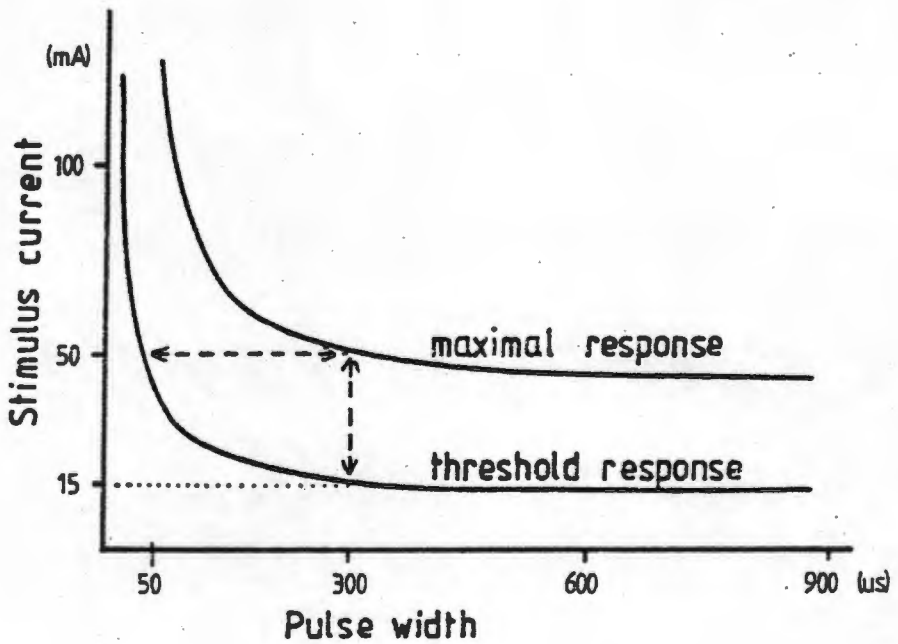


Figure 2.13 Pulse amplitude and duration relationship (adapted from Benton et al., 1981)

Stimulating pulses of a longer duration than 500 us have been found to be significantly less comfortable than pulses of shorter duration, because sensory nerve fibres (A-delta, C-fibres) become affected (Gracanin and Trnkoczy, 1975; Bowman and Meadows, 1983; Baker, 1981).

An additional characteristic of the stimulation pulse is the rise time. When the current pulse rises slowly, the nerve fibre may accomodate at the membrane level and may therefore fail to spike, despite a seemingly adequate amplitude and pulse duration. Thus, an abrupt rise is likely to activate the nerve more consistently then a slowly rising pulse (Baker, 1981).

The motor response can be altered by fixing the amplitude of stimulation and varying the pulse duration or vice versa. The combination of stimulus amplitude and pulse duration determines which nerve fibre will be activated by eletrical stimulation. The nerve/muscle motor system responds to the amplitude of stimulation with a characteristic S-shaped curve (see Figure 2.14).

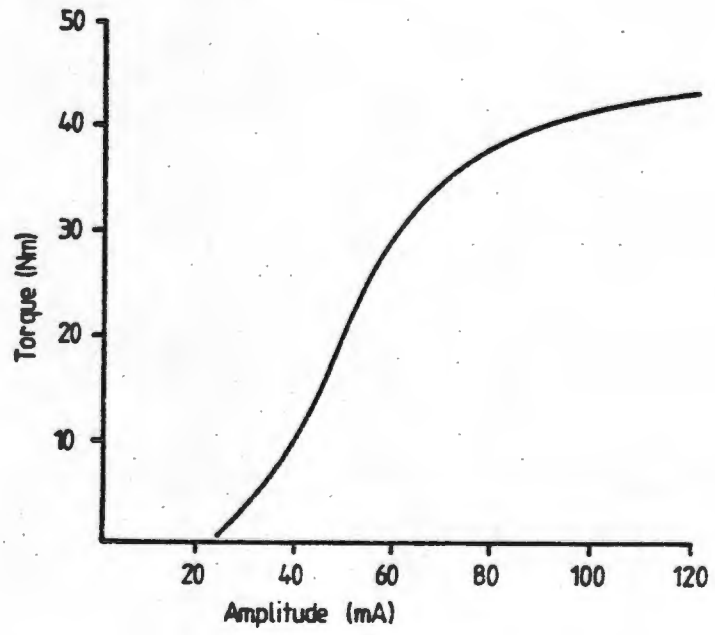


Figure 2.14 Force versus stimulation amplitude (adapted from Baker, 1981)

Note that until a certain threshold is reached the muscle shows no reaction.

2.7. RISE TIME OF STIMULATION LEVEL

A further improvement to stimulation technique is the gradual increase of stimulation level. The advantages of this are twofold : firstly, the patient is not startled by the sudden onset of stimulation, and secondly, a more normal smooth movement can be achieved. In spastic patients a quick stretch may elicit a counterproductive contraction also, while a slow rise time allows a slow, prolonged stretch of the spastic muscle and enhances the range of motion and stimulation efficiency.

The use of electrical stimulation to create fast movements as required in cycling, however, requires relatively short rise and fall times of the stimulation level and therefore is only suitable for patients with a low degree of spasticity (Baker, 1981).

2.8. DUTY CYCLE OF STIMULATION

Duty cycle is the ratio of stimulation time to rest time. The duty cycle greatly affects the fatiguing of the muscle during prolonged electrical stimulation sessions (Baker et al., 1986). When a patient first begins an FNS training program, a relatively long off-time may be used to ensure the ability of the muscle to continue to respond throughout the treatment session. If the muscle becomes trained the off-time can be reduced, progressively. (Bajd et al., 1983).

2.9. ELECTRODES AND STIMULATION SOURCES

An additional factor important for application of FNS is the type of electrical generator supplying the stimulation voltage and current. Constant voltage sources produce the same voltage waveshape and amplitude regardless of the changes in tissue and electrode impedance. This can result in unpredictable situations due to changes at the electrode/skin interface during and between treatment sessions.

A constant current source, on the other hand, maintains the same current waveform, regardless of electrode impedance. Consequently the effects of stimulation (dependent on current density through the stimulated nerve) are more consistent and easier to predict. However constant current sources must have a

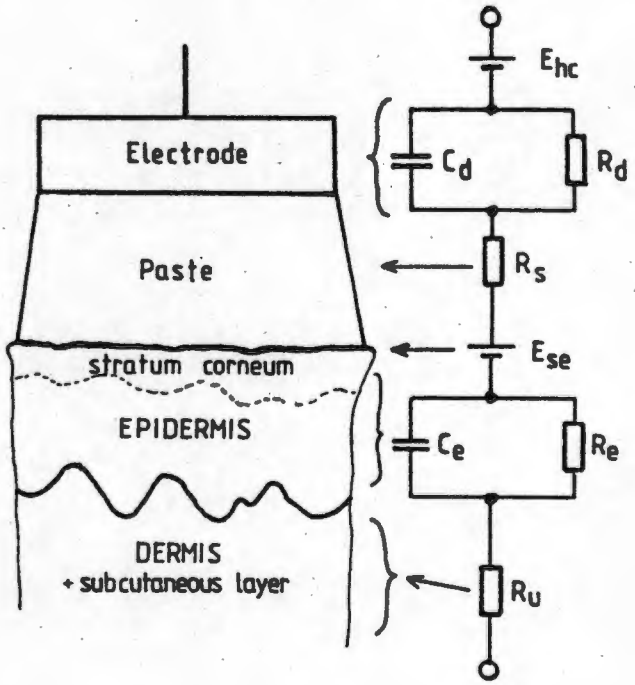


Figure 2.15 EQUIVALENT ELECTRICAL CIRCUIT DIAGRAM FOR THE ELECTRODE/SKIN INTERFACE (adapted from Neuman, 1978) : Each circuit element on the right is at approximately the same level at which the physical process that it represents would be in the left-hand diagram.

safety feature of voltage limitation to avoid possible skin irritation resulting from the increased voltage across the electrode/skin interface if the electrode contact with the skin is poor.

From Figure 2.15 it is possible to identify the different skin components which make up an electrical equivalent circuit for the electrode/skin interface (Neuman, 1978). E_{hc} represents the half cell potential created by a metal electrode and the electrolyte paste. The parallel RC circuit R_d C_d results from the electrode/electrolyte interface. The series resistance R_s is the effective resistance of the paste between electrode and the skin. The stratum corneum which is the outermost layer of the edidermis can be considered as a semipermeable membrane to ions with a characteristic potential difference of E_{se} across it. The epidermal layer can be described with a parallel circuit of R_e and C_e. The dermis and the subcutaneous layer under it behave in general as a pure resistance (R_u).

The effect of the half cell potential E_{hc} can be neglected if carbon rubber electrodes are used for stimulation as they are inert to the electrolyte paste (Neuman, 1978).

The target of the electrical stimulation are motor nerves lying in the subcutaneous layers represented by the resistive component R_u. The effectiveness of stimulation is therefore mainly determined by the ratio of R_u to the rest of the reactance of the electrode/skin interface. It has been shown that most of this reactance is due to the stratum corneum (Boxtel, 1977). The effect of the stratum corneum can be minimized by removing it, or at least part of it, from under the electrode. This process tends to short out E_{se} and to reduce C_e and R_e significantly (Boxtel, 1977; Neuman, 1978). Boxtel (1977) reports that the removal of the stratum corneum reduces the

resistive components (ie. $R_d + R_s + R_e + R_u$) from approximately 6000 Ohms to 500 Ohms (measured at a current level of 10 mA).

Figure 2.16 illustrates the current and voltage waveforms seen with carbon rubber electrodes used for electrical stimulation with current and voltage regulated sources.

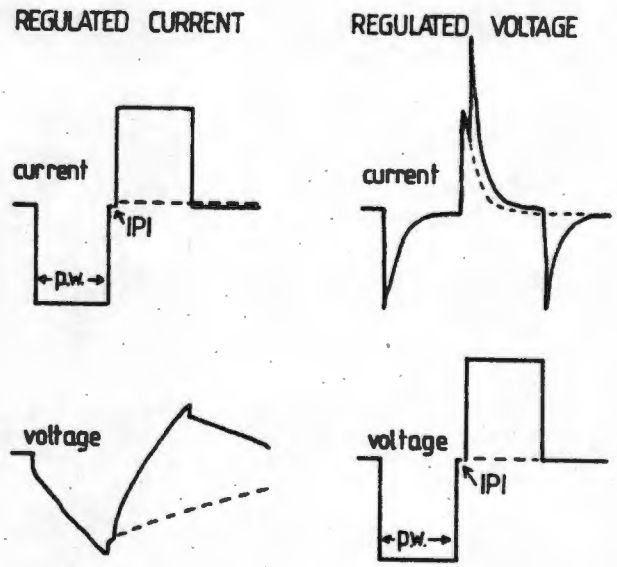


Figure 2.16 VOLTAGE/CURRENT WAVEFORMS USING REGULATED VOLTAGE OR CURRENT SOURCES AND CARBON RUBBER ELECTRODES (adapted from Bowman and Baker, 1985) : ---- = symmetrical waveform, - - - = asymmetrical waveform; p.w. = 300 µs; IPI = 20 µs;

It can be seen that due to the capacitive nature of the electrode/skin interface impedance, the current that flows during an assymmetrical regulated voltage pulse is very close to symmetrical biphasic. The symmetrical biphasic regulated voltage Pulse results in a triphasic current flow with the anodic current flow much greater than the cathodic. It should be noted that these waveforms change with different pulse durations and electrode/skin impedances.

The fact that the use of a current regulated source results

in a defined stimulating current over a range of electrode impedances, and also in a biphasic current and voltage waveform may be the reason why many research groups prefer this type of source (Petrofsky et al., 1985a; Bowman and Baker, 1985; Marsolais et al., 1984a).

2.10. Summary

For physiological reasons biphasic square wave pulses are most suitable for FNS. Stimulation frequencies between 20 and 35 Hz constitute a good compromise between resulting contraction force and muscle fatigue. The level of contraction can be varied by changing the pulse width and/or the pulse amplitude. The range of pulse widths suitable for FNS of innervated muscle lies between 30 and 300 µs, at pulse amplitudes between 10 and 100 mA. The source of the stimulation waveforms should be current regulated.

Chapter 3 : LITERATURE SURVEY :

In general there are three different types of stimulators used for Functional Neuromuscular Stimulation (FNS). Firstly, there are the stimulators acting as waveform generators with adjustable frequency, output amplitude and pulse width. Secondly, there are hardwired logic based systems which allow stimulation sequences to be defined in order to generate certain simple movement patterns. Thirdly, programmable microprocessor controlled devices make it feasible to customise stimulation patterns for each patient. In this chapter the first two groups of stimulators are described briefly. Thereafter various microprocessor based FNS systems are discussed in depth.

3.1. HARDWIRED STIMULATORS

The first group of stimulators is used routinely in rehabilitation. These devices allow the adjustments of few stimulation parameters like amplitude, pulse width and frequency by means of potentiometers and switches. They control simple movements. such as cyclical contraction of the knee extensor muscles in order to preserve muscle bulk, improve muscle strength and circulation in the paralysed extremeties (Bajd et al., 1984). When this type of strengthening program is successful the patients can stand up or even walk with a simple reciprocal gait under control of a relatively simple hardwired stimulator (Bajd et al., 1983). With the researchers' objective being to improve the quality of FNS induced gait in paralysed subjects the need for more sophisticated stimulators arose. This resulted in the development of hardwired TTL or CMOS logic controlled stimulators (Strojnik et al., 1979). With these designs simple stimulation patterns for restoring these functions could be

corrected more efficiently. The subject initiates the stimulation sequences by hand switches. However the programming of the stimulation sequences is quite complicated and thus the performance and use of these devices is limited.

3.2. COMPUTER CONTROLLED LABORATORY FNS SYSTEMS :

The limited programmability of the above-mentioned traditional FNS systems lead to the development of computer or microprocessor controlled designs. During the early 80's several microprocessor based systems appeared in the literature. Some of the research groups concentrate on programmable portabe units, which will only be mentioned briefly as they are not relevant in the context of this thesis. Bogataj et al., (1984) presented a portable six channel stimulator and discussed its use in assisting the gait of hemiplegic patients. The appropriate stimulation sequences were stored in the stimulator and could be modified after evaluating the patients gait performance on a walkway. Kobetic et al., (1984) described a programmable portable 32 channel unit which is used in conjunction with percutaneous electrodes for FNS induced walking. Stimulation frequency is fixed to 25 Hz in all channels. The amplitude of the stimulating pulses is current controlled and constant. Only pulse width is varied for changing stimulation levels. The necessary stimulation patterns are developed on a DEC MINC-11/23 minicomputer and can be stored in the EPROM (erasable programmable read only memory) of the portable unit.

Petrofsky et al., (1984a) described a Z80 microprocessor based system for FNS induced 'closed loop controlled' walking. Closed loop control meaning, that certain parameters resulting from the system performance are used to control the effect of

stimulation. For example measures of the actual hip, knee and ankle joint angles are taken into account for synchronization of the stimulation sequences. An eight channel analogue to digital converter (ADC) is used to sense the joint angles. Stimulation amplitude is controlled via 10 digital to analogue converters (DAC). Stimulation frequency and pulse width are hardware programmable and equal in all channels. The stimulation pulses are isolated via pulse transformers and the current is amplified by high voltage transistors supplied from an isolated 300 volt DC power supply. The software for defining the stimulation patterns and for the actual stimulation control is written under the operating system CPM in machine language (ASSEMBLER) modules. The system has been tested successfully in a laboratory environment. However the authors stress that the algorithms used allowed only limited movement for locomotion in man, and that the system has to be improved particularly with regard to the feedback control mechanisms.

Petrofsky also described a FNS system for aerobic training of paralysed muscles (Petrofsky et al., 1985a) . The hardware used is very similar to the above-mentioned system, however some improvements over earlier designs have been added to the stimulator. For example, in the new system stimulation amplitude profiles are used to control the different muscle groups instead of the pure on/off control. These profiles are varied according to the pedal position of the exercise trainer. In addition the computer controls the maximum settings for the workload imposed on the patient.

Eichhorn et al. (1984) described a bicycle ergometer for the training of denervated muscles. With this device the extensors and flexors of the upper legs are stimulated according to the

pedal position via transcutaneous electrodes. In addition the computer is provided information about the strain in the pedal chain which in turn depends on torques produced by the subject. During the first 360 degrees of the crank rotation the system associates step by step the produced forces with their corresponding stimulation parameters. This automatic process constitutes a real advance for a faster procedure for defining stimulation sequences and optimal stimulation parameters. Unfortunately this approach is limited to FNS applied to well defined rotating movements, where joint positions and other variables are predictable.

Thrope et al. (1985) described a computer-controlled 9 channel stimulation system for FNS in laboratory use. This system is based on a PDP 11/23 computer and customised hardware modules for gathering feedback signals and generating the stimulation waveforms. Stimulation levels are varied by changing the pulse width at a constant current amplitude. The frequency of the stimulating pulses can be adjusted to the required recruitment levels at any time and in each channel independently. Furthermore the computer processes external command inputs (eg. joint position, myoelectric signals, switch positions) for specifying stimulation parameters. System operation has been streamlined for ease of use in the clinical laboratory, ie. the definition of stimulation sequences is simplified by use of graphics routines so that even users with a purely clinical background have no difficulties in working with the system.

Meadows and McNeal (1984) described the laboratory FNS system of the Rancho Los Amigos research group for modulated control of the lower extremeties. The design philosophy of this system is

similar to that of Thrope et al. (1985). However, there are differences in the computer hardware and especially in the design of the output modules which generate the stimulation waveforms. A Cromemco System One microcomputer (S-100 bus,Z80 microprocessor) acts as a host processor which governs all input/output functions of the system. For example, it generates interactive video graphics for use in the definition of stimulation envelopes and other paramters of stimulation. It also generates data lookup tables corresponding to those stimulation parameters so determined and controls data aquisition, data storage and communication with a slave processor. The slave processor also employs a Z80 microprocessor and is responsible for controlling the specially designed output modules. Each of these output modules is programmable via five 12 bit wide data latches, and six sixteen bit timer latches. The twelve bit data latches hold the amplitude information for each of the five phases of a biphasic stimulation waveform; the six sixteen bit timers define the timing of the generated waveform. In addition, a vectored interrupt may be generated at the conclusion of any of the timing intervals which are controlled by the timers. Once the slave processor has stored all the information needed for waveform generation it only has to write to a defined memory location to trigger a stimulus pulse set to occur. Characteristics of stimulation pulses can easily be changed just by writing new data to any of the above-mentioned data latches. Each output module including an isolation amplifier resides on a single S-100 bus card and can be connected to a pair of cutaneous or percutaneous electrodes.

Control sequences stored in the host microcomputer are easily altered. The stimulation envelope of the channel to be changed is displayed on a graphics terminal screen, and a desired change is

drawn on the screen by moving the screen cursor with a Houston Instruments Hi Pad digitizing tablet. A hard-copy of the control sequence can be obtained with a graphics printer.

3.3. SUMMARY FNS STIMULATION SYSTEMS

Hardwired stimulators for FNS are commonly used to restore simple functional movements. However to optimize the results of FNS in conjunction with more complicated movements like bicycling or walking more sophisticated microprocessor based stimulators are neccessary. With these systems it is possible to meet the individual needs of each subject and to minimize the time spend to accomplish this task.

Technical details of the described stimulators are discussed in the respective sections of chapter four.

SECTION III : METHODS, HARDWARE, SOFTWARE

Chapter 4. DESIGN OF THE FNS-CONTROLLER

4.1. GENERAL DESIGN CONSIDERATIONS

4.1.1. THE FNS CONTROLLER SYSTEM

The FNS-controller can be defined as a complex system which performs various input, output and internal or control operations. Once identified these system functions can be subdivided into suitable modular blocks, such as hardware and software components, which will be described in detail. From the literature the following functions of a FNS-controller for FNS induced cycling can be identified (see figure 4.1) :

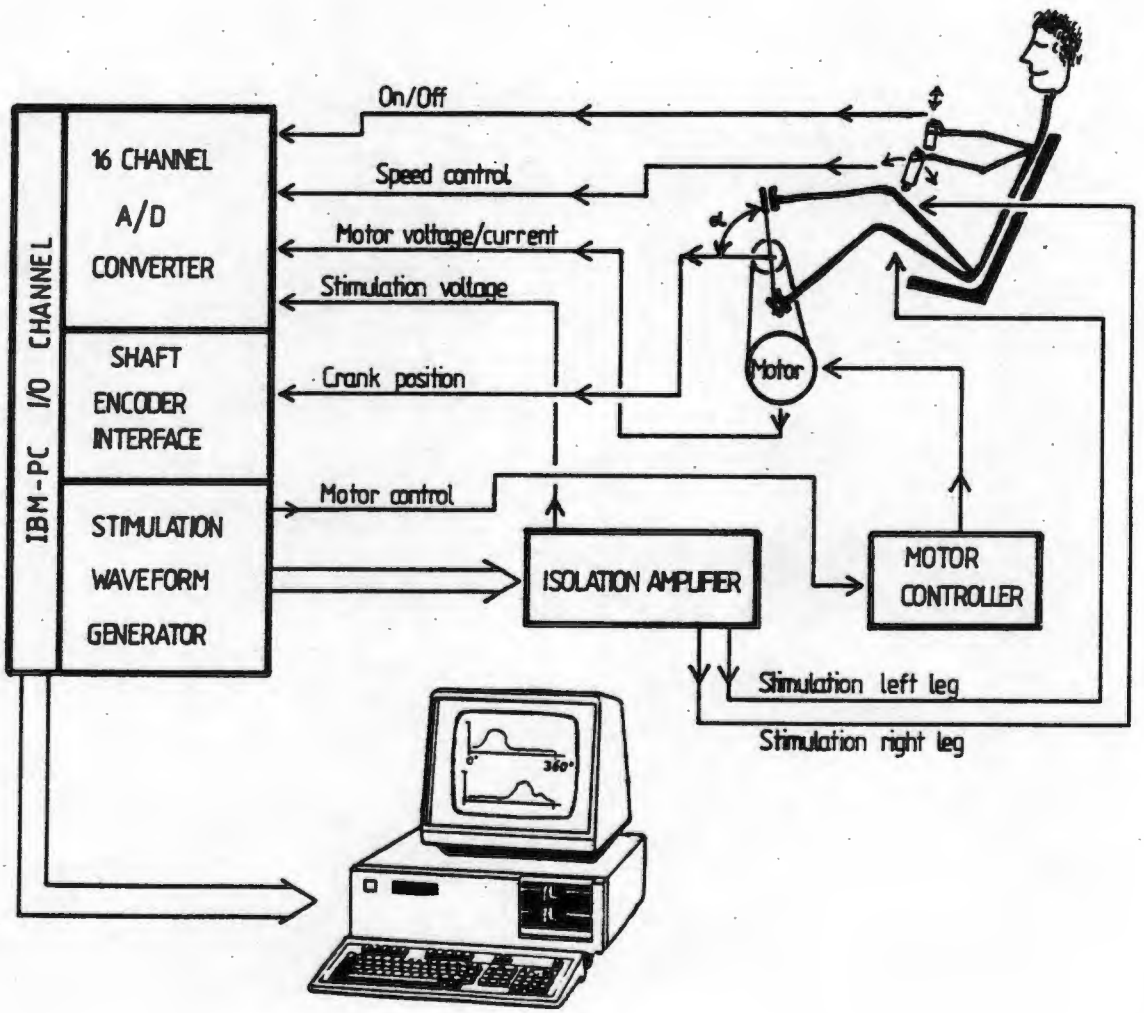


Figure 4.1 FNS-controller and Paracycle

I. Output Operations :

a.) Monophasic and biphasic pulsatile waveforms with pulse currents of up to 100 mA into a 1000 Ohms load are required (Bajd et al., 1983; Baker, 1981; Glaser et al., 1983; Kralj et al., 1980; Petrofsky et al., 1984a).

b.) The shape of the waveform, its frequency, pulse amplitude and pulse width must be individually adjustable for each channel and for each subject.

c.) Information for estimating the efficiency of stimulation must be available to the system operator during the stimulation process.

d.) Individual stimulation sequences need to be stored, preferably on floppy disk, to simplify system operation.

II. Input operations :

a.) The definition of stimulation parameters and individual stimulation sequences: As the latter are mostly determined on a trial and error basis, the system operator must be able to quickly and easily perform the necessary changes between test runs.

b.) The acquisition of information (eg. the crank position on the paracycle) which coordinates the stimulation sequence and levels.

c.) Provisions for collecting additional positional or force information for closed loop controlled FNS must be made.

III. Control functions :

a.) The information relating to stimulation parameters and sequences must be processed to allow real time control of stimulation.

b.) Facilities for stimulation data storage and retrieval management must be provided.

c.) A user friendly working environment must be provided for the system operator.

4.1.2. SPECIFIC DESIGN CONSIDERATIONS FOR THE FNS CONTROLLER

Before dividing the above-mentioned system into functional blocks, it is important to consider the special needs of the envisaged FNS-controller. The proposed controller will primarily be used in conjunction with the paracycle for FNS induced cycling. In order to confirm the results of other FNS research groups and hence gain experience, the FNS-controller must be sufficiently flexible to produce most of the commonly used stimulation waveforms and sequences for FNS using surface electrodes.

Thus following design criteria were defined :

a.) The FNS-controller is to be used as a research tool and must thus be able to produce most of the stimulation waveforms presently used for FNS (ie. biphasic and monophasic waveforms with control of stimulation level via pulse width and/or amplitude modulation).

b.) The system must be expandable to at least 12 channels of stimulation.

c.) The system must provide at least 8 analogue input channels for positional feedback and stimulation control.

d.) The system must support the use of a motor either to assist the motion or to act as a resistive brake.

e.) The system software must be easily adaptable to FNS applications other than cycling.

f.) The cost of the FNS-controller should not exceed R 7000 (ie. the total operational budget).

4.1.3. THE FNS CONTROLLER

The above design criteria precluded the implementation of a purely hardware programmable stimulation system, and it was thus decided to develop a multipurpose FNS-controller . Similar approaches have been followed by Thrope et al. (1985) using a minicomputer and by Meadows and McNeal (1984) using a multiprocessor system. However, due to financial restrictions, the design of the FNS-controller was based on an IBM Personal Computer (IBM PC).

There are several advantages in using an IBM or compatible Personal Computer for this purpose. Firstly, a Personal Computer represents a complete system including disk based data storage, graphics capabilities and access to a huge reservoir of software at a reasonable cost. Secondly, the open, well known and expandable architecture of the IBM PC, allows the user to design his own customized expansion boards. And thirdly the IBM PC currently represents an industrial standard, thus simplifying future expansions of the hardware and/or software. The SPERRY Personal computer was chosen as it has a 40 % speed advantage over the IBM original, and was also available at a lower price.

Four customized modules are linked to the SPERRY PC (see Figure 4.1) to perform the functions described earlier in this chapter. These modules are described below.

1. THE STIMULATION WAVEFORM GENERATOR:

The stimulation generator consists of an expansion board for the PC. It houses six intelligent output modules (ie. six stimulation channels) which can be programmed to produce various stimulation waveforms. These modules operate independently of the PC's processor, which is thus free to perform other control tasks. The stimulation can easily be expanded to 12 channels by the addition of another stimulation generator board.

2. THE 16-CHANNEL A/D CONVERTER:

A fast 16 channel 12 bit A/D converter allows the measurement of electrode impedance, motor current and voltage and the input of other analogue control signals from the patient or operator. It is contained on an expansion board for the PC.

3. THE SHAFT ENCODER INTERFACE:

The crank position of the paracycle is sensed using a Hewlett Packard incremental shaft encoder, interfaced to the bus of the PC. This interface, which consists of an expansion board for the IBM PC allows measurement of the absolute crank position and speed. Additionally an eight bit parallel port is provided on the board. This is used to control the motor brake/drive circuitry and also allows the input of binary signals (eg. switch settings).

4. THE MOTOR CONTROLLER:

A programmable pulse width modulator (PWM) is used to control a 24 V DC motor which either assists or loads the patient during cycling.

A fifth module, namely the isolation amplifier, is also provided. This module amplifies the signals from the waveform generation module to a level suitable for stimulation via surface electrodes. It also serves to isolate the patient from any mains operated equipment.

4.2. THE HARDWARE MODULES OF THE FNS CONTROLLER

4.2.1. THE IBM COMPATIBLE SPERRY PERSONAL COMPUTER

The system unit of the SPERRY personal computer has the following features : a system board with seven expansion slots, the INTEL 8088-2 microprocessor (running with a clock frequency of 4.77 or 7.16 MHz), 48 KByte Read only memory (ROM), 128 KByte Random access memory (RAM), and two 360 Kbyte double sided double density disk drives. A power supply is included to supply DC voltages to the system board and the disk drives. Four of the seven expansion slots are occupied with the following expansion boards :

a.) a memory expansion card fitted with an extra 256 Kbyte of dynamic RAM memory;

b.) a parallel printer adapter;

c.) an IBM compatible colour graphics card (200 x 640 pixels resolution);

d.) a floppy disk controller card.

The remaining three expansion slots are occupied by the customized FNS-controller expansion boards:

e.) the 16-channel 12-bit analogue to digital converter (ADC);

f.) the stimulation waveform generation board;

g.) the interface for the shaft encoder.

Further details regarding the hardware of the SPERRY personal computer are given in the IBM technical reference manual (IBM,1983) and the appendix A of this thesis.

4.2.2. THE STIMULATION WAVEFORM GENERATOR

The characteristics of stimulation waveforms have been described in chapter 2.4. . Figure 4.2. shows a typical example of a biphasic, charge balanced waveform. This waveform can be defined by the following parameters :

- A_+,A_- : amplitude of the positive and negative going pulses respectively;
- t_+,t_- : duration of the positive and negative going pulses respectively;
- IPI : time interval between the onset of the negative and positive pulses (ie. the interpulse interval);
- T : time interval between two negative pulses;

A monophasic waveform can be derived from a biphasic one simply by setting the duration and/or the amplitude of the positive going pulse to zero.

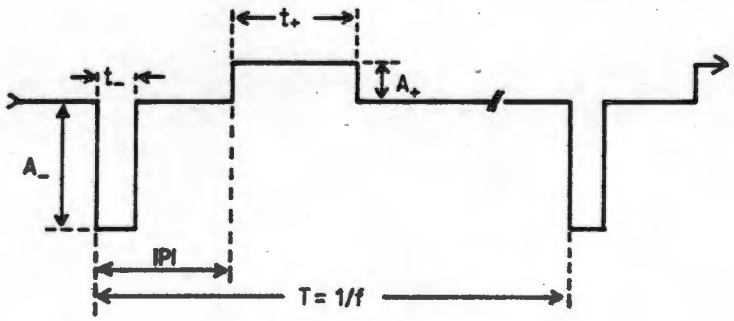


Figure 4.2. Typical Biphasic FNS waveform

In a FNS-controller designed primarily as a research tool, all of the waveform parameters mentioned should ideally be under software control in the ranges given in Table 4.1. This specification has only been met by Meadows and McNeal (1984), whose system can additionally control the amplitudes stimulating pulses. The system presented by Thrope et al. (1985) provides software control of only the duration of the negative pulse and the duration between two negative pulses. All other parameters can be changed only by hardware controls. Similarly, the system used by Petrofsky et al. (1984a, 1985a) has software control of the amplitude of the negative going pulse, with all other parameters being hardware programmable. In the FNS controller described here, all of the above-mentioned waveform parameters are software controlled.

Table 4.1 RANGES OF WAVEFORM PARAMETERS

	Minimum	Maximum
A_+	0 V (ie. 0 mA output)	+ 5 V (ie. +100mA output)
A_-	0 V (ie. 0 mA output)	+ 5 V (ie. -100mA output)
t_-	0 µs	800 µs
t_+	0 µs	4000 µs
IPI	t_-	T / 2
T	10 ms (ie. f=100Hz)	100 ms (ie. f=10Hz)

Note, that the output signal of the waveform generator board is passed to the input of the voltage controlled current isolation amplifier.

A simple design approach for generating a FNS waveform using a microprocessor based system is to have the processor directly controlling one stimulation channel by means of a digital I/O port and a digital-to-analogue converter. The timing of the generated pulse is then managed by software timing loops, and the pulse amplitude by the digital-to-analogue converter. The generation of a complex biphasic waveform for several stimulation channels would, however, demand a great deal of the microprocessor's attention. To free the processor to perform other important control tasks, it was decided to design special modules for the generation of the stimulation waveforms. This also enabled a high level language to be used for the stimulation software rather than the ASSEMBLER routines which would have otherwise been required.

A practical design approach to control a certain time interval in a microprocessor based system is the use of programmable timer chips. To generate a biphasic waveform similar to that shown in Figure 4.2, four time intervals and two amplitudes must be controlled. Unfortunately INTEL's 8253/8254 programmable timer houses only 3 separate timer blocks, so more than one of these devices would be needed for one stimulation channel. It was thus decided to use the Motorola 6840 timer chip. Although this chip also contains only three separate timer blocks, it allows the generation of a square wave with a variable duty cycle. Thus the two time intervals between the negative and positive going pulses can be controlled with only one timer block, the other two timing blocks being free to determine the negative and positive pulse durations.

The block diagram of a single waveform generation module is illustrated in Figure 4.3. Timer 2 generates a square wave with a

programmable frequency and duty cycle. Timers 1 and 3, which are programmed as single shots with programmable pulse durations, produce the positive and negative pulses respectively. These

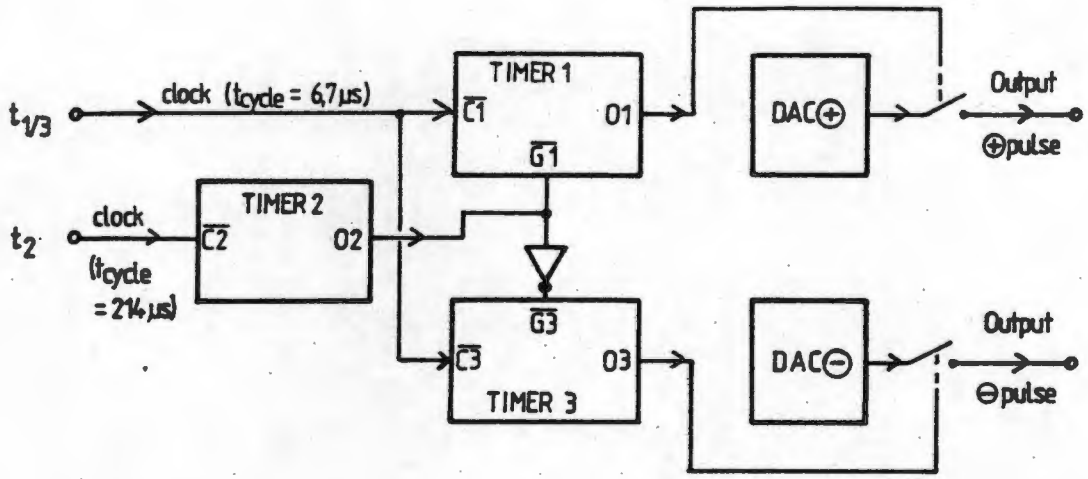


Figure 4.3. Block diagram waveform generation module

single shots are triggered by a negative transition on their gate inputs. Timer 3 is triggered by the inverted output of timer 2 and thus generates a pulse triggered by a positive transition of timer 2. Figure 4.4. shows the timing diagram of the waveform generation. To produce a typical biphasic stimulation waveform, the signal of the output of timer 3 is inverted and summed to the output of timer 1. This is done in the isolation amplifier. The amplitude of both the negative and positive pulses can be varied independently using two 8-bit digital-to-analogue converters (DAC).

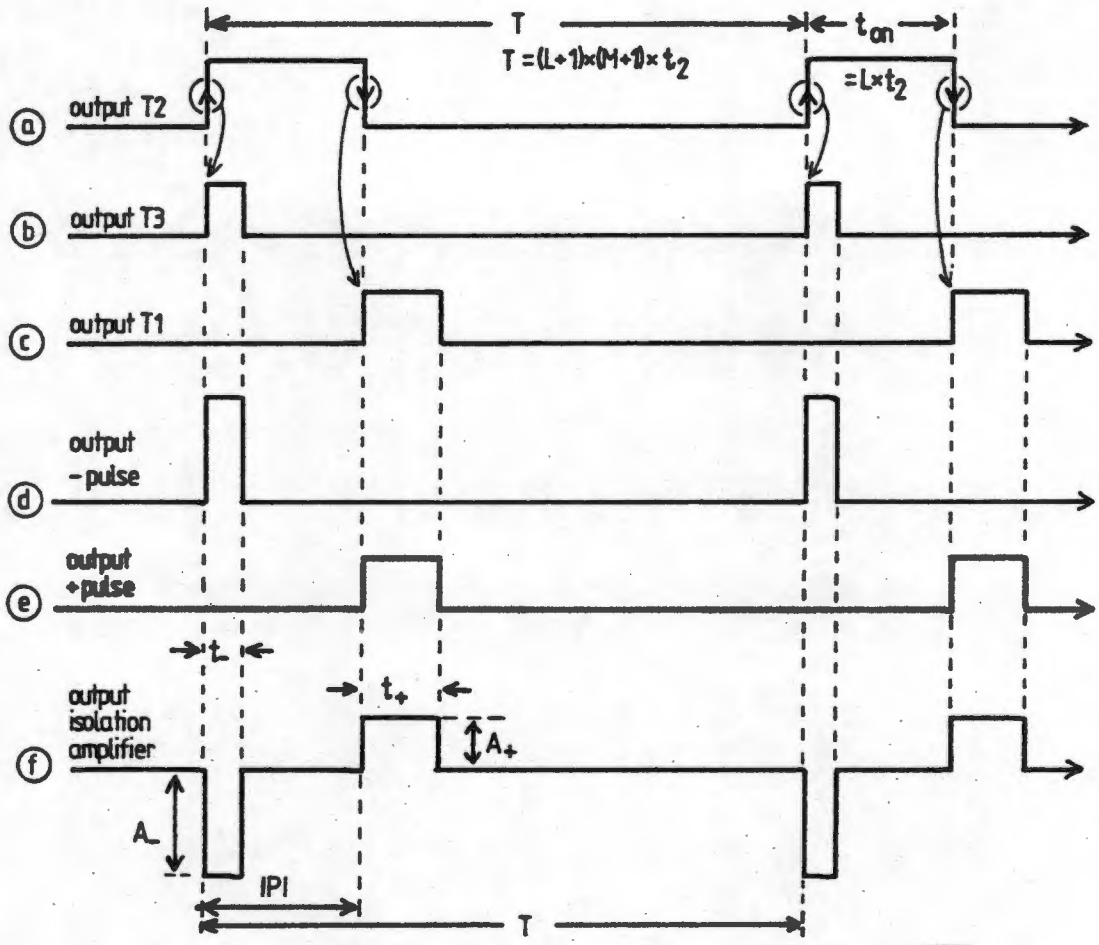
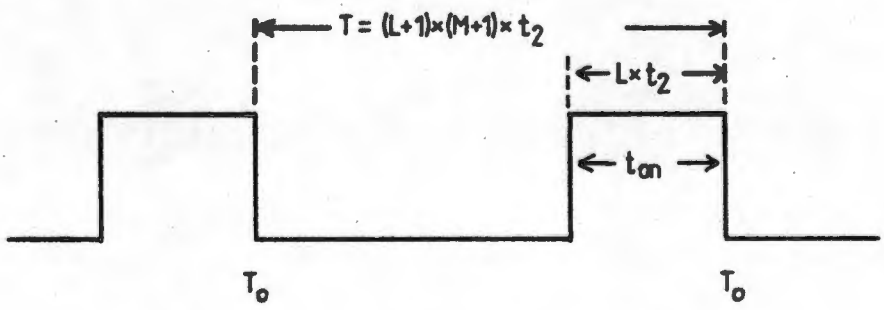


Figure 4.4. Timing diagram of waveform generation

OPERATION OF TIMER 2

The principle of generating a software controllable square wave with variable duty cycle is shown in Figure 4.5.



Continuous operating mode with variable duty cycle

Figure 4.5. t_2 = time period of the external clock
L = 8-bit number loaded in LSB counter latch
M = 8-bit number loaded in MSB counter latch
T_o = Counter time out (all zero condition)

The on-time of this square wave generated by timer 2, and thus the IPI of the stimulation waveform (see Figure 4.4f)is defined as:

$$t_{on} = L * t_2 \; ; \qquad (I)$$

where t_2 is the period of the external clock frequency.

The period T of the square wave generated by timer 2, and thus the period of the stimulation waveform (see Figure 4.4f) is defined as :

$$T = (L+1) * (M+1) * t_2 \; ;$$

where L and M are Byte variables ranging from 0 to 255;

thus the frequency f of the stimulation waveform is :

$$f = 1 / ((L+1) * (M+1) * t_2) \; ; \qquad (II)$$

The period t_2 of the external clock of timer 2 is 214 µs (see Appendix B for details). Thus, the time t_{on} (ie. IPI of the stimulation waveform) and the frequency f can be varyied in the following ranges :

t_{on} : 214.7 µs ---> 4000 µs ;
f : 9 Hz ---> 100 Hz ;

From equation II it can be seen that the frequency of the stimulation waveform is dependent on both the value of L and M. Thus the increment with which the frequency can be adjusted is dependent on the value of L. For L = 1 the change in frequency by changing M is very small. However, when L is increased, the resulting minimum change of frequency is greater. For a given t_{on}

(and thus L) the frequency f of timer 2 can be changed by varying the value of M. Only discrete values of frequency are possible. If the target frequency f lies half way between two such values for f, the resulting error will be maximum, and can be expressed by the following relationship :

$$f_{error}\ (t_{on},f,t_2) = 1/2*100\% * (f*(t_{on}+t_2/(1+f*(t_{on}+t_2))); \quad (III)$$

The deduction of this formula and plots of the error function are given in appendix B. From these plots it can be seen that the error rises with f and t_{on}. However, this error is limited to approximately +- 10% for f ranging from 10 to 65 Hz and t_{on} ranging from 0.2 to 3.7 ms .

OPERATION OF TIMER 1 AND 3

Timer 3 is triggered by the positive transition and timer 1 by the negative transition of the square wave generated by timer 2 (see figure 4.4d and 4.4e). Triggering takes place via the gate inputs G (see figure 4.3). Both timers are programmed as single shots in a 16-bit mode and use an external clock frequency with a cycle period of 6.7 µs (see appendix B for further details). This clock is derived by dividing the 4.77 MHz system clock by 32. Thus the minimum pulse width, which is also equal to the minimum pulse width increment $t_{1/3}$, is 6.7 µs. The value N which must to be loaded into the 16-bit timer latches to obtain the required pulse width, can be calculated from the following relationship :

$$N = \text{pulse width} / t_{1/3} \qquad (IV)$$

where $t_{1/3}$ = 6.7 µs;

For the FNS-controller the maximum pulse widths are 800 µs for timer 3 (negative pulse) and 4000 µs for timer 1 (positive pulse) .

Timers 1 and 3 can either be triggered by the output of timer 2, as described above, or by a 'write to latches' command, ie. the pulse generation is triggered by a software command to the respective timer. This feature, which allows the generation of special FNS waveforms under the control of software timing loops, is used to generate test pulses for electrode impedance checking. Further details are given in appendix B and H.

PULSE AMPLITUDE CONTROL

For maximum flexibility of the FNS-controller, the amplitude of both the negative and positive pulses can be varied independently (see figure 4.3). Two 8-bit digital-to-analogue converters (DACs) are used to generate two programmable voltage sources which are adjustable between 0 and 5 volts. The outputs of the DACs are connected to analogue CMOS switches which are controlled by the outputs of timers 1 and 3.

MECHANICAL CONSTRUCTION

Six stimulation modules and two additional 8-bit DACs are assembled on a full length expansion board for the SPERRY personal computer. The two additional DACs are provided for the control of the motor voltage or other control tasks. All outputs are accessible via a 15 pin D-connector at the rear panel of the board.

SPECIFICATIONS OF THE STIMULATION WAVEFORM GENERATION BOARD

The characteristics of the stimulation waveform generation board are summarized in the following seven points :

1. six independent programmable stimulation channels;
2. biphasic or monophasic waveforms;
3. pulse repetition frequency adjustable between 10 and 100 Hz;
4. inter pulse interval adjustable between 214 µs and 5000 µs in steps of 214 µs;
5. negative and positive pulse amplitudes independently adjustable from 0 to 5 V in 255 steps. This corresponds to positive and negative stimulation current amplitudes between 0mA and 100 mA, adjustable in steps of 0.39 mA;
6. negative and positive pulse widths independently adjustable from 0 to 5000 µs in increments of 6.7 µs.
7. Two additional 8-bit D/A converters with unipolar output voltage range from 0 to + 6 Volts.

4.2.3. THE ISOLATION AMPLIFIER

The isolation amplifier is a separate unit which is connected to the waveform generation board and to the A/D converter. It has to fulfill the following functions :

a.) combine the two monophasic pulses from the waveform generation board into a biphasic (or monophasic) stimulation waveform;

b.) act as a voltage controllable current source, which is able to sink at least 100 mA into a load of 1.5 KOhm;

c.) amplify the waveform with a low level of distortion;

d.) achieve total isolation between all stimulation channels and

ground, thus avoiding interference between adjacent channels and ensuring the patient's safety.

e.) limit the output voltage and current to safe values;

f.) measure the electrode impedance before or during stimulation;

From the literature, two design techniques emerge. The first of them uses optocouplers and high voltage amplifiers powered by DC/DC converters or batteries (Strojnik et al., 1979; Thrope et al., 1985; Nauman et al., 1985; Petrofsky et al., 1984a). The second technique makes use of low voltage power amplifiers to drive high voltage pulse transformers (Petrofsky et al., 1985a).

A design using opto-couplers for isolation would consist of the following sections:

a.) an amplifier which combines the two unipolar pulses from the waveform generation board to form a bipolar waveform;

b.) the opto-coupler isolation stage;

c.) a high voltage push-pull current amplifier capable of sinking 100 mA into 1500 Ohms load;

d.) safety circuitry to disconnect the electrodes from the output in case of hardware failures;

e.) a separate DC/DC converter to produce the supply voltage of +- 100 Volts for the output stages.

The drawbacks of this approach are the amount of hardware necessary for one stimulation channel and also the inherent safety problems arising from possible hardware failures in the output stages. It was thus decided to follow the second design approach using a pulse transformer for isolation and voltage amplification. Figure 4.6 shows the block diagram of one channel of the isolation amplifier.

The incoming signal is combined into a biphasic waveform by a differential amplifier and then current amplified by a push-pull power amplifier.

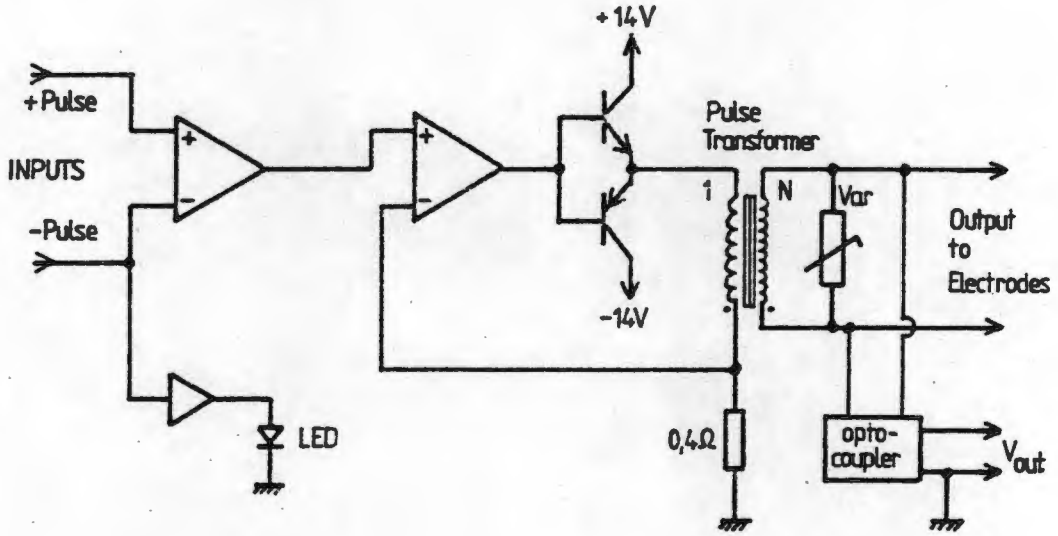


Figure 4.6 Isolation amplifier (one channel)

The current of the stimulation pulse in the secondary winding of the transformer is sensed by the voltage drop over the 0.4 Ohm resistor on the primary side. This voltage is used in a negative feedback circuit to maintain a constant current output despite changes in the electrode impedance. The output voltage on the secondary of the transformer is sensed via an opto-coupler to allow measurement of electrode impedance, even during stimulation. The pulse transformer consists of a 44 mm Siemens Sifferit Ferrite E-core. The core size was determined by the maximum pulse width which is to be transmitted through the transformer without core saturation (see appendix C for details). The turns ratio is 1:53 (primary 38 turns, secondary 2000 turns). A ferrite core was chosen primarily because of its high resistivity which keeps eddy currents to a minimum and thus

insures low distortion of short pulses. Primary and secondary windings are wound on top of one another to keep leakage inductance and thus the pulse rise time small. The windings are separated by insulating material to achieve maximum insulation. The output voltage is limited by a Varistor Var (ie. a voltage dependent variable resistor) to approximately 200 Volts.

This current-amplifier and transformer combination delivers 100 mA pulses with a maximum width of 820 us into a load of 1500 Ohms. Using an output current of only 20 mA, a maximum pulse width of approximately 4000 µs is possible before core saturation occurs. The output current is constant within 10 % if the electrode impedance changes from 300 to 1500 Ohms. The pulse slew-rate is typically 15 V/us. A LED (light emitting diode) is provided to signal that the respective channel is active.

Each isolation amplifier is assembled on a single EURO CARD, of which six fit into a 19 inch rack. The power supply consists of a 60VA ferrite ring core transformer. The primary and secondary winding of the transformer are wound separately onto the ring core to ensure maximum isolation. The supply voltage is regulated to +- 14 volts.

The isolation amplifier was tested using procedures of the IEC for medical electro-stimulation equipment (see appendix C for details).

4.2.4. THE ANALOGUE TO DIGITAL CONVERTER (ADC) BOARD

As shown in figure 4.1, the ADC board is used to measure a number of control variables and to sense the voltage supplied to the electrodes in order to check (calculate) the electrode impedance. The design of the ADC board is based on a successive approximation analogue-to-digital converter chip (AD-574A). This device contains a complete 12-bit ADC, voltage reference and bus interface in a 28-pin dual-in-line package. The typical conversion time is 25 µs (Popp, 1986). Figure 4.7 shows the block diagram of the ADC board.

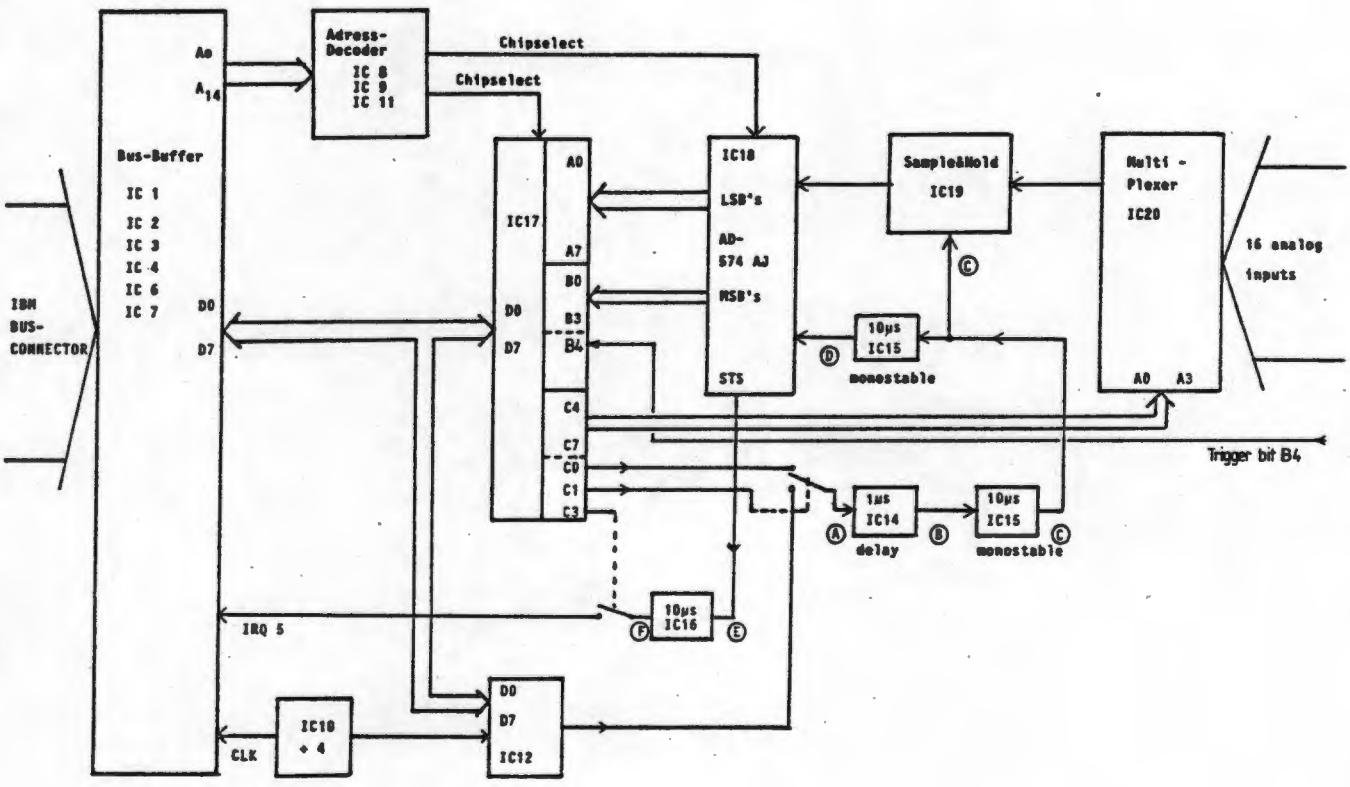


Figure 4.7 Block diagram 12-bit ADC board

In order to keep the load imposed on the bus as small as possible, all bus lines to and from the personal computer are buffered. The address decoder detects the I/O port address of the ADC board which can be selected as either 700 or 780 Hex. The AD 574A chip, the lines for controlling the conversion process and the channel selection of the multiplexer are interfaced to the 8-bit IBM data bus with the 8255-5 INTEL programmable parallel interface (PPI) chip (IC 17). To capture the data from 16 independent analogue channels, a CMOS analogue multiplexer is used (IC 20). Input voltages may range in the monopolar mode between 0 and + 10 Volts, and between - 5 V and + 5 Volts in the bipolar mode. Because the ADC takes about 25 µs to digitize an analogue input, changes in this signal during conversion can result in errors so that the final digital value would not accurately represent the data level prevailing at the instant at which the conversion command was transmitted. A sample and hold chip (IC 19) is therefore introduced into the link between the multiplexer and the ADC chip. This device makes a fast aquisition (within 6 µs) of the varying signal and holds the signal during conversion.

Conversion can be initiated by the driving software (software clock) or by a programmable interval timer (IC 12). In the latter case, the end of conversion signal of the ADC generates an interrupt to the personal computer (interrupt controlled sampling). The advantage of the software clock is its higher maximum speed (22 KHz as compared to 11 KHZ in the interrupt mode), because the interrupt handling of the 8088 microprocessor is relatively slow. However, the exact conversion rate depends on the execution speed of the user program which is a function of the clock frequency of the microprocessor. While the clock

frequency of the IBM Personal Computer is 4.77 MHz, other Personal Computers differ considerable (eg. the SPERRY Personal Computer runs with a clock frequency of 7.16 MHz). The programmable interval timer and the hardware interrupt mode avoid this problem as the clock frequency of the interval timer is the same for all IBM compatible Personal Computers. Another advantage of interrupt-controlled sampling is that at low sampling rates the processor of the personal computer is not tied up in timing loops and therefore can perform other tasks at the same time.

The ADC board is assembled on a double sided through-hole plated printed circuit board which plugs into one of the expansion slots of the SPERRY Personal Computer. The 16 analogue inputs, ground and power lines are accessible at a 25-pin D-connector on the rear panel of the PC board.

4.2.5. THE SHAFT ENCODER INTERFACE CARD

An 8-bit Hewlett Packard incremental shaft encoder is used to sense the position and speed of the crank of the paracycle. This device was selected because of its accuracy, long term stability and life expectancy which makes it far superior to 360 degree potentiometers. Measurement of absolute crank position is made possible by hardwired logic circuitry using up-and-down counters. The logic is interfaced to the PC's bus with a programmable parallel interface. Position is coded in an 8 bit word giving a resolution of 1.406 degrees. A measure for crank speed is available once per revolution as another 8-bit word. Both measurements can simply be accessed by the Personal Computer by reading the respective 8-bit ports. Additionally eight binary input/output lines are available on the shaft encoder interface

for other control purposes. The shaft encoder interface board is assembled on a double-sided printed circuit board which plugs into one of the expansion slots of the SPERRY Personal Computer.

4.2.6. THE MOTOR CONTROLLER

This device controls a 24 Volt wheelchair motor which is connected via a gearbox and chains to the pedal crank of the paracycle. The motor can be used to drive the crank for passive motion of the patient's legs or to assist the patient in overcoming dead spots over the 360 degree crank cycle. Alternatively the motor can be used to impose a variable load at various crank positions.

The motor controller consists of a voltage controlled pulse width modulator. A power VMOS-FET is used as a power switch for the following reasons : (1) ease of interfacing this voltage controlled device to low power control circuits; (2) freedom from secondary breakdown makes VMOS-FET devices highly suitable for driving inductive loads such as motors; (3) high current ratings (DC 40 A continuous, 150 A peak); and (4) low drain/source ON-impedance of only 35 milli-Ohms results in high switching efficiency. A mechanical relay is used to switch from the driving mode to the braking mode. In the braking mode, the motor is used as a generator which drives a resistive load which can be varied by the same pulse width modulator. The generated energy is dissipated in an external resistor. The motor voltage is controlled by a voltage which may vary between 0 and + 6 Volts. The relay for switching between the drive and brake mode is controlled via a single TTL compatible line. Voltages corresponding to the actual motor voltage and current are made

available for feedback purposes to the computer (via ADC board see figure 4.1).

The motor controller is assembled on a single sided printed circuit board and mounted with the power switching devices and the relay in a separate box which is attached to the paracycle.

4.2.7. HARDWARE SPECIFICATIONS OF THE FNS CONTROLLER SYSTEM

The following is a list of the specifications regarding hardware performance and characteristics of the entire FNS-controller.

a.) six independently programmable stimulation channels;

b.) generation of biphasic or monophasic waveforms;

c.) pulse frequency programmable from 10 to 100 Hz;

d.) interpulse interval programmable from 214 to 5000 µs in increments of 214 µs;

e.) pulse amplitudes programmable from 0 to 100 mA in increments of 0.39 mA (current source);

f.) pulse widths maximum 820 µs at 100 mA pulse current into 1500 Ohms, or 4000 µs at 20 mA into 1500 Ohms load; (the maximum pulse width is due to the performance of the pulse transformer and not to the stimulation generation board);

g.) 16-channel 12-bit analogue-to-digital converter with a conversion time of 40µs;

h.) 8-bit shaft encoder interface for absolute measurement of crank position and speed;

i.) eight digital input/output lines

j.) motor controller for 200 watts 24 volts DC motor with brake/drive facility and measurement of motor voltage and current;

k.) provision for measurement of electrode impedance;

4.3. SOFTWARE FOR THE PARACYCLE FNS-CONTROLLER

4.3.1. INTRODUCTION

The hardware described in the previous section enables the user to control virtually all the parameters of a stimulating waveform independently for each channel. The software for the FNS-controller serves to support the system operator in his task of defining stimulation parameters and sequences and also allows real time control of stimulation. The software must be user friendly ie. the program must be self explanatory to the point that no extra information in form of lengthy manuals is necessary to run it. Furthermore the source code should be so designed that future expansions can be realized easily.

Similar prerequisites for software of a control system for FNS were defined by Thrope et al. (1985). Their software package, with the exception of a few real time processes, is written in a high level language (Fortran) on a PDP 11 minicomputer. Stimulation parameters are entered via the keyboard and graphics screen. Meadows and McNeal (1984) has described a host computer and slave processor system in which the host is responsible for the user interface (written in a high level language) and the slave processor controls the actual stimulation (using assembler routines). Stimulation envelopes are entered on the graphics screen using an XY digitizing tablet. Look up tables are calculated from these envelopes and are then used by the slave processor for real time stimulation. Petrofsky et al. (1985) has described a Z80 microprocessor based system for his 'aerobic FNS training device'. The program, which is written in assembler, uses stimulation patterns stored in EPROM. The system senses the

crank position and velocity as well as the speed control of the patient to control the stimulation. In addition, subject specific physiological characteristics and limits are stored in an EEPROM.

Based on these reports, it was decided to develop the software using a high level language and, if necessary, to use assembler subroutines for real time processes where speed is essential. The high level language to be used should have the following features :

- High resolution graphics capabilities to support the operator in drawing stimulation envelopes and to improve data presentation on the screen.
- Medium to high execution speed for fast and convenient data processing.
- A reasonable program structure allowing use of multipurpose program modules. This enables future software changes to be made easily.

On the basis of past experience, the programming languages Basic and Fortran are not suitable with respect to the above requirements. The disadvantage of Basic is the slow speed and lack of structure whereas Fortran lacks the required graphics routines. However, TURBO PASCAL, in conjunction with the TURBO GRAPHIX TOOLBOX Utilities of Borland International Inc., seemed capable of meeting all the requirements. TURBO PASCAL closely follows the definition of Standard Pascal and is extended in respect of graphics procedures, the access to hardware ports and other special features related to the IBM Personal computer (Borland 1985a,b). In general, TURBO PASCAL allows the development of highly structured programs in which the flow of

logic is relatively easy to follow. As a result, it should be reasonably easy for future users to understand the structure of the software package and to make necessary changes. Moreover, the execution speed of the compiled TURBO PASCAL program proved to be sufficient to allow even the real time stimulation processes to be written in this high level language.

The software can be devided into two main sections. The first section allows the definition of stimulation parameters and sequences and the storage of data while the second section controls the real time stimulation process. The functions of these two sections are described fully in this chapter. Examples of the screen prompts are given to illustrate the user guidance provided by the program. Programming details, a structure chart and the full program listings are given in the appendix H.

4.3.2 INPUT AND STORAGE OF STIMULATION PARAMETERS AND SEQUENCES

The complete Paracycle FNS system which is to be controlled by the software is shown in Figure 4.1

The stimulation of the various muscle groups is controlled according to the crank position and the controls operated by the subject. This means that stimulation sequences for each muscle group must be entered and stored in look up tables with reference to the crank position. Similarly the level of motor assist or load needs to be defined as a function of the crank position. Other stimulation characteristics such as frequency and the type of stimulation waveform must also be defined and stored. Additionally, personal patient data must be entered and stored together with these individual stimulation sequences for later reference.

DEFINITION OF STIMULATION ENVELOPES

As described in Chapter two, the level of muscular contraction can be controlled by the amplitude and/or the width of the stimulating pulses, and the pulse repetition frequency. This level of contraction must be defined for each muscle group with reference to the crank position. In the described software package, modulation of muscle contraction is accomplished by varying the pulse width at a fixed amplitude. The provision of additional software could allow both pulse amplitude and frequency to also vary as a function of crank position. However, this was not deemed necessary in the present system.

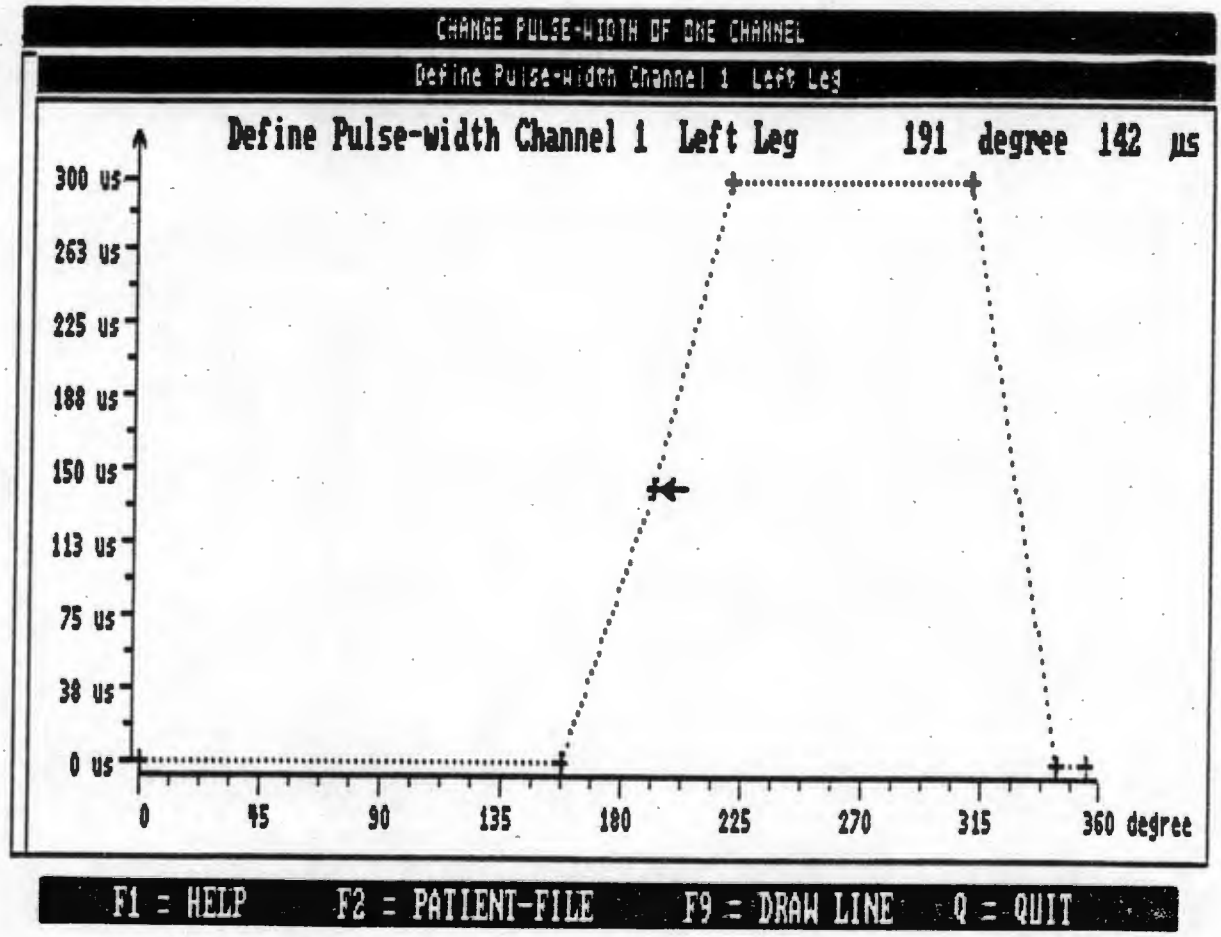


Figure 4.8 Screen display for pulse width envelope definition

Figure 4.8 shows the graphics screen for the definition of the pulse width envelope versus crank position for the first channel in the left leg (usually the Quadriceps muscle group).

The pulse width envelope is defined by placing discrete data points (the crosses) with a moving cursor (the arrow) on the screen. On the x-axis, the crank position is displayed ranging from zero to 360 degrees (where zero degree represents the same position as 360 degrees). The y-axis shows the chosen range of the stimulation pulse width, the maximum pulse width having been defined by the operator previously. The cursor can be moved using either the standard cursor control keys of the IBM keyboard or a mouse. The exact location of the cursor is displayed at the top right-hand corner of the display window. Data points are entered at a cursor position by pressing the Return key. Data points are deleted at the cursor position by pressing the Backspace key. The entered data points are joined with straight line segments after pressing function key F9. The entire patient file, containing previously entered data can be displayed by pressing function key F2 (see Figure 4.9). An online help facility is also available (function key F1).

Once the operator is satisfied with the defined envelope, he quits this screen by pressing the 'q' key. The program then gives the operator the option of copying this envelope, shifted 180 degrees to the right, to the right leg. (The stimulation patterns of the left and right leg can be assumed, as a first approximation, to be 180 degrees out of phase).

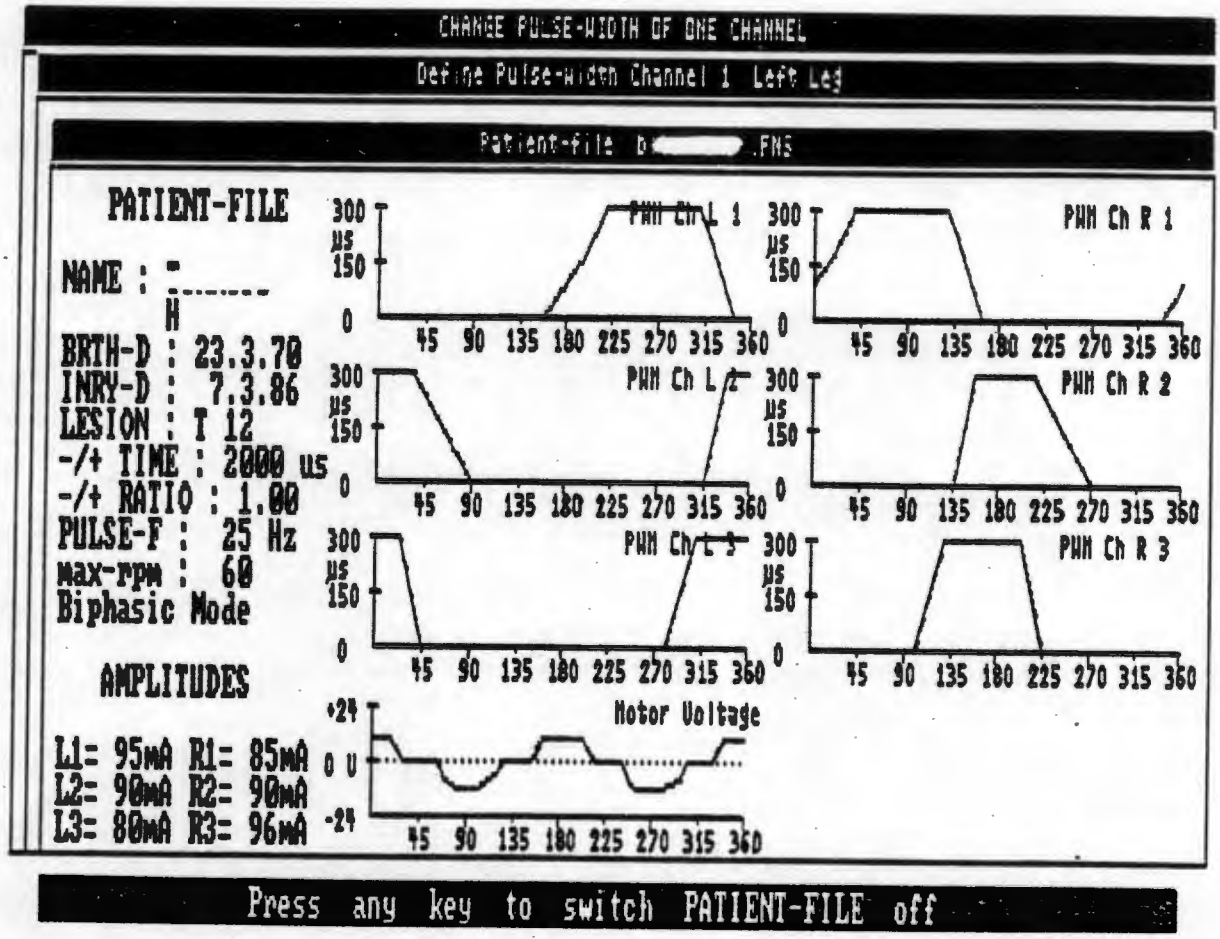


Figure 4.9 Display of patient file

DEFINITION OF THE MOTOR VOLTAGE ENVELOPE

The motor voltage is defined as a function of crank position in the same way as the pulse width envelope (Figure 4.10). Positive motor voltages indicate that the motor assists the patient in the cycling motion. Negative motor voltages lead to a loading of the patient, -24 Volts defining the maximum load. Thus it is possible to assist the subject in the dead spots of the crank cycle (for example, at 0 and 180 degrees crank position, where the subject can produce only limited torques to turn the crank). Similarly, a load can be imposed on the subject at 90 and 270 degrees crank

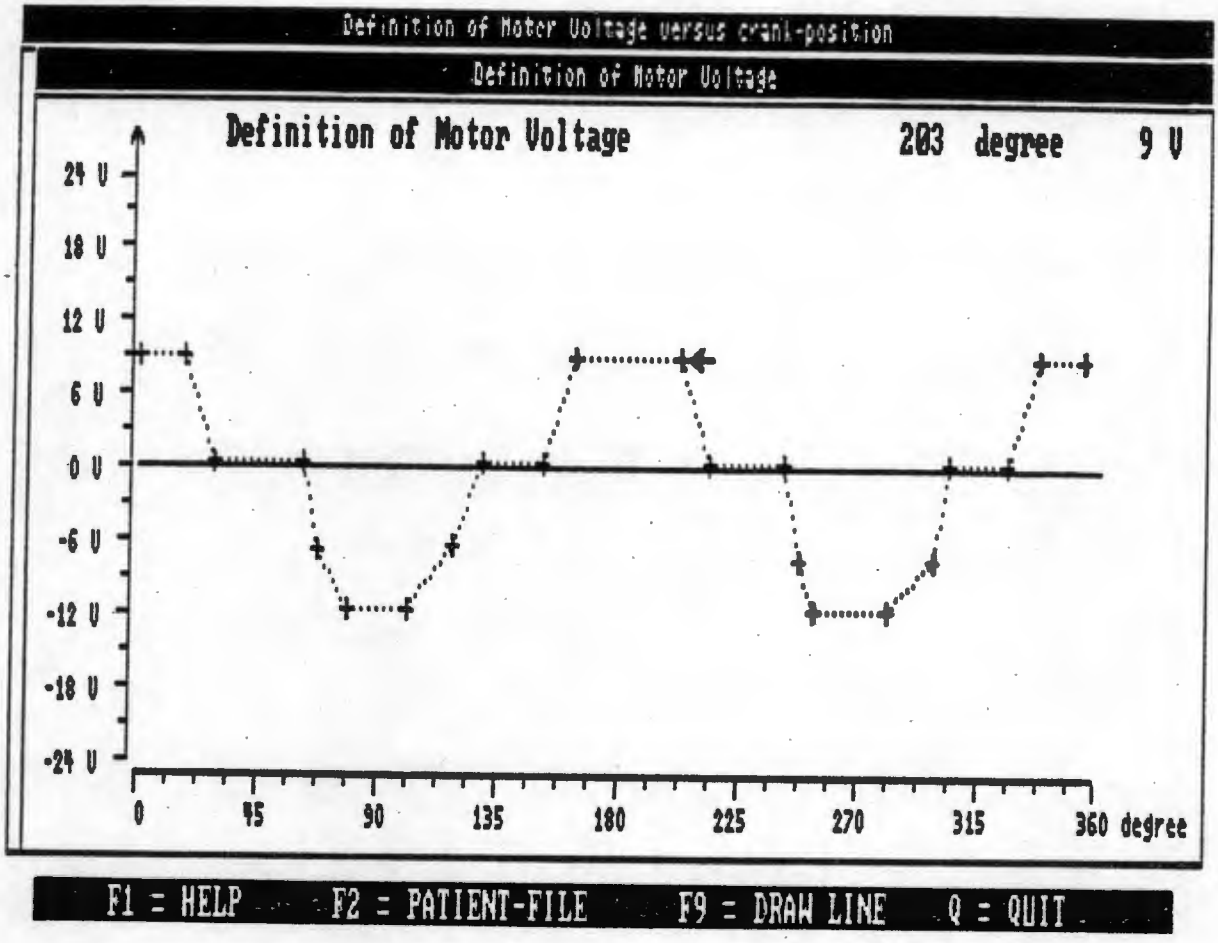


Figure 4.10 Screen display for definition of the motor voltage

position, that is where the maximum torques can be produced due to contractions of the quadriceps muscle group. This assisting and loading during parts of the crank cycle has proven to be most effective in achieving a smooth cycling movement.

DEFINITION OF OTHER STIMULATION PARAMETERS

A menu driven program guides the operator in the definition of stimulation parameters, such as the type of waveform, the pulse amplitude of each channel and the pulse repetition frequency. An online help facility is also provided. Figure 4.11 illustrates the definition of a biphasic pulse waveform. This waveform is

defined by the ratio between the pulse width of the negative and positive pulses, the time interval between the two pulses, and the maximum pulse width of the negative pulse. The amplitude and pulse width of the positive pulse is determined by the program so as to ensure charge balancing.

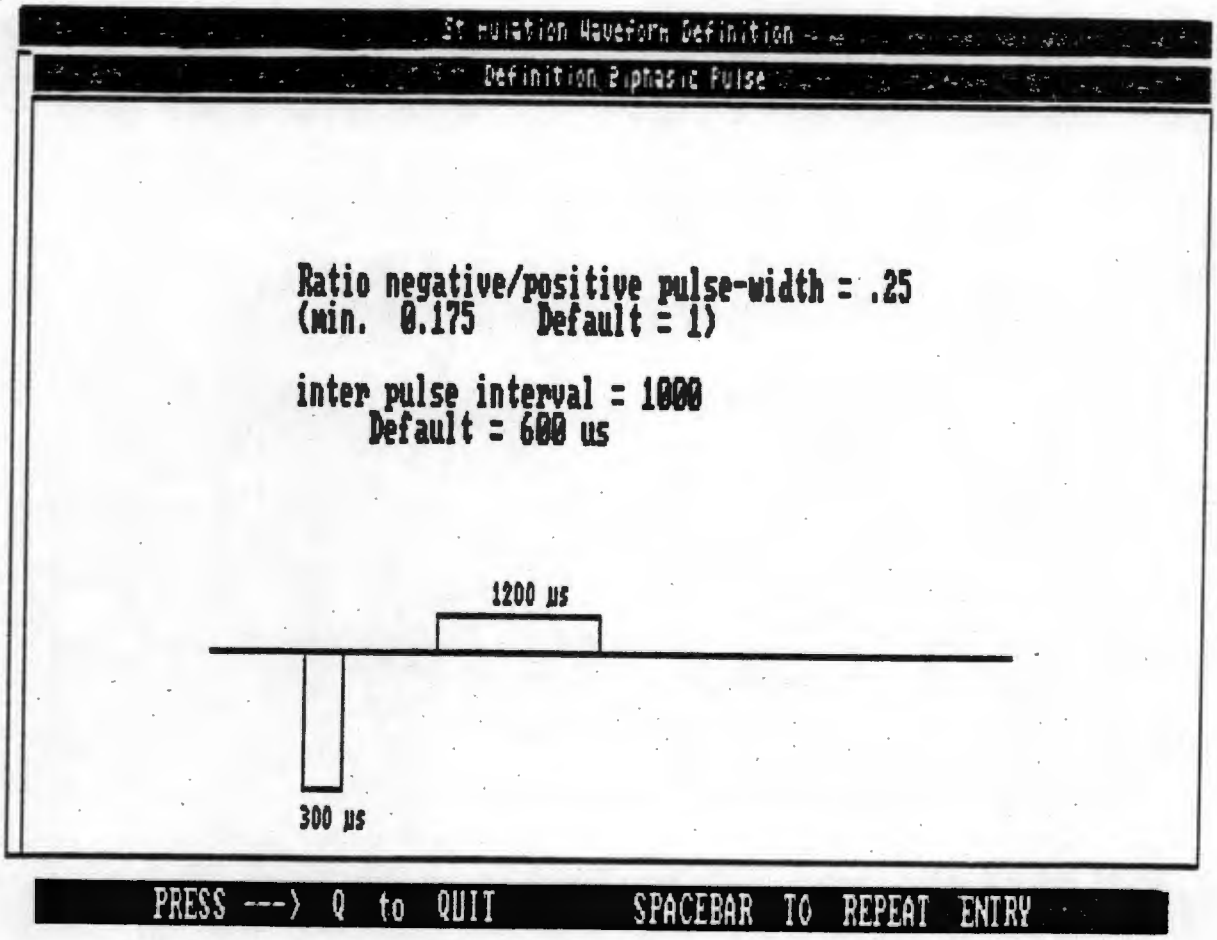


Figure 4.11 Screen for definition of a biphasic FNS waveform

The shape of the waveform is the same in all stimulation channels. However, the pulse amplitudes can be defined independently for each channel (see Figure 4.12). The pulse repetition frequency can be specified in a range of 10 Hz to 99 Hz. The frequency is the same in all channels.

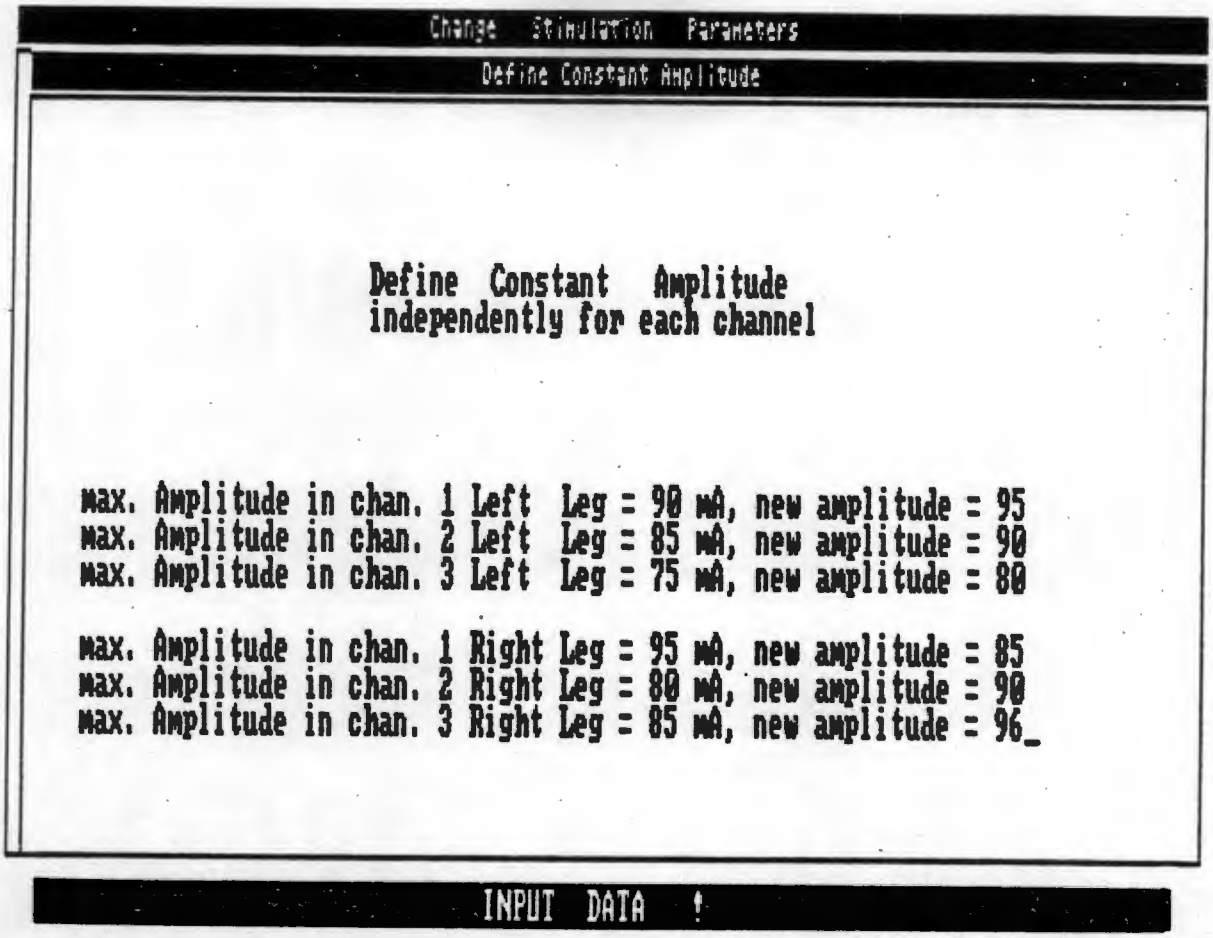


Figure 4.12 Definition of constant amplitudes

CREATION OF THE LOOK UP TABLES

The look up tables used by the real time stimulation process are calculated from the defined stimulation envelopes. The shaft encoder has a resolution of 8-bits and thus the pulse width stimulation envelope of each channel is placed in a one dimensional array of length 256 byte values. Each byte represents the pulse with at a specific crank position. Although pulse amplitudes and frequencies are not defined as a function of the crank position, their values are stored in similar arrays. This makes provision for future expansions of the software which must be compatible with already existing data files created using the present version of software.

DATA STORAGE ON DISK

The stimulation sequences and parameters, as well as personal patient information can be stored on floppy disk. At the beginning of a stimulation session, the operator can specify the drive to be used for data storage (see figure 4.13).

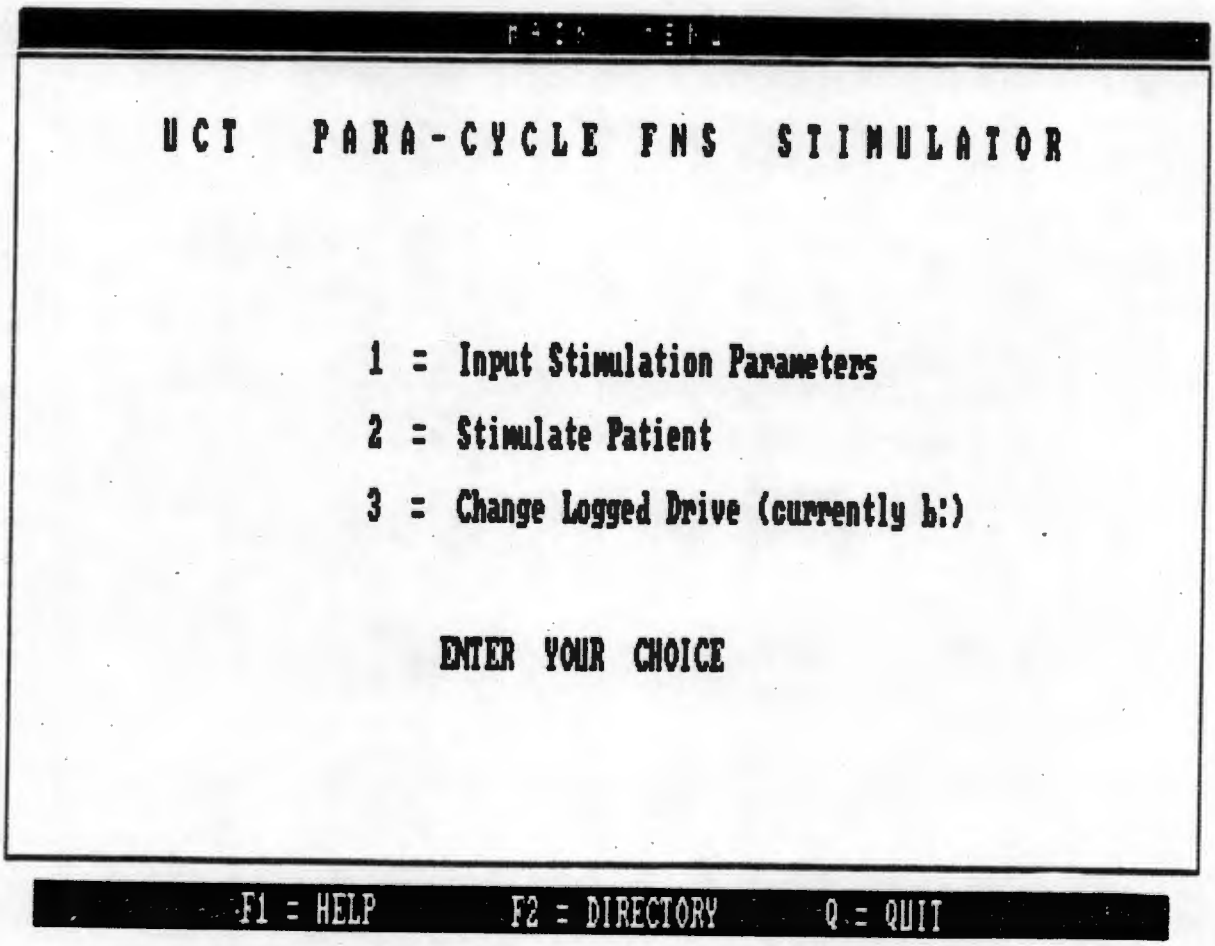


Figure 4.13 Screen display of the main menu

In the 'Input stimulation parameters' mode, the already existing patient files are displayed in the form of a directory (figure 4.14).

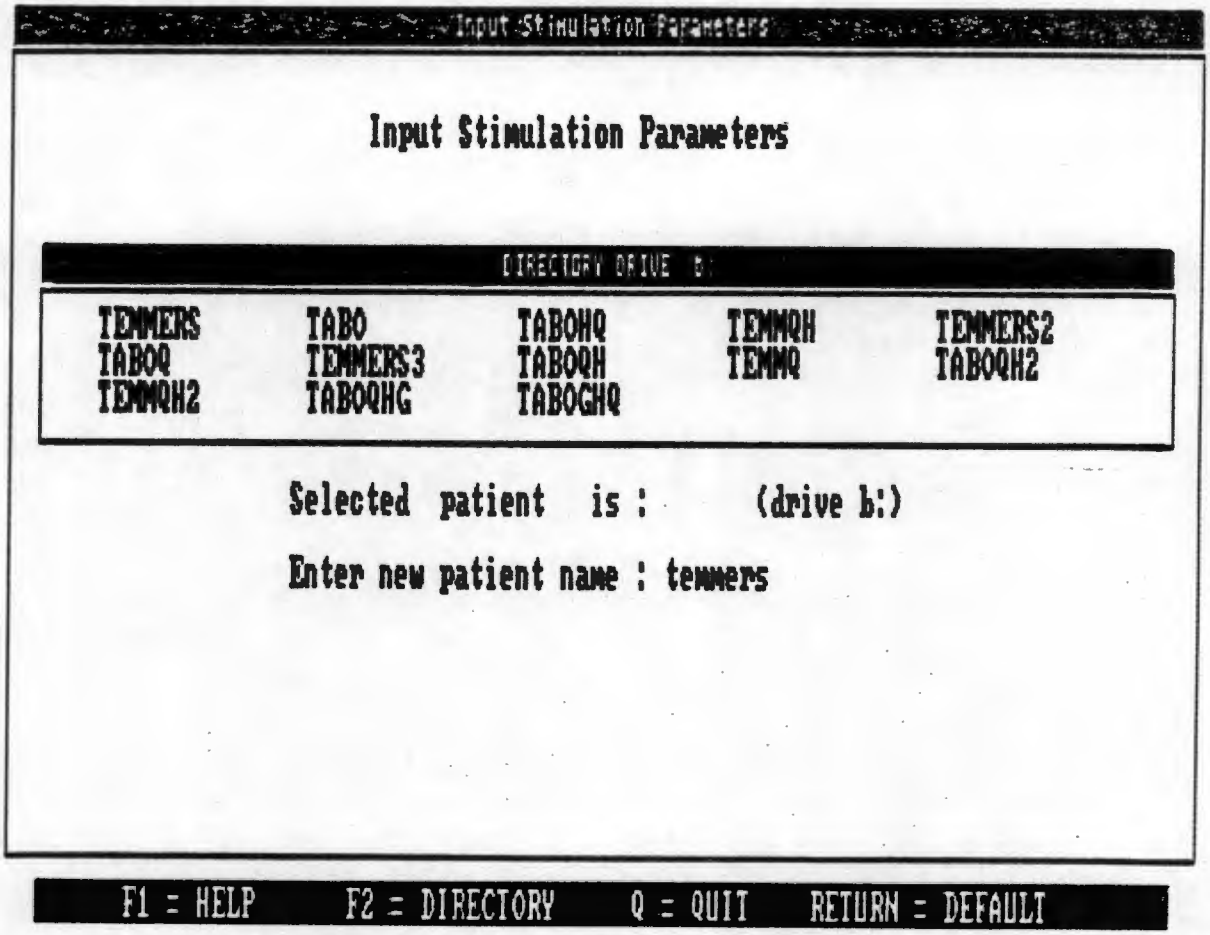


Figure 4.14 Screen display for file selection

The file names consist of a maximum of eight characters (usually the patient's name and an index number). The software automatically adds a three character file extender 'FNS'. Existing files can be copied under a different name into a new file. Thus it is possible to use an already existing patient file for a new patient just by changing the relevant patient data.

Each patient file occupies 17 KByte of disk space. Thus 20 patient files can be stored on a single 360 KByte double sided, double density floppy disk.

Throughout the program, a command line is displayed on the bottom of the screen. The command line informs the operator about his present options within the program (figure 4.15). An on-line help facility can also be accessed if this option given in the commandline (function key F1). The window technique is used throughout the program so that the operator is aware of his level within the program.

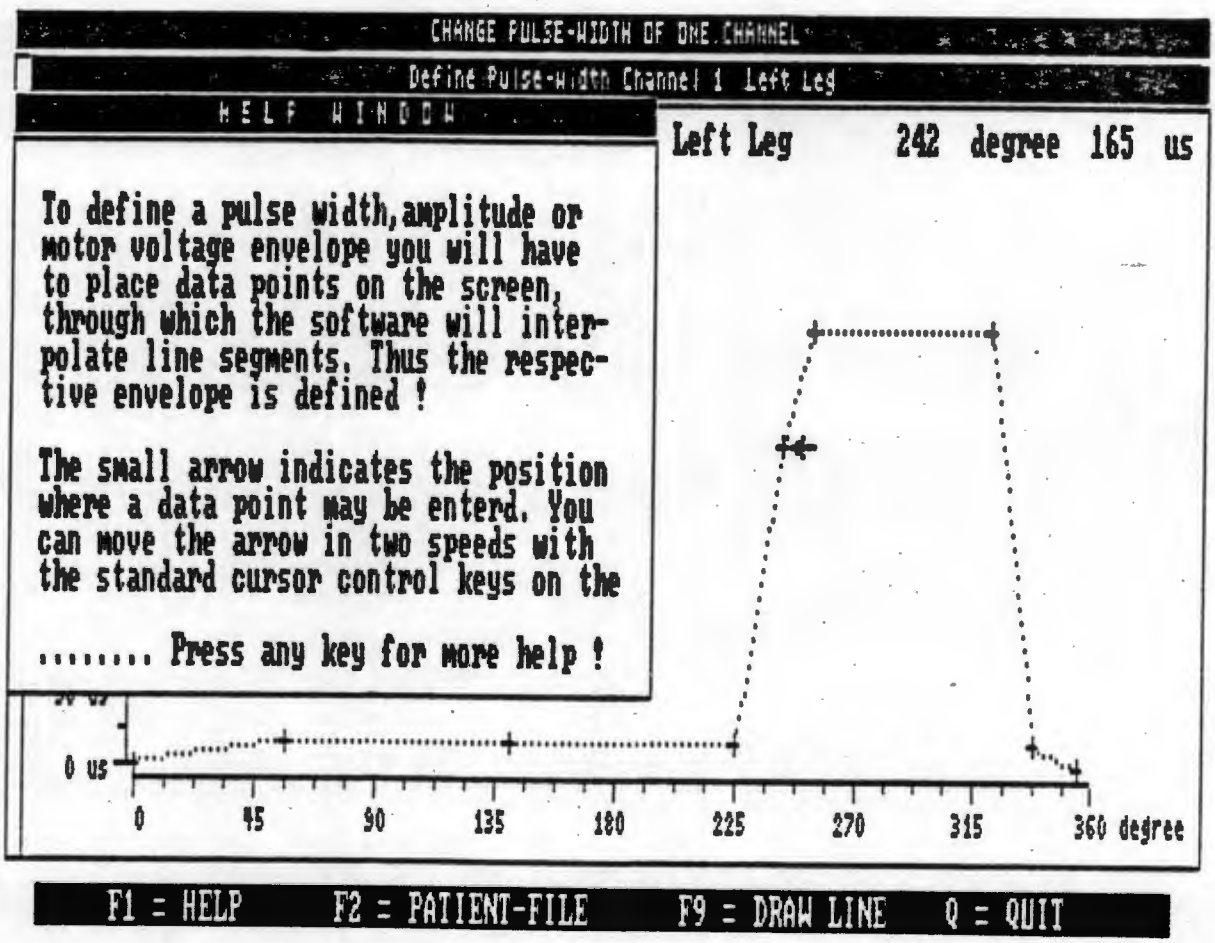


Figure 4.15 Display of a Help text file

4.3.3. THE REAL TIME STIMULATION PROCESS

The real time stimulation process coordinates the six stimulation channels according to the input control variables. The FNS paracycle system is illustrated in Figure 4.1. Table 4.2 defines the various input control signals and their method of measurement.

Table 4.2 INPUT CONTROL SIGNALS AND THEIR SOURCES

Control signal	Method of measurement
crank position	8-bit port (shaft encoder)
crank speed	8-bit port (shaft encoder)
motor voltage	A/D converter
motor current	A/D converter
patient speed control	A/D converter
patient watch-dog switch	single binary I/O port
operator motor control	Keyboard input
operator FNS level control	Keyboard input

Table 4.3 gives a summary of the output signals and variables which have to be controlled by the real time process.

Table 4.3 OUTPUT SIGNALS AND THEIR CONTROL TARGET

Output signal	Target
look up tables for stimulation amplitude, pulse-width, frequency, waveform	36 8-bit ports on the waveform generation board
motor voltage level	8-bit port
motor power (calculated from sensed motor voltage and current)	screen output
crank position	screen output
crank speed	screen output
actual stimulation levels	screen output

The stimulation is a function of three variables which are measured by the real time stimulation process. The first of these variables is the crank position, which serves as an index pointer to the data stored in the look up tables. Secondly, the operator can vary the level of stimulation and/or motor assist (or load) via the keyboard (using four keys of the numerical key pad on the right-hand side of the IBM keyboard). Those parameters can also be controlled by the patient using a 'speed control' lever on the right arm rest of the paracycle. These variables are combined by a method represented in Figure 4.16. Thus the values for the negative and positive pulse width as obtained from the respective look up tables are multiplied by the patient and operator controls. Similarly, the motor voltage data is multiplied by these controls. To improve execution speed, all the multiplications are performed using Byte and Integer arithmetic. The described method of combining the control variables has the advantage that both the patient and the operator have an

influence on the outcome of the stimulation process. In this way, the patient is given control over the function of his otherwise paralyzed extremities and thus plays an active role in the training sessions.

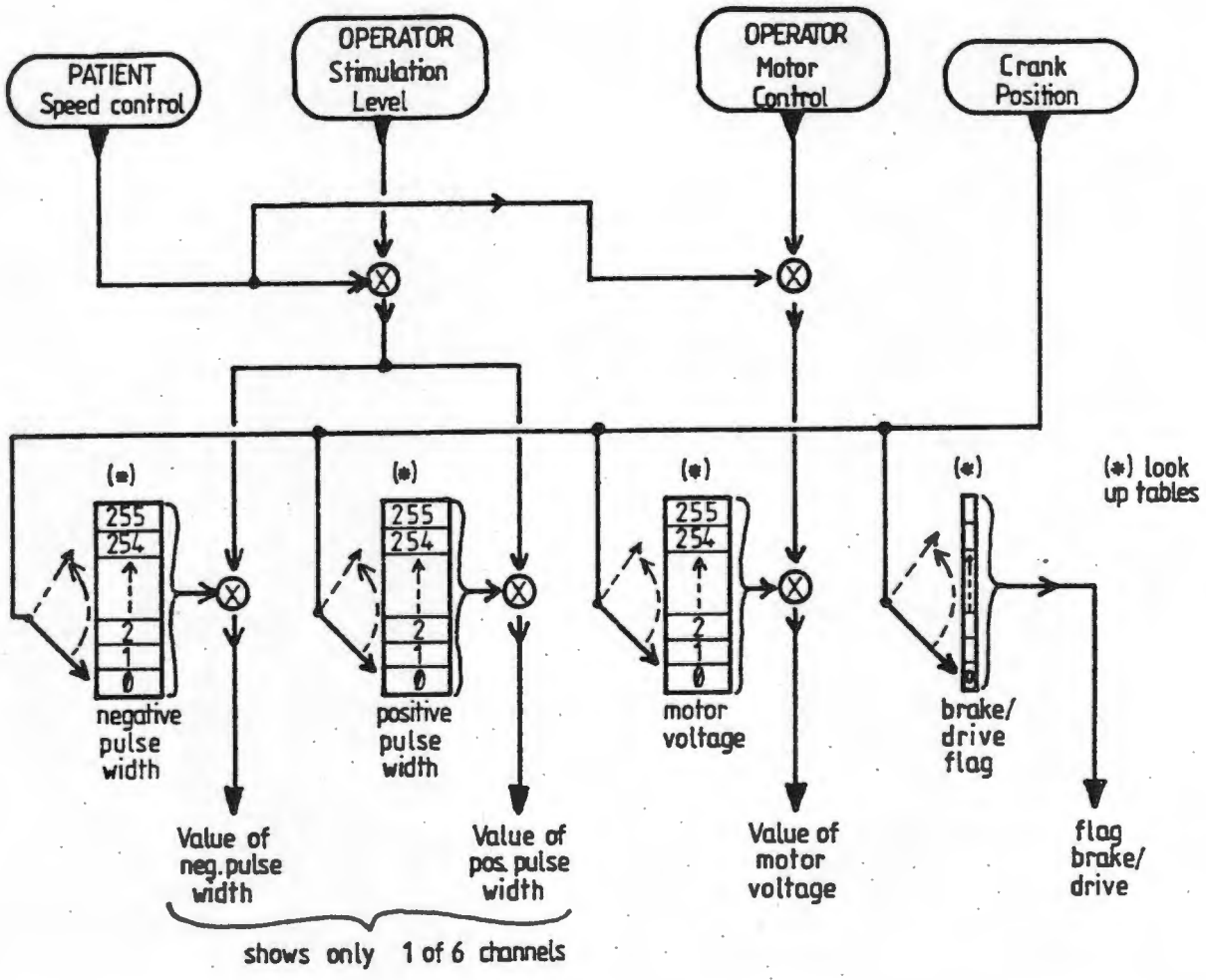


Figure 4.16 Combination of control variables

The flow chart of the real time stimulation process is illustrated in Figure 4.17 , while the screen layout during real time stimulation is illustrated in figure 4.18.

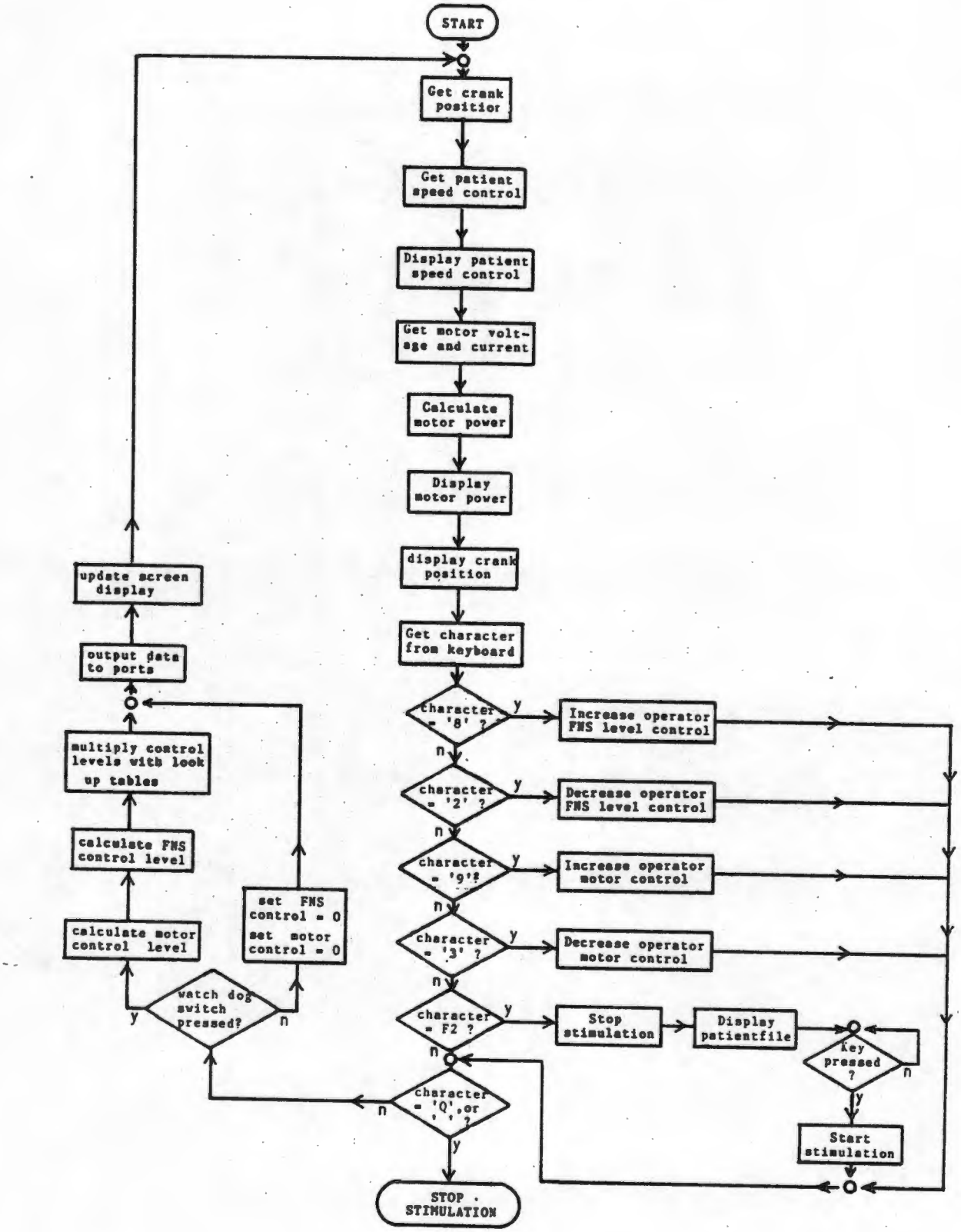


Figure 4.17 Flow chart of the real time stimulation process

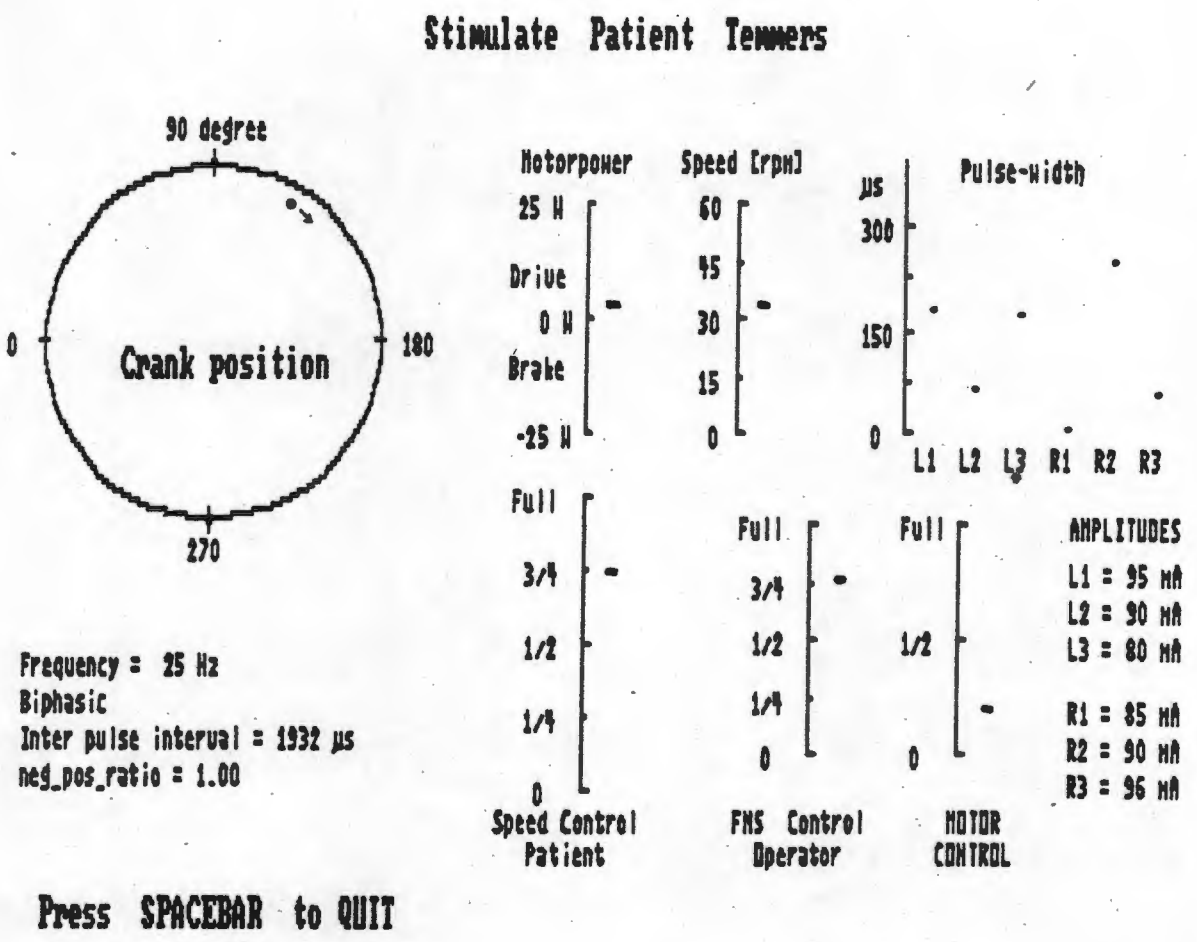


Figure 4.18 Screen display during real time stimulation

The levels of the operator and patient controls are displayed in the form of simplified bar graphs. This type of display has a considerable speed advantage over numerical print commands or more advanced graphics displays. The crank position is displayed in real time as a moving dot within the crank circle. The actual pulse width of the negative-going pulses of each channel are also displayed in real time. The motor power used to drive or brake the patient is also displayed using a modified bar graph. The crank speed is measured every revolution and then updated on the screen.

On the SPERRY personal computer, the main loop of the real time stimulation process takes approximately 11.6 ms to execute. If a keyboard command from the operator has to be processed, this execution time is increased by another 2 ms in the worst case. Without the real time displays of crank position, pulse width and motor power, the execution time can be decreased to about 5 ms (ie. the stimulation parameters can be updated every 5 ms). The results of FNS training sessions showed, that for crank speeds up to 60 revolutions per minute (rpm), there was no visual difference in the patients cycling performance irrespective whether the stimulation parameters were updated every 5 ms or every 11.6 ms. One reason for this lies in the fact that a pulse repetition frequency of 25 Hz, ie. a pulse every 40 ms was used. Thus the pulse width values are updated at a rate four or eight times higher than the actual pulse frequency. To determine the worst update rate it can be assumed that the pulse repetition frequency is fixed to 25 Hz and a pulse with a certain pulse width has just occured. It can be assumed further, that the crank moves on to a position where the new pulse width found in the respective look up table is much different from the last one, already occurred. This new pulse width value is sent to the pulse generating hardware but the pulse with the new pulse width is produced earliest 40 ms. However, as long as the controlling program loop (which updates the stimulation parameters) executes faster than the pulse repetition frequency, it is essentially the pulse repetition frequency which determines the update rate of the stimulation process. This is the case with the described program for pulse repetition frequencies up to 80 Hz (the loop executes at a rate of 1/11.6 ms = 86 Hz). Thus, with a loop execution time of 11.6 ms, every single pulse occurring with

repetition frequencies up to 80 Hz can be controlled without losing track. This is more than adequate for the present application of the program.

4.3.4. SUMMARY

The described software package for the FNS controller enables the user to define individual stimulation parameters and sequences for each subject. The stimulation patterns and patient data can be stored on floppy disk. The real time stimulation process measures the crank position of the paracycle and controls the stimulation accordingly. The entire software package, including all real time stimulation procedures, is written in TURBO PASCAL. Thus future changes dealing with the control of all stimulation parameters and the implementation of closed loop algorithms is made relatively easy . Despite the use of a high level language, the existing real time software updates the stimulation hardware and the screen displays every 11.6 ms. Without screen displays the update interval can be reduced to about 5 ms. Commented program listings are given in the appendix H.

SECTION III : SYSTEM EVALUATION

Chapter 5. FNS PATIENT TRIALS - METHODS AND RESULTS

5.1. SUBJECT SELECTION

After both the hardware and the software of the FNS controller have been tested extensively with regard to reliability and safety the system was applied in a clinical environment in trials involving two paraplegic subjects. The purpose of these tests was to evaluate the performance of the FNS controller in conjunction with the Paracycle. The following guidelines were defined for the selection of the patients (Kralj et al., 1983) :

a.) Complete paraplegia - low level, upper motor neuron lesion of recent origin ;

b.) No heavy traumatic involvement;

c.) No lower motor neuron involvement ;

d.) Acceptable level of spasticity;

e.) No physical contra-indications (eg. pressure sores, heterotrophic ossification)

f.) Good understanding of the purpose of the project and informed consent;

The reason for choosing a patient with a lesion of recent origin was to allow the system to be tested with subjects who showed a reasonable response to FNS. The two subjects finally selected by the medical staff of the local spinal rehabilitation centre (Conradie Hospital) are described below :

- Subject A : 16 year old male; complete lesion at T3; knife wound; two month post injury; medium spasticity level;
- Subject B : 19 year old male; complete lesion at T6; knife wound; two month post injury; no spasticity;

The patient trials were conducted under medical supervision.

5.2. ELECTRODES AND ELECTRODE PLACEMENT

Electrodes play a major role in determining the effectiveness of electrical stimulation and the ease with which it can be applied. From the variety of electrodes commercially available for electrical stimulation, the carbon rubber type was chosen. These electrodes represent a current standard used by many FNS research groups (Benton et al., 1981; McMiken et al., 1983; Petrofsky et al., 1984c). Two sizes of silicon carbon rubber type electrodes were used (Medtronic Inc. Order # 3795 size 50 x 40 mm and # 3793 50 x 100 mm). The electrode impedance was found to vary between 300 and 600 Ohms, depending on the size of the electrodes. This low resistance was achieved using only a thin layer of gel between the skin and the electrodes. The electrode resistance is determined by the FNS controller by measuring the output voltage of the stimulator during a pulse of width 90 μs and a known current amplitude.

Electrical stimulation via surface electrodes is most efficient when the electrodes are placed over the motor points of the muscle (where the motor nerve enters the muscle) or over the endplate zone (where there is concentration of nerve endings) (Bowman and Baker, 1985). The position of the motor points differs between patients whereas the end-plate zone generally lies near the central portion of the muscle fibres (Benton et al.,1981).

Stimulation, using a bipolar electrode configuration and biphasic waveforms, was shown to be approximately 25 % more effective than using a monopolar configuration with monophasic waveforms (McNeal and Baker, 1985). For bipolar stimulation of a muscle group, one needs to define two electrode locations per

muscle. A monopolar system (ie. one active and one indifferent electrode) is used to probe for these points. The method for determination of the electrode locations for the quadriceps and hamstrings muscle groups of the subjects was were adapted from that used by McNeal and Baker (1985), and Bowman and Baker (1985).

THE DETERMINATION OF BIPOLAR ELECTRODE LOCATIONS

For determination of the two electrode locations of the quadriceps muscle group the patient is lying supine with 60-90 degrees of knee flexion. A force transducer is attached to the ankle to measure the isometric force generated by the quadriceps response to stimulation. For determination of torque the distance between the knee joint and the force transducer attachment is measured. A monopolar electrode configuration is used with the indifferent electrode (50 x 100 mm) placed immediately superior to the patella. A smaller active electrode (40 x 50 mm) is used to decrease the 'target area' of the stimulation (Benton et al.,1981). Figure 5.1 illustrates the grid used to define the location of the active electrode on the patient's thigh. The grid layout is adapted from McNeal and Baker (1985) in order to allow comparison of results. The stimulation waveform is biphasic with a ratio between negative and positive pulse width of 0.2 . The negative pulse width is maximum 300 μs, interpulse interval is 1000 μs and repetition frequency is 30 Hz.

For each grid cell a measurement of generated force versus stimulation pulse width is measured. The pulse width is increased from 0 to 300 μs within 5 seconds, while keeping the pulse amplitude constant at 90 mA. The maximum generated force is recorded to give an indication of the effectiveness of the

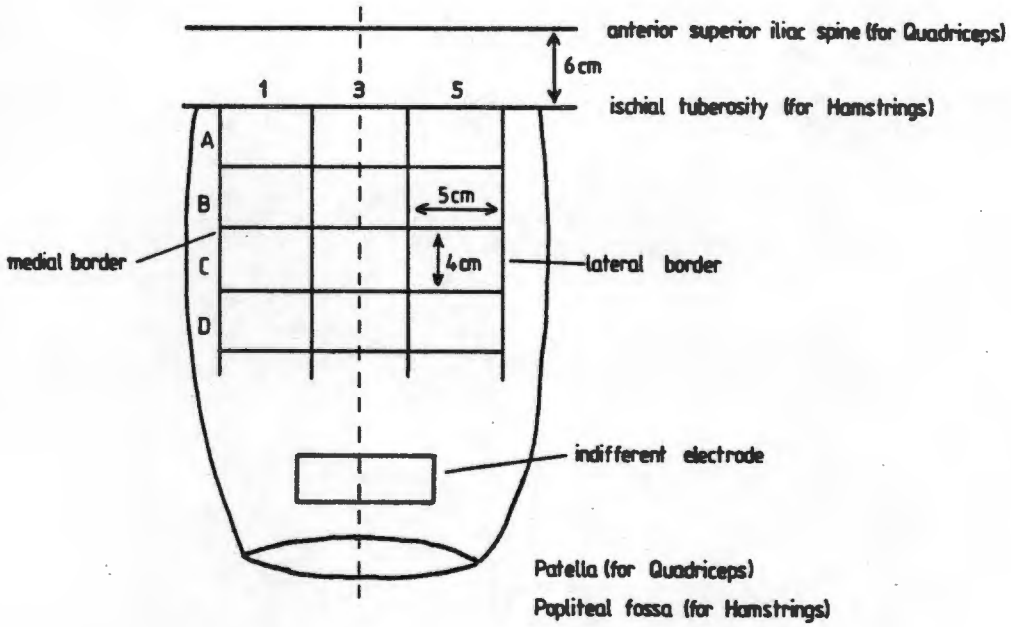


Figure 5.1 Grid used for definition of Quadriceps electrode location (adapted from McNeal and Baker, 1985)

respective electrode location. The measurement is controlled by a special software package in conjunction with the FNS-controller hardware. For safety, the software automatically stops stimulation if the generated force exceeds 100 Newton. The test was performed on both legs.

For determination of the bipolar hamstring electrode positions the patient was in a prone position with the knee resting in approximately 30 degree of flexion. The indifferent electrode (50 x 100 mm) is placed distal to the active just superior to the popliteal fossa. The active electrode (40 x 50 mm) is positioned according to a grid illustrated in Figure 5.1 . The resulting isometric forces are measured with a force transducer attached to the ankle. These measurements are performed as described above.

The results of the procedure are given in Table 5.1 . These results vary considerably between the two subjects and do not coincide with the results of McNeal and Baker (1985). However, this case study allows no final conclusions in this respect.

TABLE 5.1 BIPOLAR ELECTRODE LOCATIONS FOR SUBJECT A and B

	SUBJECT A	SUBJECT B
left Quadriceps	E 3 - I 3	G 4 - J 4
right Quadriceps	E 3 - I 3	G 3 - K 3
left Hamstrings	E 3 - C 5	G 3 - K 4
right Hamstrings	E 3 - C 3	G 3 - K 4

5.3. STIMULATION TRIALS ON THE PARACYCLE

As a first approximation, the stimulation sequences used to generate a cycling movement in paralyzed subjects were based on the electromyographic (EMG) activity recorded from non paralyzed subjects using the Paracycle. Thus four nonparalyzed subjects were asked to pedal on the Paracycle at 30 and 60 revolutions per minute (rpm) against loads of 100 Watts and 200 Watts. The EMG was recorded in five muscle groups simultaneously. The raw EMG was fullwave rectified and smoothed with a low-pass filter with 50 Hz cut off frequency. Each channel was sampled at a sampling rate of 200 Hz and the data averaged over 10 revolutions. The results of the Quadriceps and Hamstrings muscle groups averaged over five subjects are illustrated in Figure 5.2 and Figure 5.3 respectively. For comparison the results of similar EMG studies and stimulation strategies on bicycles are also given. It can be

seen that there are considerable differences in the recorded EMG patterns during cycling which may be due to the varying mechanical arrangements of the seating and subject posture.

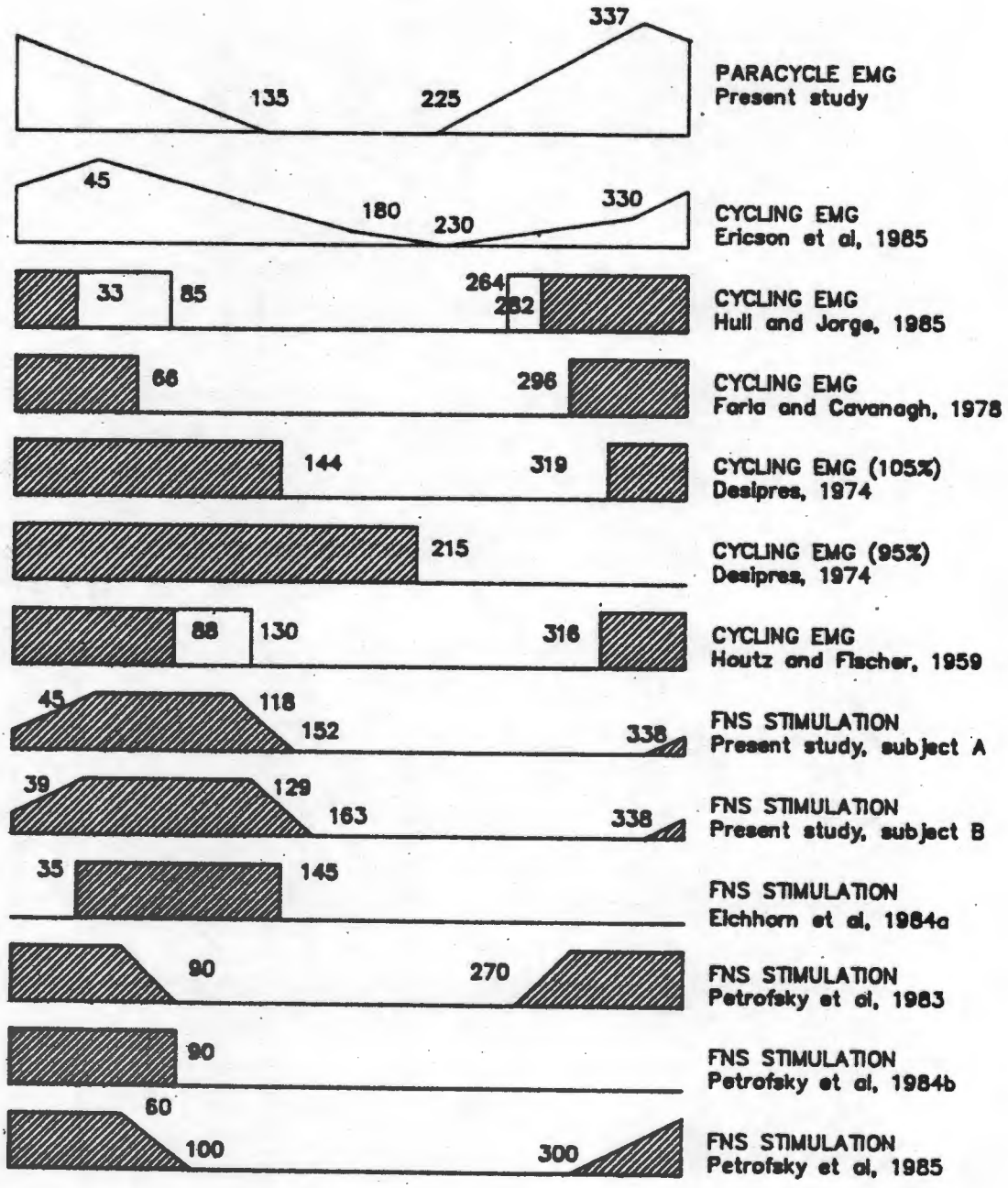


Figure 5.2 EMG activities and stimulation strategies of the Quadriceps muscle group

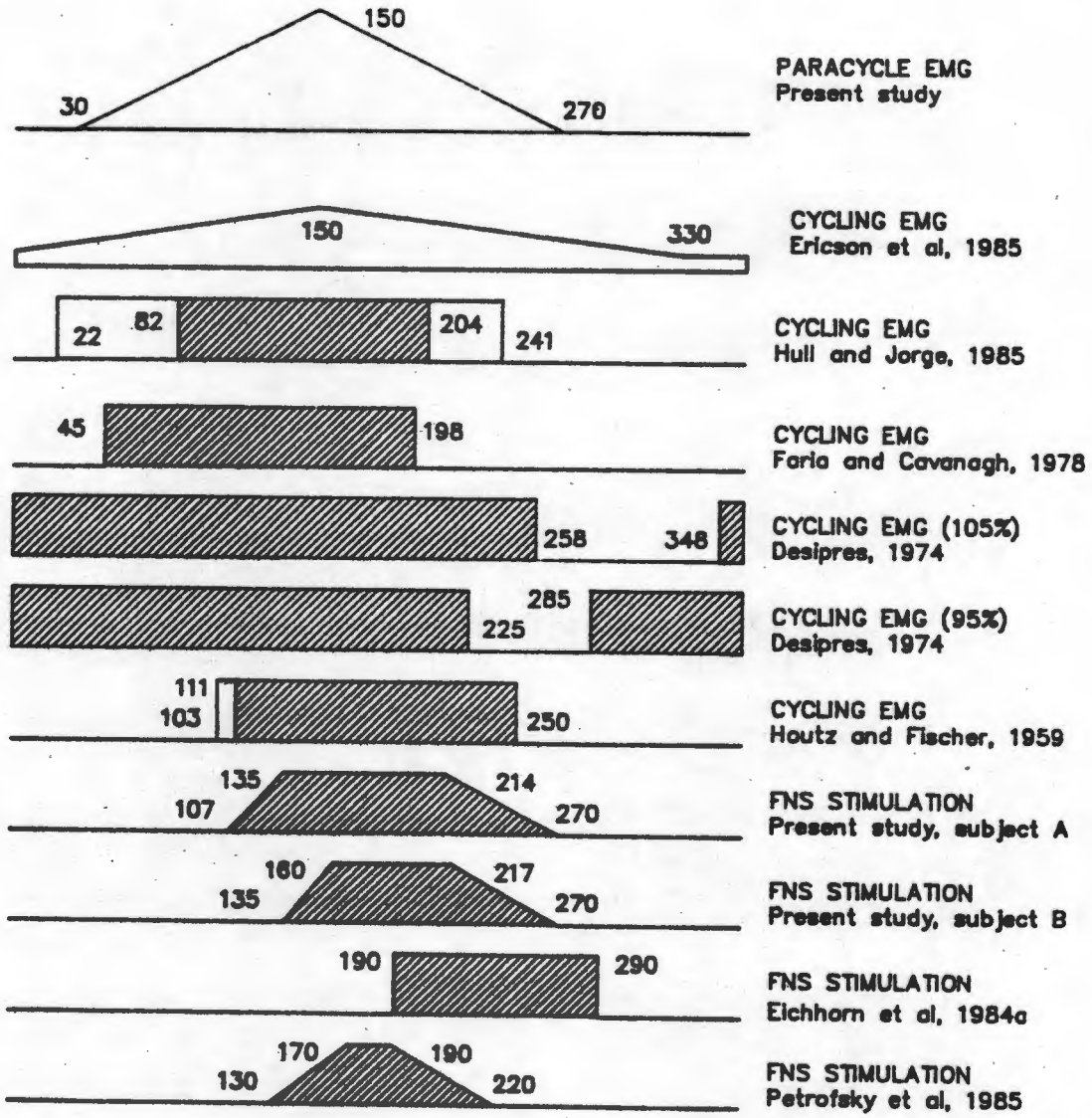


Figure 5.3 EMG activities and stimulation strategies of the Hamstrings muscle group

Based on these recordings of EMG activity during cycling, the stimulation sequences for the quadriceps and hamstrings muscle groups were programmed into the FNS-controller. The motor was programmed to assist the subject in the 'dead spots' of the crank cycle (ie. at 0 and 180 degrees). The resulting stimulation pattern is also illustrated in Figures 5.2 and 5.3 . However,

stimulation of a paralyzed subject using these sequences did not yield an useful movement. Particularly interesting was the fact that the activation of the quadriceps group at crank positions between 270 and 360 degrees resulted in a crank movement in the reverse direction. This could be explained by the mechanical arrangement of the Paracycle and the known function of the quadriceps as a knee extensor.

The reason for the unsuitability of these EMG-based sequences for stimulation purposes may be that during the EMG study most of the leg muscles are active at some stage during a cycling movement. In particular this includes antagonistic muscle activity which does not contribute to the actual movement, but results in joint stability and fine movement coordination. However, when cycling under stimulation only two muscle groups are being utilized.

It was thus decided to define first approximation stimulation sequences based on the mechanical arrangement of the Paracycle and the main functions of the quadriceps and hamstrings muscle groups. These sequences could then be improved during stimulation sessions on a trial and error basis. The final stimulation pattern for subject B is shown in Figure 5.4 .

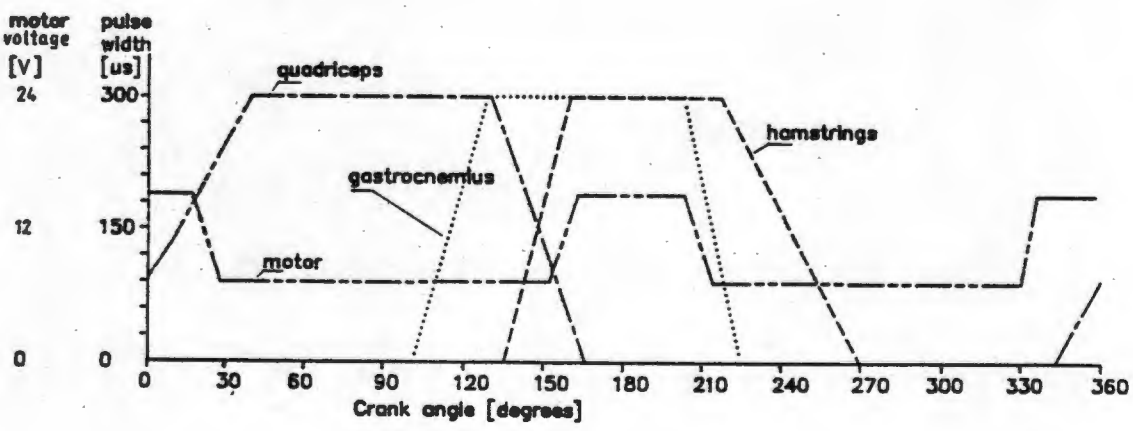


Figure 5.4 Final Stimulation pattern subject B

The amplitudes of the different channels have been adjusted to accommodate for strength differences between the muscle groups. During the early stages of the training program the motor assists the subject during small parts of the crank cycle at crank positions around 0 and 180 degrees. During the rest of the crank cycle, the subject actively turns the crank, the gearbox and the motor. The load imposed by the friction and mechanical losses depends on crank speed. At 30 round per minute (rpm) the load was determined experimentally to be 8 Watts and at 60 rpm approximately 18 Watts. Thus the subject is performing work during parts of the crank cycle. After six weeks FNS training (see following paragraph) the subject was able to pedal without motor assist. The stimulation pattern of the Gastrocnemius muscle as shown in Figure 5.4, was used during the final sessions of the training program. The use of Gastrocnemius resulted in a smoother cycling movement than when using Quadriceps and Hamstrings alone.

Both subjects took part in training sessions on the Paracycle three times a week, for a period of six weeks. Each session consisted of six training periods during which the subject performed FNS-induced cycling. The duration of the training periods was initially two minutes and was increased in steps of half a minute, depending on the subjects progress. The motor assistance or load during stimulation was adjusted by the operator to ensure that the subject did not fatigue. Each session was started with a start period (2 minutes) of passive motion at 30 rpm. The aim of passive motion is to decrease spasticity and stiffness and to increase joint mobility and circulation. The subject was also driven passively by the motor between each training period.

After 6 weeks of training, subject A was able to cycle for 4 minutes against a load of approximately 30 Watts. Subject B was able to cycle for 4 minutes against a load of approximately 10 Watts without motor assistance. It is important to note that with the present system it is difficult to quantify the work performed by the subject if the motor is used to assist parts of the cycle. However, the fact that both subjects could cycle without motor assistance against a greater load for a longer period than at the start of the training program, allows one to conclude that the endurance of both subjects had improved.

5.4. TORQUE VERSUS PULSE-WIDTH RELATIONSHIP

The influence of the stimulating pulse width on the level of muscular contraction was investigated using the methods already described for determination of electrode locations. However, a

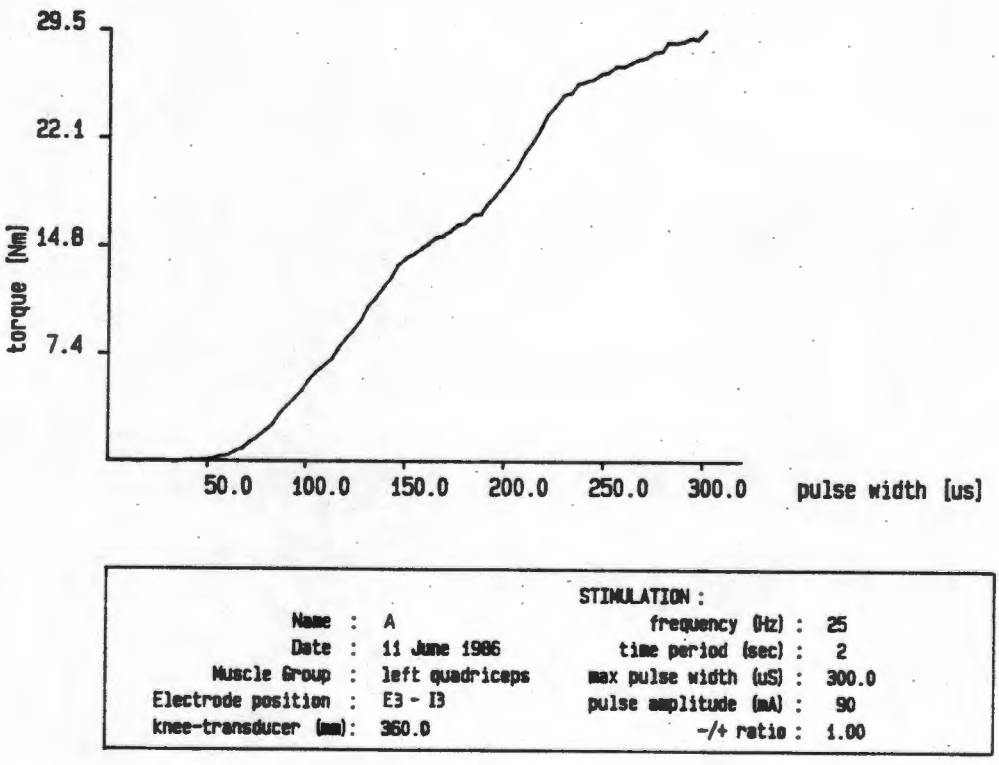


Figure 5.5 Torque versus Pulse width relationship subject A

biphasic waveform and bipolar electrode configuration was used. The results for subject A is illustrated in Figure 5.5. It can be seen that there exists an minimum pulse width which is necessary to create a functional useful knee torque. This coincides with the results published by Benton et al. (1981) and Edwards et al. (1977).

5.5. TORQUE VERSUS PULSE AMPLITUDE RELATIONSHIP

The relationship between the pulse amplitude and the resulting knee torque was investigated with the method described in chapter 5.3. with the difference that the pulse width was constant 300 us and the pulse amplitude was increased from 0 to 90 mA. Figure 5.6 illustrates the result of this test.

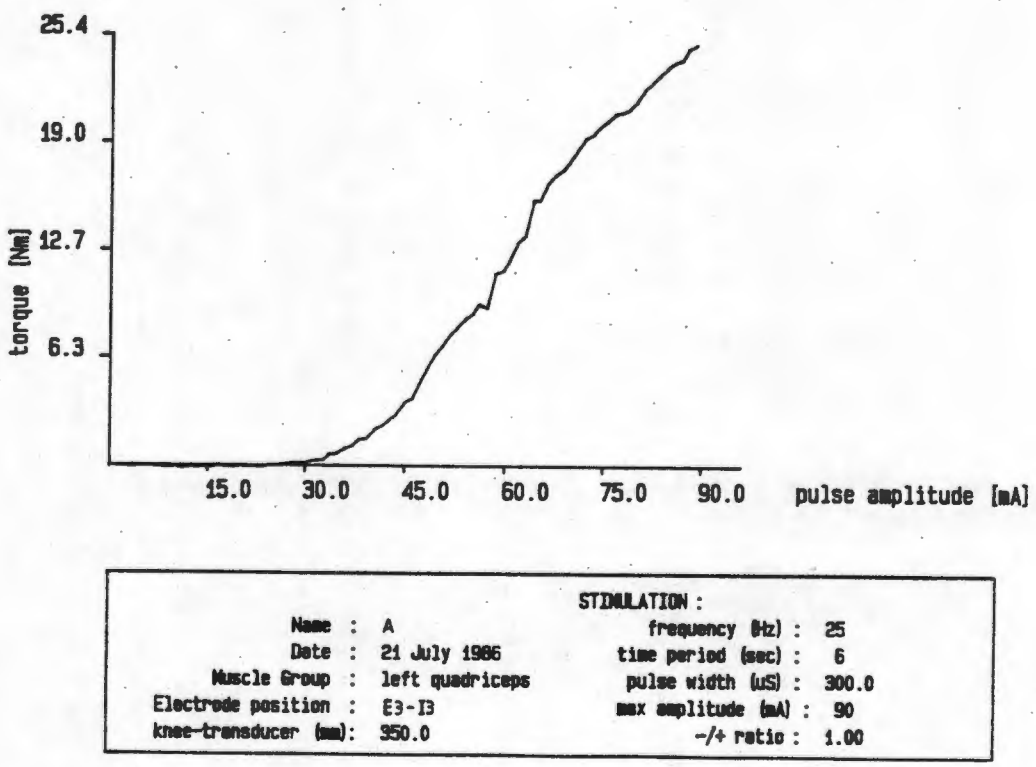


Figure 5.6 Torque versus Pulse amplitude relationship subject A

Similar to the torque versus pulse width relationship a minimum amplitude exists which is necessary to produce a detectable knee torque. However, this minimum amplitude amounts to about 30 % of the maximum amplitude used in this test, whereas the minimum pulse width amounts only to about 20 % of the maximum pulse width used. This more linear characteristic of the torque/pulse width relationship may support the decision to control the muscular contraction level by changing the pulse width rather than the pulse amplitude. However, a test with a single subject allows no final conclusions.

5.6. TORQUE VERSUS FREQUENCY RELATIONSHIP

The torque generated by the quadriceps muscle group was measured as a function of the pulse repetition frequency (see Figure 5.7).

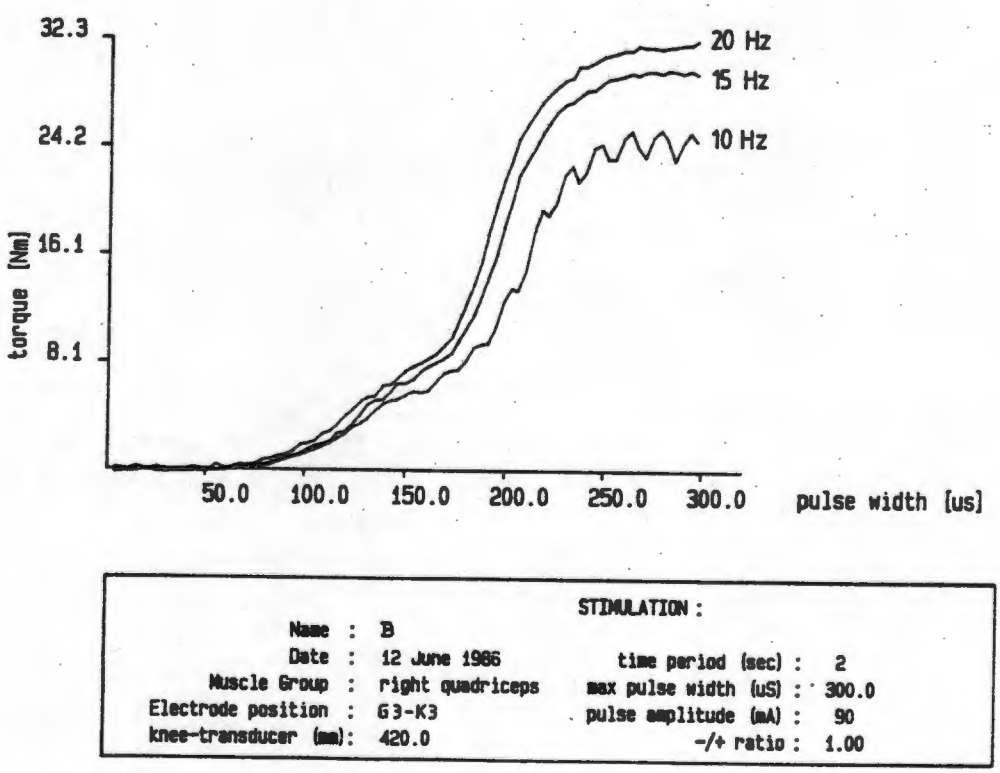


Figure 5.7. Torque versus pulse frequency

A biphasic symmetrical waveform with a positive and negative pulse width of 300 µs, and constant pulse amplitude of 90 mA was used. These results are similar to those presented by Edwards et al. (1977), except that the torques produced at a repetition frequency of 10 Hz were greater than expected. Based on these findings, a pulse repetition frequency of 25 Hz was used during the Paracycle training program to minimize the fatigue effects.

5.7. FATIGUE AS FUNCTION OF PULSE FREQUENCY

The influence of pulse frequency on fatigue was investigated with a prolonged supramaximal stimulus (biphasic waveform, pulse width 200 us, amplitude 90 mA, frequency 25 Hz and 100 Hz). Between the two measurements, the muscle was rested for 10 minutes. The result is illustrated in Figure 5.8 .

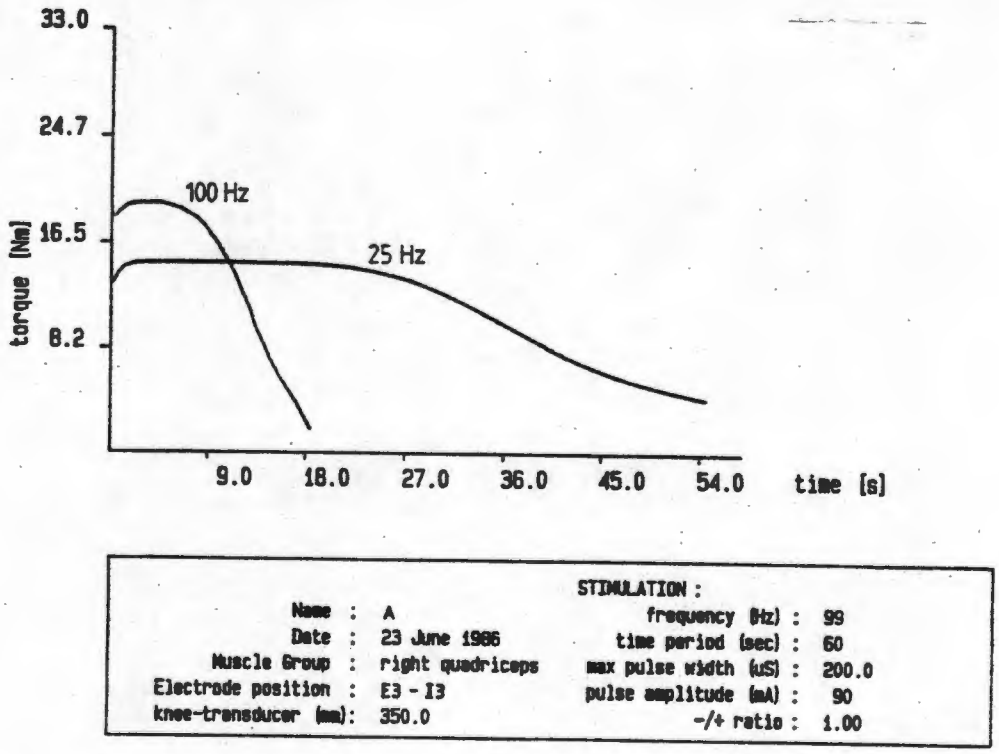


Figure 5.8 Fatigue as function of pulse frequency.

The generated torque at the beginning of stimulation with 100 Hz pulse frequency is greater than with 25 Hz. At 100 Hz, the torque rapidely decreases after 7 seconds of stimulation whereas at 25 Hz the torque is maintained for as long as 23 seconds. However, even at 25 Hz, fatigue is evident much earlier than in non-paralyzed subjects investigated by Edwards et al. (1977). This may be due to the inefficiency of electrical stimulation described in chapter 2.5. of this thesis.

5.8. SUMMARY OF RESULTS OF THE PATIENT TRIALS

The aim of the described patient case studies was to evaluate the performance of the FNS-controller in a clinical research environment. After selection of suitable subjects, electrode locations and the optimal stimulation sequences were determined for each individual. The two subjects participated in a six week training program on the Paracycle. The response of the subjects to FNS was monitored weekly using standard force measurements. Although both subjects had a similar lesion level and case history, their response to FNS was quite different. Subject A was able to produce torques approximately 25 % greater than subject B. This may be due to the higher level of spasticity observed in subject A (resulting in a better preserved muscle bulk). It was observed that the level of spasticity decreased after about 2 minutes of passive motion on the Paracycle. Therefore each training session was started with a short period of passive movement. The endurance of the subjects had increased notably at the end of the training program. Subject A was able to drive the Paracycle from a stationary position for about 10 meters without any motor assistance, the distance covered being limited mainly by the length of the cables connecting the Paracycle to the personal computer.

SECTION IV : CONCLUSIONS AND RECOMMENDATIONS

Chapter 6. SUMMARY AND DISCUSSION OF THE FNS-CONTROL SYSTEM AND ITS PERFORMANCE

6.1. SUMMARY

6.1.1. GENERAL DESIGN PROCEDURE OF THE FNS-CONTROLLER

A Functional Neuromuscular Stimulation control system (FNS-controller) based on a standard IBM compatible Personal Computer has been described. The FNS-controller is designed for use with a stationary cycling device (Paracycle), used for the training of the lower extremeties of paraplegic subjects.

One of the aims in the design of the FNS-controller was to develop a system with which the results of similar studies elsewhere could be confirmed and improved on. It was therefore decided to design a system which is as flexible as possible, in order to cope with all the variations concerning stimulation parameters used by other researchers. After extensive study of the relevant literature, specifications for stimulation waveforms and characteristics of FNS using surface electrodes were drawn up. Furthermore, it was noted that the determination of stimulation sequences used to achieve a certain functional movement with FNS is mostly a trial and error procedure.

These considerations led to the development of an FNS-controller with which it is possible to set stimulation parameters, such as pulse waveform, amplitude, width and frequency, independently for each channel and as a function of certain system input variables. A software package written in TURBO PASCAL controls the procedure to define the stimulation parameters and sequences, which are entered as a function of crank position. The entered data sets can be edited on the graphics screen between stimulation sessions and be stored on floppy disk for later reference.

The stimulation waveforms are generated by digitally programmable

modules, which operate independently from the Personal Computer's processor. These modules allow the computer simply to coordinate the stimulation process in a high level language rather than to generate the actual stimulation waveforms.

The variables controlling stimulation are the position of the paracycle's crank and the positions of controls operated by the subject and the system operator.

To accomodate paralyzed subjects with a high degree of muscle atrophy it was decided to provide a motor to assist the cycling motion in defined parts of the crank cycle. The same motor can be used to impose a programmable load on the subject.

6.1.2. TESTING OF THE FNS-CONTROLLER

The FNS-controller hardware and software was extensively tested for reliability and safety in a laboratory environment. The user-friendliness of the software was evaluated informally by members of the research team.

The FNS-controller/Paracycle system was tested in a clinical case study with two paraplegic subjects (complete T3 and T6 lesions of recent origin). The subjects were selected according to their response to FNS and criteria described previously. Suitable electrode locations and torque versus stimulation parameter characteristics were determined in each subject for the quadriceps and hamstrings muscle groups. The Quadriceps muscle group of both subjects were able to produce knee joint torques greater than 30 Nm. Knee joint torques produced by the Hamstring muscle group were approximately 8 Nm.

The stimulation sequences used to produce a cycling movement were

based on the mechanical characteristics of the paracycle and the known biomechanical actions of the muscle groups in question. The stimulation sequences and individual stimulation parameters were stored in the FNS-controller. Initially the subject was assisted through the 'dead spots' of the crank cycle. During the first sessions both subjects were able to cycle with partial motor assist for 2 minutes at 30 rpm at an average load of approximately 5 watts. During the subsequent training program the stimulation sequences were adapted to accomodate the subjects' improved response to FNS. After six weeks, both subjects were able to cycle for 5 minutes at 30 rpm without motor assistance against a load of 30 watts and 10 watts respectively.

6.2. ADVANTAGES OF THE DESIGN

The FNS-controller offers easy programmability of stimulation parameters. The combination of a standard IBM compatible Personal Computer with customized expansion boards and a software package written in TURBO PASCAL will prove to be highly advantageous for future expansions of the system. This is mostly due to the fact that the waveform generation is performed by dedicated output modules which gives the Personal Computer freedom to acquire and process data for the real time control of the stimulation. The speed of TURBO PASCAL allows even the real time control routines to be written in this high level language. This will enable more complex control algorithms to be implemented with ease.

The possibility to assist the paralyzed subject in the early stages of the training program on the paracycle to overcome 'week spots' of the crank cycle, and thus achieve a smooth cycling movement, proved to be a valuable design feature. In addition, it is possible to drive the subject's legs passively

during periods of rest to increase joint mobility and circulation in the extremities. Once the subject gained sufficient strength, the motor can be used as a programmable load.

Finally, the FNS-controller can be used for applications other than cycling, such as the assessment of patient response to different FNS waveforms and the measurement of muscle torques produced by FNS.

6.3. DISCUSSION AND FUTURE DEVELOPMENTS

The FNS training sessions with the FNS-controller / Paracycle system showed the suitability of this system to create a smooth cycling movement in paraplegic subjects. The editing facilities of FNS-controller allow a speedy definition of stimulation sequences in a laboratory environment, mostly on a trial and error basis. As EMG studies undertaken on the Paracycle did not yield suitable stimulation sequences, the need for a model of the muscular activity of the lower extremities during cycling is anticipated. Because FNS uses only a small population of the muscles normally envolved in cycling such a model should be based on the action of the muscle groups in question rather than EMG analysis. Furthermore, the possibility to change the stimulation levels in different muscle groups during the stimulation would enhance the definition of stimulation sequences even further. Because of the use of TURBO PASCAL and its speed this could be implemented into the current stimulation control procedures without difficulties and without slowing the control process down too much.

In the present design the stimulation level is varied by defining the pulse widths. However, the assessment of the knee

torque versus pulse width characteristic could be used to allow the definition of knee torques directly in Newton meters. Furthermore, the stimulation frequency could be defined differently for each muscle (or as a function of crank position) to allow stimulation with as little fatigue as possible.

During the patient trials it was observed that the instantaneous crank speed is a good indication for the need of change of stimulation levels in the particular muscles or for motor assist or braking. In the current design crank speed is measured only every revolution. However, with additional hardware the shaft encoder would allow the measurement of crank speed 256 times per revolution. The crank speed measurement would thus allow the implementation of a closed loop algorithm with the control aim of a defined constant crank speed.

The feature of using a motor to assist or load in dependence of the crank position proved very valuable. For the implementation of closed loop algorithms, however, the motor controller needs to be improved in respect of control linearity and speed. With the current design, it is possible to vary the motor speed open loop via the motor voltage. The actual motor current can be sensed and a display of the motor power is provided. However, it would be more appropriate to design a motor controller which incorporates already a hardware feedback to allow the absolute setting of motor power. Thus, it would be possible to set the assist or load power without using up valuable processing time of the FNS-controller.

Ultimately for a closed loop FNS-controller, a measurement of the absolute pedal forces in all three dimensions is necessary to assess the effectiveness of the stimulation and to ensure the subjects' safety. Such a measurement would also allow the

detection of spastic contractions during stimulation which were encountered several times at the beginning of training sessions with one of the subjects showing signs of moderate spasticity.

The anticipation with which the paraplegic subjects reacted after they were able to move with the Paracycle a few meters showed the need for a portable control unit which can provide them true mobility over longer distances. Stimulation sequences could be loaded from the FNS-controller into this unit, which then would control the stimulation in a manner similar to that described earlier.

Chapter 7 : CONCLUSIONS

The hardware and software design of a functional neuromuscular stimulation (FNS) system for the modulated control of the lower extremities during cycling is described. The systems hardware is designed around a standard IBM compatible Personal Computer (PC). The hardware consists of three plug-in boards for the PC: (a) a 16 channel 12-bit A/D converter which is used to sense control parameters and output voltage of the stimulator; (b) a 6 channel stimulation waveform generator; (c) an interface for an 8 bit shaft encoder which senses the pedal position. Three programmable timers and two 8 bit D/A converters per channel are used to generate the desired stimulation waveforms. Monophasic or biphasic mode of stimulation as well as other waveform characteristics such as frequency, width and amplitude of the pulse can be defined as a function of crank position, independently for each channel. A separate unit includes 6 amplifiers with pulse-transformers to provide complete isolation of the patient. An extensive software package is provided (written in TURBO PASCAL) to support the user of the system with interactive video graphics in the definition of the stimulation sequences and other relevant parameters. These data are stored in templates which are used to control the stimulation waveforms generated by the customized expansion boards. Templates and relevant patient information can be stored on floppy disk and printed on a graphics printer. The performance of the described FNS-controller/Paracycle system was evaluated in a controlled case study with 2 paraplegic subjects.

APPENDIX A : THE SPERRY PERSONAL COMPUTER (PC)

The components of the SPERRY PC are described in chapter 4. This appendix deals with the information which is necessary to interface custom made plug in boards for the expansion slots of the IBM PC compatible SPERRY PC.

The heart of the SPERRY PC is the INTEL 8088-2 microprocessor. This processor is an 8-bit external bus version of INTEL's 16-bit 8086 processor, and is software-compatible with the 8086. Thus, the 8088-2 supports 16-bit operations, including multiply and divide, and supports 20 bits of addressing (ie. 1 megabyte of storage). The clock frequency of the 8088-2 (which is a faster version of the 8088) can be selected by a hardware switch as 4.77 or 7.16 MHz. The 4.77 MHz setting being the speed of the original IBM personal computer. The clock frequency of 7.16 MHz gives the SPERRY personal computer a speed advantage of approximately 40 % compared with the IBM original.

Figure A.1 shows the signal lines which are available on the IBM expansion slot. Table A1, A2, A3 give a short description of the function of each signal line. Figure A.2 and Figure A.3 show the timing diagram of the IBM bus during $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$ cycles respectively.

For further reading the technical reference manual for the IBM personal computer (IBM, 1983) and the 8086/8088 Primer (Morse, 1982) are recommended.

I/O Channel

The I/O channel is an extension of the 8088 microprocessor bus. It is,.however, demultiplexed, repowered, and enhanced by the addition of interrupts and direct memory access (DMA) functions.

The I/O channel contains an 8-bit, bidirectional data bus, 20 address lines, 6 levels of interrupt, control lines for memory and I/O read or write, clock and timing lines, 3 channels of DMA control lines, memory refresh timing control lines, a channel-check line, and power and ground for the adapters. Four voltage levels are provided for I/O cards: +5 Vdc, -5 Vdc, +12 Vdc, and -12 Vdc. These functions are provided in a 62-pin connector with 100-mil card tab spacing.

A 'ready' line is available on the I/O channel to allow operation with slow I/O or memory devices. If the channel's ready line is not activated by an addressed device, all processor-generated memory read and write cycles take four 210-ns clock or 840-ns/byte. All processor-generated I/O read and write cycles require five clocks for a cycle time of 1.05 μs/byte. All DMA transfers require five clocks for a cycle time of 1.05 μs/byte. Refresh cycles occur once every 72 clocks (approximately 15 μs) and require four clocks or approximately 7% of the bus bandwidth.

I/O devices are addressed using I/O mapped address space. The channel is designed so that 512 I/O device addresses are available to the I/O channel cards.

A 'channel check' line exists for reporting error conditions to the processor. Activating this line results in a Non-Maskable Interrupt (NMI) to the 8088 processor. Memory expansion options use this line to report parity errors.

The I/O channel is repowered to provide sufficient drive to power all five system unit expansion slots, assuming two low-power Schottky loads per slot. The IBM I/O adapters typically use only one load.

The following pages describe the system board's I/O channel.

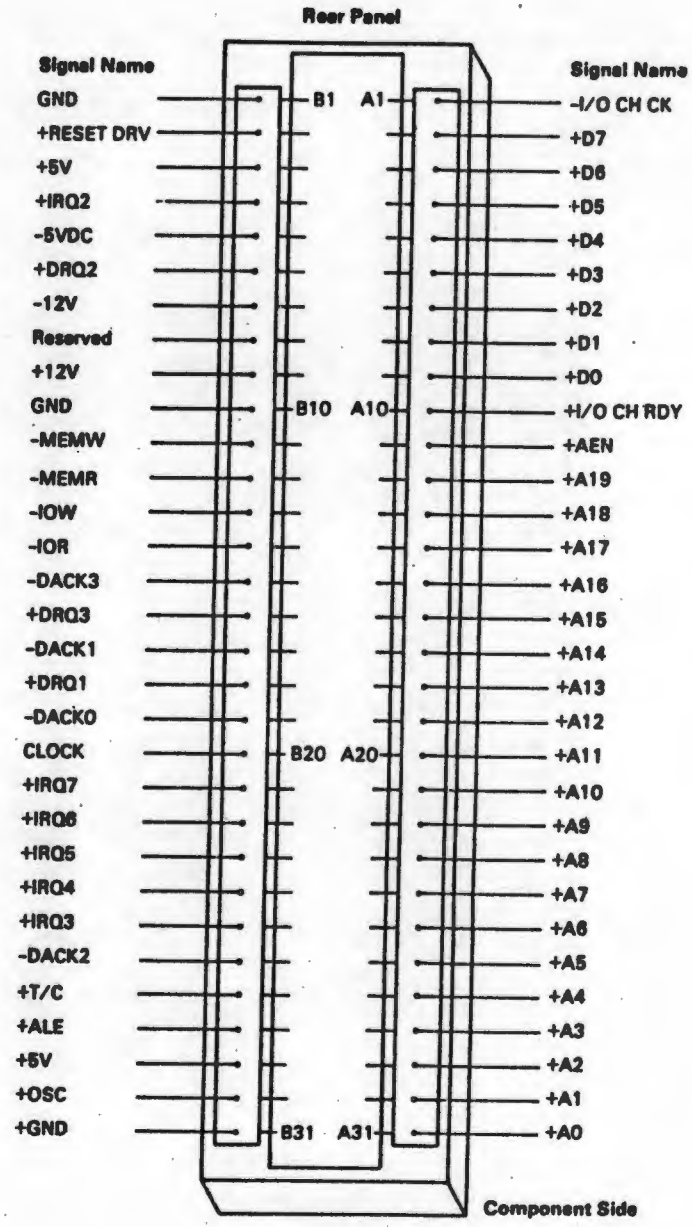


Figure A 1 IBM I/O-Channel

Table A.1 : Signal lines IBM I/O channel

I/O Channel Description

The following is a description of the IBM Personal Computer I/O Channel. All lines are TTL-compatible.

Signal	I/O	Description
OSC	O	Oscillator: High-speed clock with a 70-ns period (14.31818 MHz). It has a 50% duty cycle.
CLK	O	System clock: It is a divide-by-three of the oscillator and has a period of 210 ns (4.77 MHz). The clock has a 33% duty cycle.
RESET DRV	O	This line is used to reset or initialize system logic upon power-up or during a low line voltage outage. This signal is synchronized to the falling edge of clock and is active high.
A0-A19	O	Address bits 0 to 19: These lines are used to address memory and I/O devices within the system. The 20 address lines allow access of up to 1 megabyte of memory. A0 is the least significant bit (LSB) and A19 is the most significant bit (MSB). These lines are generated by either the processor or DMA controller. They are active high.
D0-D7	I/O	Data Bits 0 to 7: These lines provide data bus bits 0 to 7 for the processor, memory, and I/O devices. D0 is the least significant bit (LSB) and D7 is the most significant bit (MSB). These lines are active high.
ALE	O	Address Latch Enable: This line is provided by the 8288 Bus Controller and is used on the system board to latch valid addresses from the processor. It is available to the I/O channel as an indicator of a valid processor address (when used with AEN). Processor addresses are latched with the failing edge of ALE.
$\overline{\text{I/O CH CK}}$	I	-I/O Channel Check: This line provides the processor with parity (error) information on memory or devices in the I/O channel. When this signal is active low, a parity error is indicated.

Table A.2 : Signal lines IBM I/O channel (cont'd)

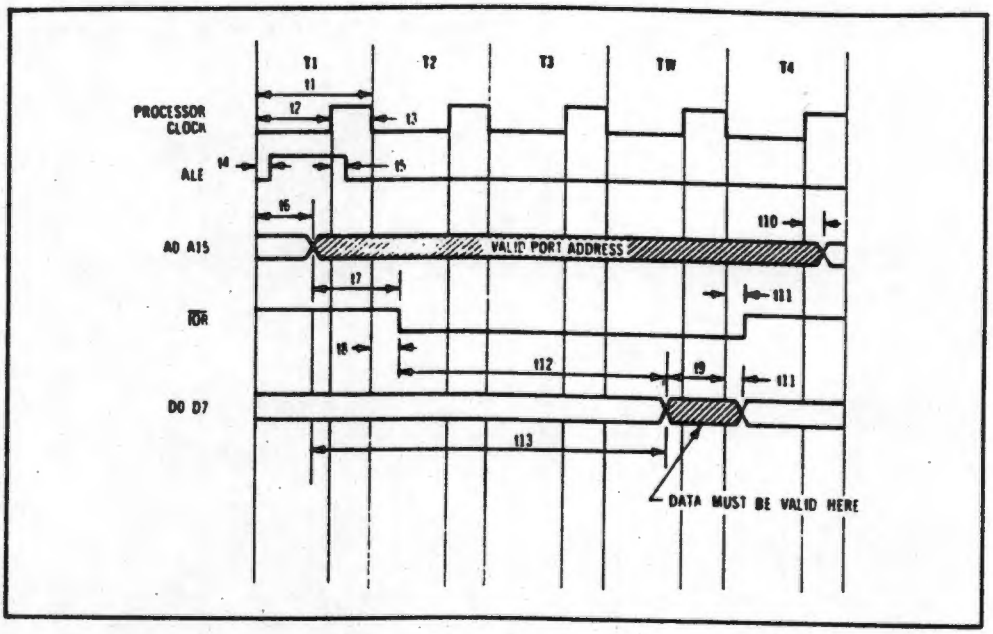
Signal	I/O	Description
I/O CH RDY	I	I/O Channel Ready: This line, normally high (ready), is pulled low (not ready) by a memory or I/O device to lengthen I/O or memory cycles. It allows slower devices to attach to the I/O channel with a minimum of difficulty. Any slow device using this line should drive it low immediately upon detecting a valid address and a read or write command. This line should never be held low longer than 10 clock cycles. Machine cycles (I/O or memory) are extended by an integral number of CLK cycles (210 ns).
IRQ2-IRQ7	I	Interrupt Request 2 to 7: These lines are used to signal the processor that an I/O device requires attention. They are prioritized with IRQ2 as the highest priority and IRQ7 as the lowest. An Interrupt Request is generated by raising an IRQ line (low to high) and holding it high until it is acknowledged by the processor (interrupt service routine).
$\overline{\text{IOR}}$	O	-I/O Read Command: This command line instructs an I/O device to drive its data onto the data bus. It may be driven by the processor or the DMA controller. This signal is active low.
$\overline{\text{IOW}}$	O	-I/O Write Command: This command line instructs an I/O device to read the data on the data bus. It may be driven by the processor or the DMA controller. This signal is active low.
$\overline{\text{MEMR}}$	O	Memory Read Command: This command line instructs the memory to drive its data onto the data bus. It may be driven by the processor or the DMA controller. This signal is active low.
$\overline{\text{MEMW}}$	O	Memory Write Command: This command line instructs the memory to store the data present on the data bus. It may be driven by the processor or the DMA controller. This signal is active low.

Table A.3 : Signal lines IBM I/O channel (cont'd)

Signal	I/O	Description
DRQ1-DRQ3	I	DMA Request 1 to 3: These lines are asynchronous channel requests used by peripheral devices to gain DMA service. They are prioritized with DRQ3 being the lowest and DRQ1 being the highest. A request is generated by bringing a DRQ line to an active level (high). A DRQ line must be held high until the corresponding DACK line goes active.
$\overline{\text{DACK0}}$-$\overline{\text{DACK3}}$	O	-DMA Acknowledge 0 to 3: These lines are used to acknowledge DMA requests (DRQ1-DRQ3) and to refresh system dynamic memory (DACK0). They are active low.
AEN	O	Address Enable: This line is used to de-gate the processor and other devices from the I/O channel to allow DMA transfers to take place. When this line is active (high), the DMA controller has control of the address bus, data bus, read command lines (memory and I/O), and the write command lines (memory and I/O).
T/C	O	Terminal Count: This line provides a pulse when the terminal count for any DMA channel is reached. This signal is active high.

The following voltages are available on the system-board I/O channel:

+5 Vdc ± 5%, located on 2 connector pins
-5 Vdc ± 10%, located on 1 connector pin
+12 Vdc ± 5%, located on 1 connector pin
-12 Vdc ± 10%, located on 1 connector pin
GND (Ground), located on 3 connector pins



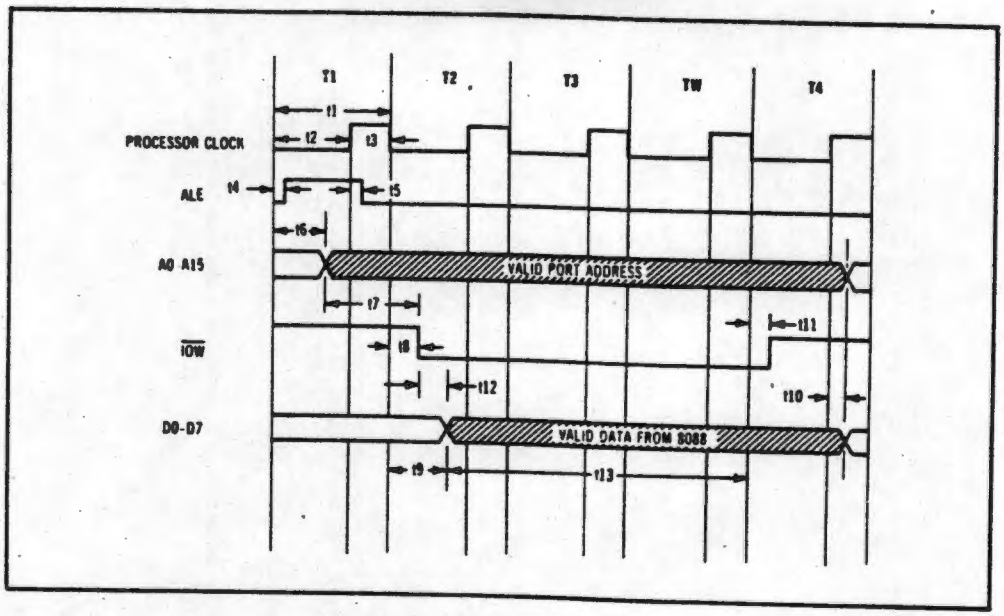
I/O port read bus cycle timings.

I/O Port Read Bus Cycle Timings

Symbol	Max	Min
t1	—	209.5
t2	—	124.5
t3	—	71.8
t4	15	—
t5	15	—
t6	128	16
t7	—	91.5
t8	35	10
t9	—	42
t10	—	10
t11	35	10
t12	—	551.5
t13	—	668

*All times are in nanoseconds.

Figure A.2 I/O read-cycle IBM expansion bus



I/O port write bus cycle timings.

I/O Port Write Bus Cycle Timings

Symbol	Max	Min
t1	–	209.5
t2	–	124.5
t3	–	71.8
t4	15	–
t5	15	–
t6	128	16
t7	–	91.5
t8	35	10
t9	122	14
t10	–	10
t11	35	10
t12	112	–
t13	–	306.5

*All times are in nanoseconds.

Figure A.3 I/O write-cycle IBM I/O channel

APPENDIX B : STIMULATION WAVEFORM GENERATION BOARD

Aim: To design a computer controlled stimulator for the generation of biphasic and monophasic rectangular waveforms for Functional Neuromuscular Stimulation (FNS). As this stimulator will be primarily used for research purpose to study the influences of various stimulation waveforms and frequencies, each channel can be controlled independently. The following parameters are under dynamic software control (ie. in realtime):

Specifications:

- Amplitude of the negative pulse : $0\ mA < A_- < 100\ mA$
- Duration of the negative pulse : $0\ \mu s < t_- < 800\ \mu s$
- Amplitude of the positive pulse : $0\ mA < A_+ < 100\ mA$
- Duration of the positive pulse : $0\ \mu s < t_+ < 4000\ \mu s$
- Interpulse interval : $200\ \mu s < IPI < 4000\ \mu s$
- Pulse repetition interval : $10\ ms < T < 100\ ms$

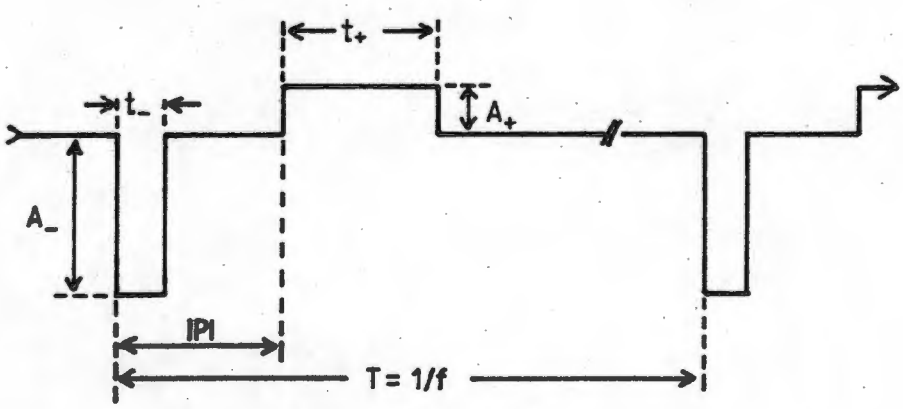


Figure B.1 Charge balanced biphasic FNS waveform

CONTINUOUS SIGNAL GENERATION

The waveform illustrated in Figure B.1 can be defined by software and is continuously produced by a dedicated timer module for each FNS channel. Whenever it is necessary to change one or more of the above-mentioned parameters the new parameter values are written to the respective timer module. Changes regarding the waveform timing (ie. pulse duration, interpulse interval, repetition frequency) take place after the current cycle has been completed. Thus the continuum of the waveform generation is ensured. Changes in pulse amplitude however have immediate influence on the waveform generated.

SIGNAL GENERATION VIA SOFTWARE TRIGGER

The timer modules can be triggered by software routines in order to generate more complex waveforms.

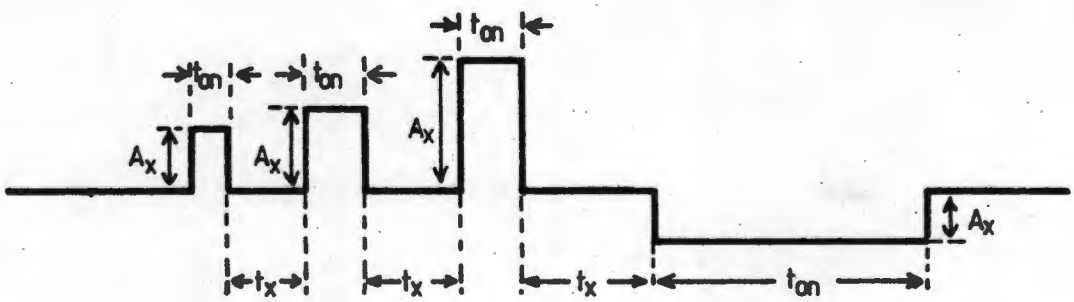


Figure B.2 Hypothetical FNS waveform

A waveform as shown in Figure B.2 can be obtained if the timer modules, which are programmed as single shots, are initialized (ie. triggered) by a 'write to latch' command. In this way it is possible to control the timing t_x between the generated pulses via software time loops. The durations t_{on} and the amplitudes A_x of the generated pulses are controlled by the timer modules. Thus

the computer is freed to perform other important tasks.

This method of pulse generation is used to generate the 100 µs long test pulse for the electrode impedance check (see appendix H, procedure 'impedance_check' for details).

HARDWARE DESCRIPTION

BUS BUFFERING

Figure B.3 shows the circuit diagram of the bus-buffering, address decoding and the wait state circuitry. Bus lines to and from the IBM bus connector are buffered in order to keep the load imposed on to the Personal Computer as small as possible (the only lines loaded with two LS-TTL IC's are CLK (B20), $\overline{IOW}$ (B13) and Reset-drive (Resdrv, B2)). As data transfer occurs only from the CPU to the stimulation timer modules, an unidirectional octal data buffer (IC12) is sufficient for the datalines. The data buffer is enabled by the IBM's $\overline{IOW}$ signal.

ADDRESS DECODING (see Figure B.3)

The I/O address of the stimulation board can be selected as 800, 840, 880 or 8C0 Hex. The address decode logic (IC1, 2, 6, 7, 8, 10) does not use line A15. That means that if any other part of the IBM PC uses an I/O address between 8800 and 88FF hexadecimal, there could be an I/O conflict. Most of the boards built to be compatible with the IBM PC and the computer itself use I/O addresses of 1000 hexadecimal or less. This puts these I/O addresses safely above the system I/O. This kind of multi-addressing is commonly traded off for lower chip count and easier PC-layout. Two one-out-of-eight decoders (IC 8,10) generate the chip-select signals for the 6 timer chips and 7 D/A converters respectively.

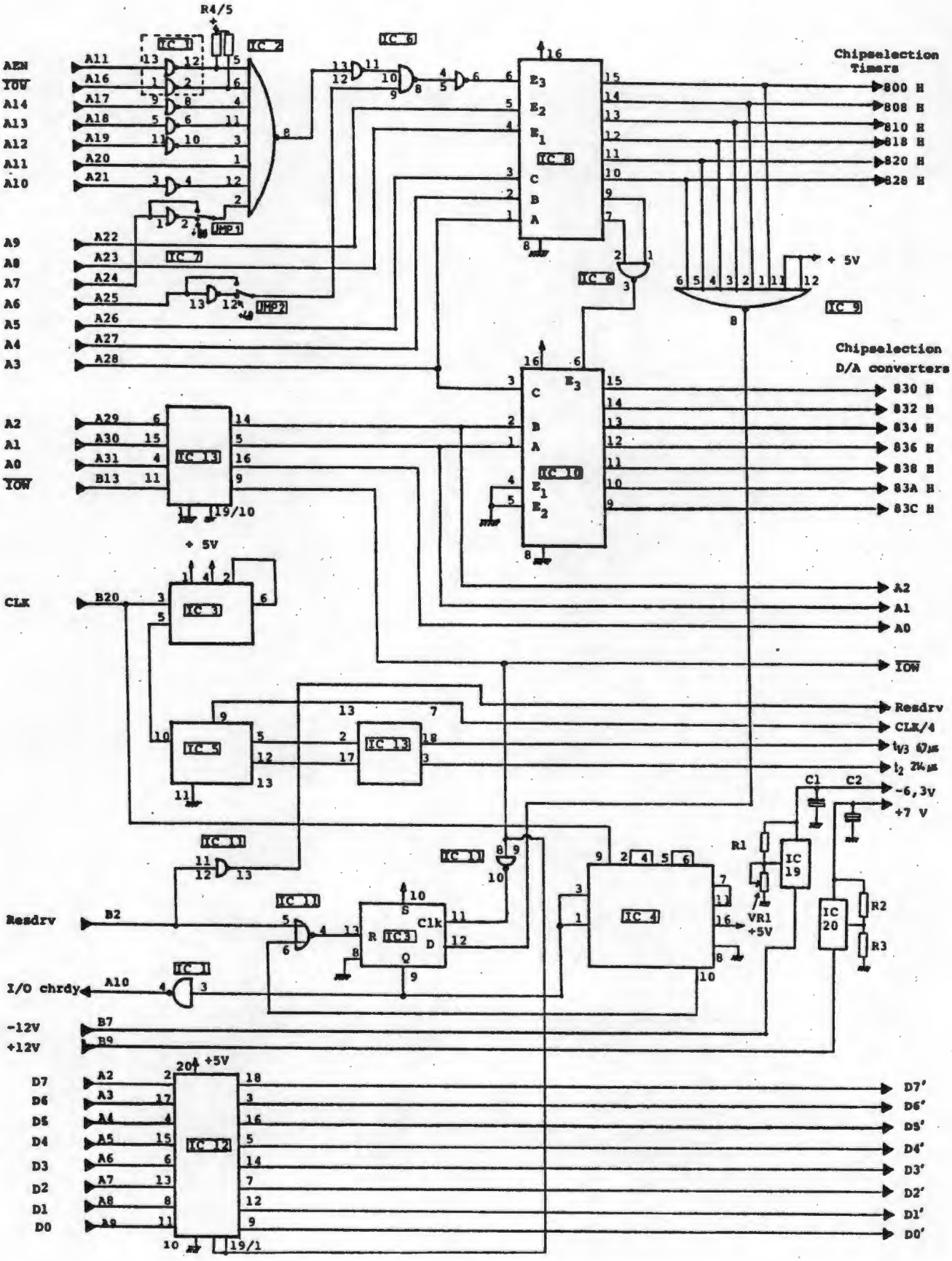


Figure B.3 Circuit diagram waveform generation board (sheet 1 of 3)

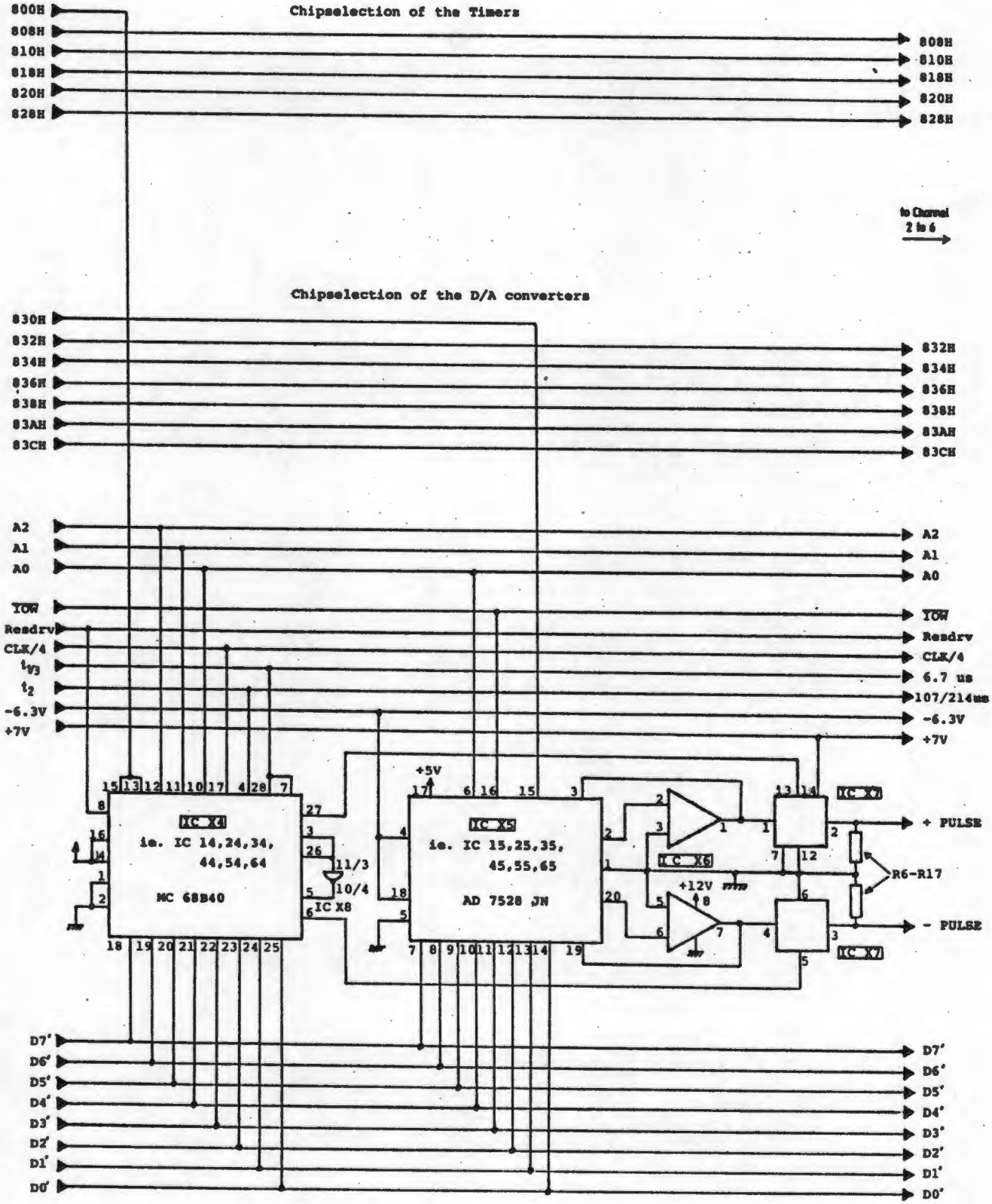


Figure B.4 Circuit diagram waveform generation board (sheet 2 of 3)

INTERFACING THE TIMER CHIP MOTOROLA MC 6840

The waveform generation for each channel is realized by one Motorola MC 6840 chip (Motorola, 1979). This device contains 3 independently programmable timers which can be operated in various modes. The main advantage of the Motorola 6840 over the INTEL timers 8253 or 8254 is that the 6840 allows the generation of a rectangular waveform with variable duty cycle.

Figure B.5 illustrates the MC 6840 chip interface to the IBM's 8088 INTEL bus which comprises eight unidirectional datalines (D0-D7), three register select lines (A0-A2), one chip select line (connected to CS0 and R/$\overline{W}$) and one reset line (Reset).

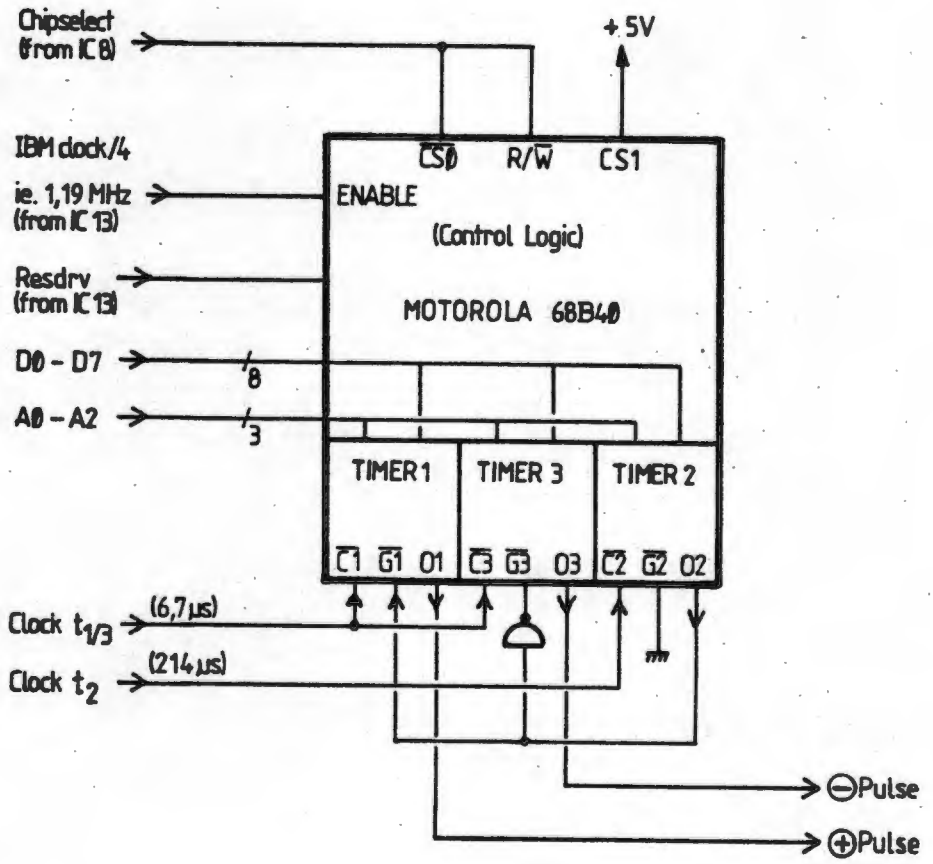


Figure B.5 Interface 68B40 chip to the IBM-bus

As the ENABLE line which normally controls the data transfer from a 6800 microprocessor to the 6840 has no equivalent on the INTEL 8088 bus, a special solution had to be found. There are three fundamental prerequisites in choosing the right ENABLE signal for the MC 68B40 (2.5 MHz version) in this application (see Figure B.6)

1. The repetition frequency of the ENABLE signal may be maximum 2.5 MHz (minimum high time is 220ns, minimum low time is 210ns).
2. For data transfer, ENABLE has to go high at least 70 ns after the chip select lines ($\overline{CS0}$, CS1), the register select lines RS0,RS1,RS2, and the read/write line (R/$\overline{W}$) are activated.
3. Changes at the external clock and gate inputs ($\overline{C1}$,$\overline{C2}$,$\overline{C3}$ and $\overline{G1}$,$\overline{G2}$,$\overline{G3}$ are recognized and clocked in by ENABLE pulses (4 ENABLE periods are used to synchronize and process the external clock and decrement the internal counters).

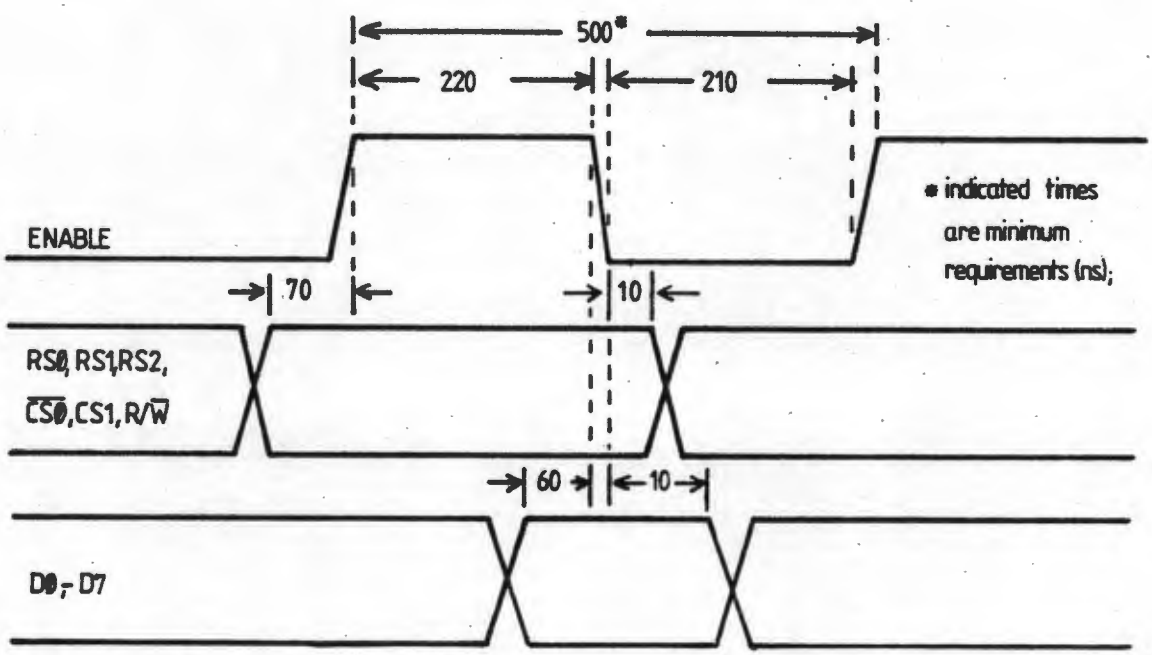


Figure B.6 Timing diagram write data to the 68B40 chip

Figure B.7 illustrates the bus timing for interfacing the MC 68B40 to the IBM PC bus (see as well interface diagram Figure B.5). To generate an ENABLE signal that meets all the abovementioned requirements the IBM's system clock (4.77 MHz) is divided by 4 (signal clk/4). The duty cycle is 50 % so that on- and off-durations are 420 ns each. In order to speed up data transfer to the 6840 a higher ENABLE frequency would have been desirable, but division of the IBM clock by three would have yielded an ENABLE signal with a high time of 210 ns which is just outside the MC 68B40's limit of 220 ns. To ensure that a positive transition of the ENABLE signal (which triggers the data transfer from the IBM to the MC 68B40 timer) takes place during the $\overline{IOW}$ low period, the IOW low period must be at least 2 * 420 ns + setuptime long. The minimum setup time from address, R/$\overline{W}$ and chipselect valid to the positive transition of ENABLE is 70 ns.

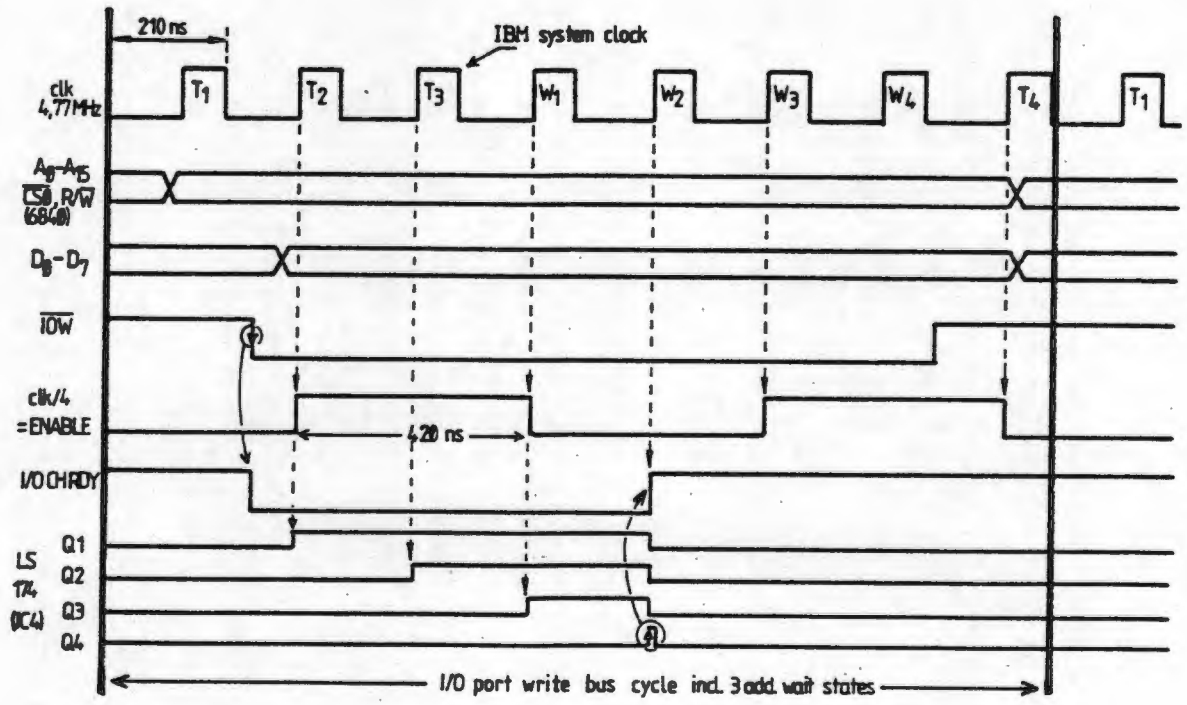


Figure B.7 Bus timing 68B40 to IBM-bus

In the worst case the ENABLE signal goes high a little less than 70 ns after IOW goes low (ie. R/$\overline{W}$ high). In this case it takes another 840 ns (time period of the ENABLE signal to the next positive transition of ENABLE. After this transition the data has still to be kept stable for at least 10 ns to ensure a safe data transfer. This sums up to 838 ns + 70 ns + 10 ns = 918 ns. To be on the safe side we add another 80 ns for data delay (gate delays) in the address-decoding etc.. Thus the $\overline{IOW}$ cycle has to be low for at least 1000ns or 1 µs. As the normal $\overline{IOW}$ cycle of the IBM bus is only three clock cycles long, ie 3 * 210 ns = 630 ns, at least two additional 'waitcycles' of 210 ns each, have to be inserted to meet the abovementioned timing requirements. For safety reasons (medical equipment !) three additional wait cycles are inserted to stretch the $\overline{IOW}$ cycle to 1260 ns.

Figure B.8 shows the circuit which is used to generate three additional wait cycles during $\overline{IOW}$ data transfers to the MC 68B40 timer.

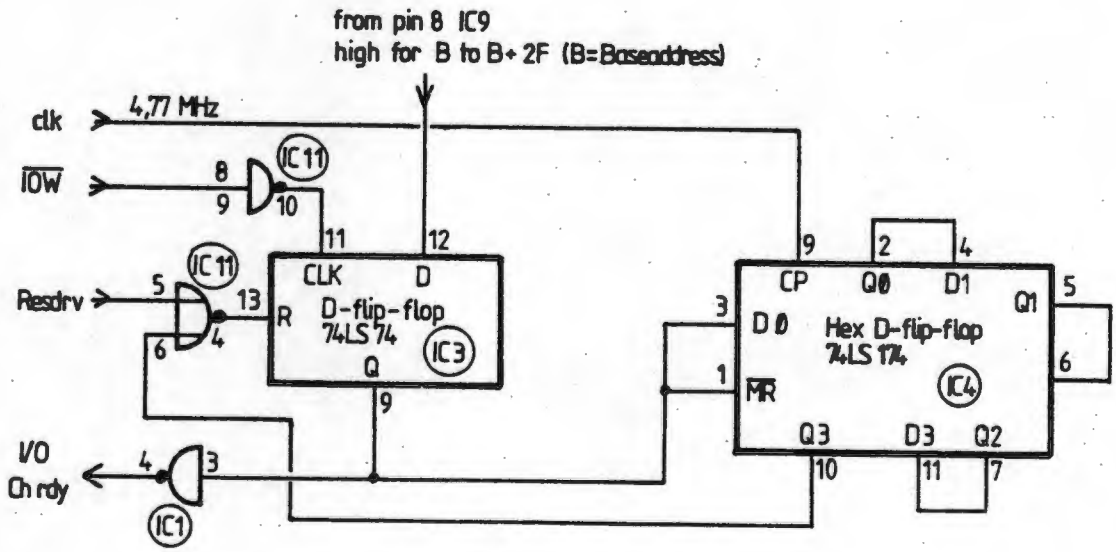


Figure B.8 Circuitry for wait state generation

In this circuit the significant address range that is to have the extra wait cycles (address range Ba+00 to Ba+28 Hex., where Ba is the base address of the waveform generation board) is detected by IC 9 (Figure B.3). The falling edge of the $\overline{IOW}$ bus signal is used to sample the output of IC 9 and set the SN 74LS74 latch (IC3) (see figure B.8). This latch upon being set will pull the I/Ochrdy signal on the IBM PC bus down to signal that the $\overline{IOW}$ transfer is not finished. The SN 74LS174 hex D-Flipflop (IC 4), acting as a shiftregister will count the clock cycles and, thus, the wait cycles generated, before its output Q3 is used to clear the IC3 latch. When this latch is cleared, I/Ochrdy will become high and the $\overline{IOW}$ bus cycle will end. The number of inserted wait cycles is equal to the number of clock cycles occurring while the I/Ochrdy line is low. Thus 4 wait cycles are inserted. As the normal I/O write cycle inserts one wait state anyway, three additional wait cycles (3*210 ns = 630 ns) are added (IBM technical reference manual page 1.18, IBM 1983)

GENERATION OF THE STIMULATION WAVEFORMS

The waveform generation for each channel is realized by one Motorola MC 68B40 timer chip. This timer IC contains three independently programmable timers which are interconnected as shown in Figure B.9 .

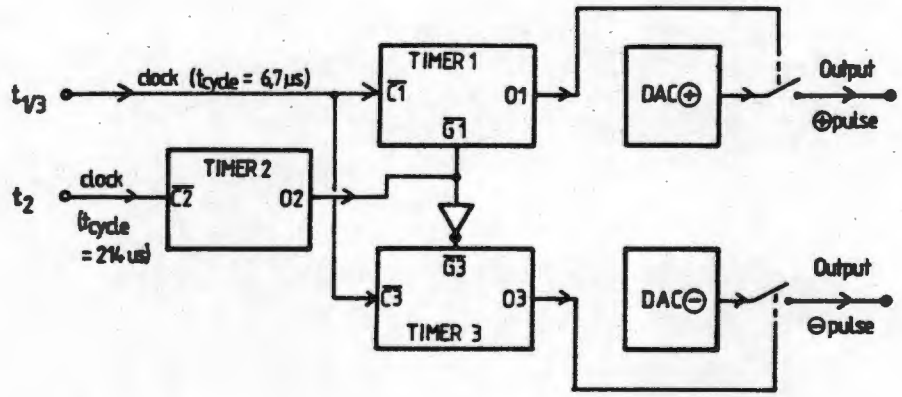


Figure B.9 Block diagram of the one waveform generation channel

Timer 2 generates a squarewave with programmable frequency and duty cycle. Timers 1 and 3 are programmed as single shots with programmable pulse duration. These single shots are triggered by a negative transition on their gate inputs. Because an inverter is inserted between the output of timer 2 and the gate input of timer 3 timer 3 is triggered by positive transitions and timer 1 by negative transitions of the output of timer 2. (see Figure 9.10). The time interval T of the squarewave generated by timer 2 determines the repetition frequency (1/T) of the pulses generated by timer 3 and 1. The time the output of Timer 2 is high is

equivalent to the interval between the pulse generated by timer 3 and 1. To give a typical biphasic FNS waveform (see Figure B.1) the signal of timer 3 is inverted in the isolation amplifier to give a negative going pulse. The amplitudes of both the negative and positive going pulses are programmable from 0 to +5 Volts.

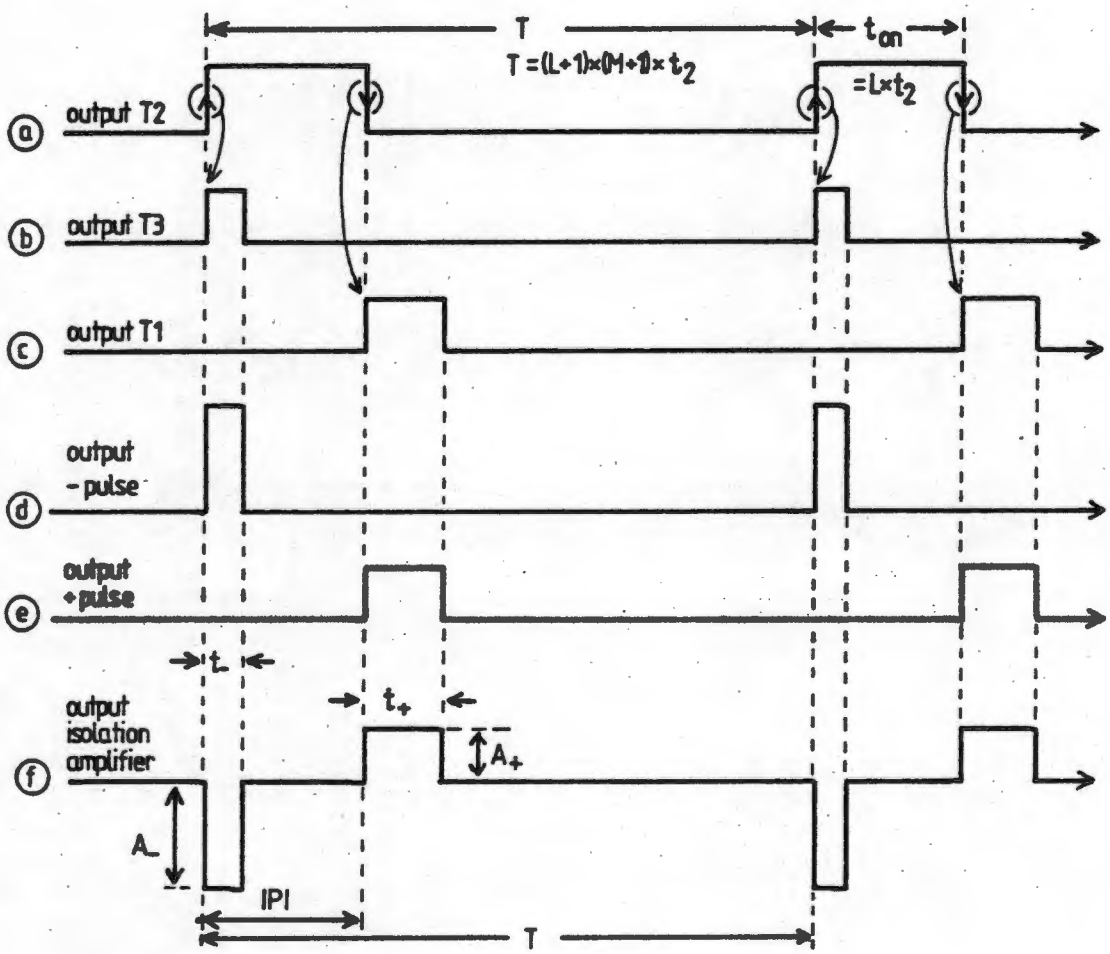


Figure B.10 Timing diagram of one waveform generation channel

THE 68B40 TIMER CHIP (see Figure B.11)

The MC68B40 timer chip is a programmable subsystem component of the 6800 microprocessor family of Motorola, designed to provide variable system time intervals. The MC68B40 has three 16-bit binary counters, three corresponding control registers and a status register. These counters are under software control and can be used to cause system interrupts or to generate rectangular waveforms of various duty cycles (Motorola, 1979).

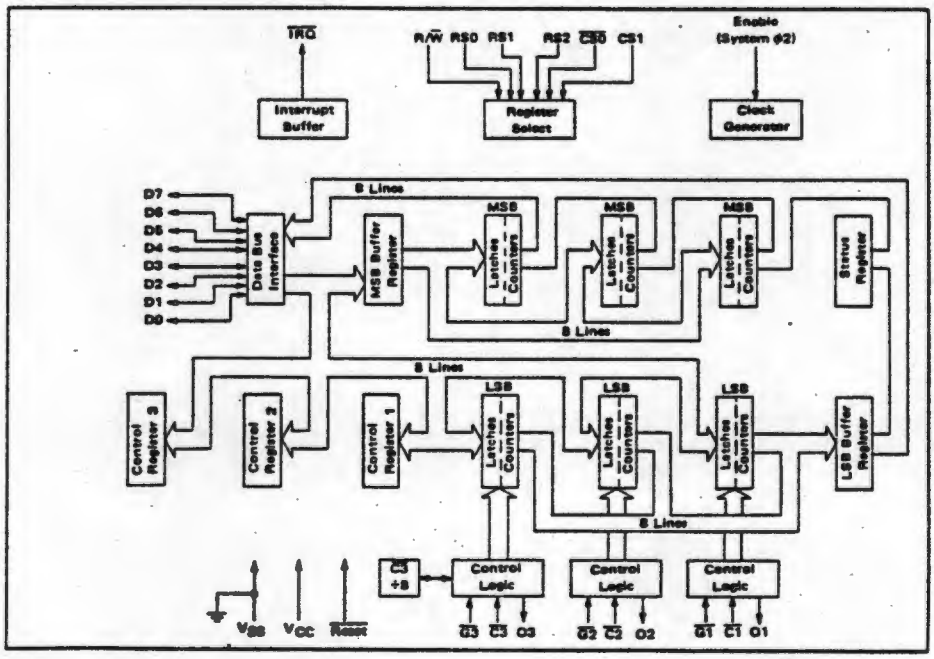


Figure B.11 Internal block diagram 68B40 timer chip

Control registers of the MC68B40

The three timers of the MC 68B40 can be accessed via a set of six registers for data transfer and three control registers. These registers are located within eight physical I/O port addresses which are selected by the register select inputs RS0, RS1 and RS2 (see Table B.1). To each timer two registers are dedicated for

data transfer. This leaves, however, only two registers for the accommodation of the three control registers. The control register for timer 2 has an unique address-space (RS0=1, RS1=0, RS2=0) and can be written to any time. The remaining control registers for timer 1 and 3 share the address-space selected by a logic 0 on all register select inputs. The least significant bit of the control register of timer 2 is used as an additional addressing for the distinction between control register of timer 1 and 3 (see Table B.2). Thus , with all register select inputs logic zero and R/$\overline{W}$ = 0 control register of timer 1 will be written into if the least significant bit of control register CR2 is a logic one (ie.CR20=1). Under the same conditions control register CR3 will be written into if CR20 is logic 0 (ie.CR20=0). (In Motorola terminology CR20 means bit 0 of the control register 2. This slightly confusing terminology is maintained to keep this write up compatible with the motorola data sheet of the 6840 timer chip). Control register 3 (CR3) can also be written into after a RESET low condition has occurred, since all control register bits (except CR10) are cleared. Therefore, one may write to the control registers in the sequence CR3, CR2, CR1.

The least significant bit of control register CR1 ie.CR10 is used as an internal reset bit (for function of the control register bits see Table B.2). When this bit is a logic zero, all timers are allowed to operate in the mode prescribed by the remaining bits of the control registers. Writing a one into CR10 causes all counters to be preset with the contents of the corresponding counter latches, all counter clocks to be disabled, and the timer outputs and interrupt flags (ie. status register) to be reset. Counter latches and control registers are undisturbed by an internal reset and may be written into regardless of the state of CR10.

TABLE B.1 REGISTER SELECTION

Register Select Inputs			Operations	
RS2	RS1	RS0	R/W = 0	R/W = 1
0	0	0	CR20 = 0 Write Control Register = 3 CR20 = 1 Write Control Register = 1	No Operation
0	0	1	Write Control Register #2	Read Status Register
0	1	0	Write MSB Buffer Register	Read Timer #1 Counter
0	1	1	Write Timer #1 Latches	Read LSB Buffer Register
1	0	0	Write MSB Buffer Register	Read Timer #2 Counter
1	0	1	Write Timer #2 Latches	Read LSB Buffer Register
1	1	0	Write MSB Buffer Register	Read Timer #3 Counter
1	1	1	Write Timer #3 Latches	Read LSB Buffer Register

Counter latch initialization

Each of the three independent timers consists of a 16 bit addressable counter and a 16 bit addressable latch (see block diagram in Figure B.11. The counters are preset to the binary numbers stored in the latches. Counter initialization results in the transfer of the latch contents to the respective counter. Since the MC 68B40 data bus is only 8-bits wide and the counters are 16 bits wide, a temporary register (most significant byte buffer) is provided. This "write only" register is for the most significant byte of the desired latch data.

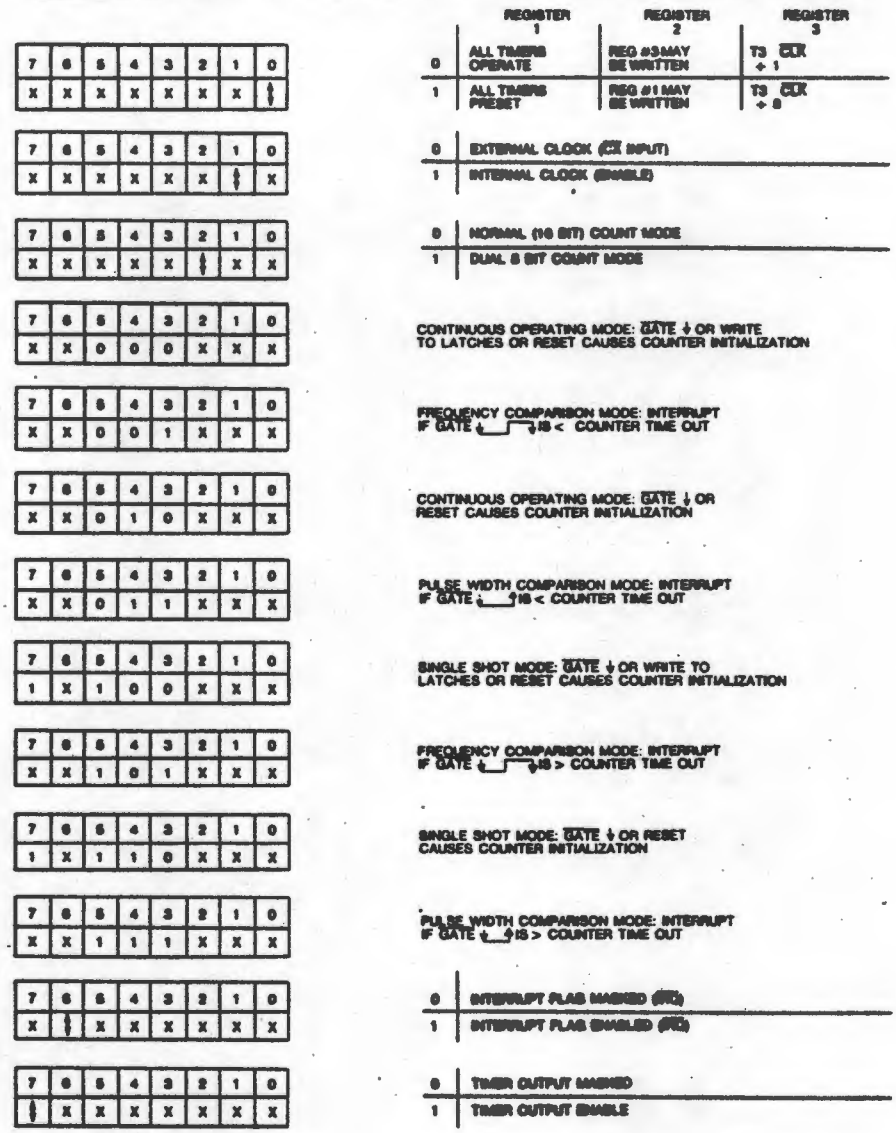
TABLE B.2 CONTROL REGISTER BITS

TABLE 2 – CONTROL REGISTER BITS

CR10 Internal Reset Bit	CR20 Control Register Address Bit	CR30 Timer #3 Clock Control
0 All timers allowed to operate 1 All timers held in preset state	0 CR#3 may be written 1 CR#1 may be written	0 T3 $\overline{\text{Clock}}$ is not prescaled 1 T3 $\overline{\text{Clock}}$ is prescaled by ÷8
CRX1*	Timer #X $\overline{\text{Clock}}$ Source	
0 1	TX uses external clock source on $\overline{\text{CX}}$ input TX uses Enable clock	
CRX2	Timer #X Counting Mode Control	
0 1	TX configured for normal (16-bit) counting mode TX configured for dual 8-bit counting mode	
CRX3 CRX4 CRX5	Timer #X Counter Mode and Interrupt Control (See Table 3).	
CRX6	Timer #X Interrupt Enable	
0 1	Interrupt Flag masked on $\overline{\text{IRQ}}$ Interrupt Flag enabled to $\overline{\text{IRQ}}$	
CRX7	Timer #X Counter Output Enable	
0 1	TX Output masked on output OX TX Output enabled on output OX	

*Control Register for Timer 1, 2, or 3, Bit 1.

TABLE B.3 CONTROL REGISTER PROGRAMMING



Three addresses are provided for the MSB-buffer register (see Table B.1) but they all lead to the same buffer. Data from the MSB-buffer will automatically be transferred into the most significant byte of the respective timer latch when the least significant byte is transferred to the same latch.

Counter Initialization

Counter initialization is defined as the transfer of data from the latches to the counter with subsequent clearing of the individual interrupt flags associated with the counter. Counter

initialization always occurs when a reset condition is recognized (ie. Reset pin = 0; or bit 0 of control register 1 is 0). It can also occur - depending on the timer mode - with a write to latches command or recognition of a negative transition at the gate input. Counter recycling (ie. reinitialization) occurs when a negative transition of the clock input is recognized after a counter has reached an all-zero state. In this case, data is transferred from the latches to the counter.

Asynchronous input/output lines

The external clock inputs ($\overline{C1}$, $\overline{C2}$, $\overline{C3}$) will accept asynchronous TTL voltage level signals to decrement timers 1, 2 or 3 respectively, the output signals of which appear in Figure B.10a,b,c. The high and low levels of these clocks must each be stable for at least one ENABLE system clock period plus the sum of setup and hold times for the signals. We use the 4.77 MHz clock divided by 4 as ENABLE system clock, so the clock period is 840 ns long. Setup and hold times for the MC 68B40 sum up to 125 ns. Thus the minimum pulse duration for one external clock impulse is 965 ns. As the duration of the high/low states of the external clock for timer 1 and 3 is 3.35 µs this timing requirement is easily met.

Gate inputs

The gate inputs $\overline{G1}$, $\overline{G3}$ accept asynchronous TTL compatible signals which are used to trigger timer 1 and 3 by the output of timer 2. A negative transition at the $\overline{G1}$ or $\overline{G3}$ input triggers the respective timer (which is programmed as a single shot).

Programming the timer output waveform

Output waveform definition is accomplished by selecting either single 16 bit or dual 8 bit operating modes. The single 16 bit mode will produce a square-wave with 50 % duty cycle in the continuous timer mode, and will produce a single pulse in the single shot mode. The dual 8 bit mode will produce a square-wave with variable duty cycle in both the continuous and single shot mode.

Figure B.10 shows the principle of generating a typical functional electrical stimulus. Note that the pulses generated by timer 3 will be inverted in the isolation amplifier and not on the waveform generator board.

The FNS waveform parameters shown in Figure B.10f are variable in the following ranges :

- IPI : 214 µs <= IPI <= T/2 ;
- T : 10 ms <= T <= 100 ms (ie. repetition frequency from 10 Hz to 100 Hz)
- t_- : 0 µs <= t_- <= 800 µs ;
- t_+ : 0 µs <= t_+ <= 4000 µs ;
- A_+ : 0 mA <= A_+ <= 100 mA (into 1.5 KOhms load)
- A_- : 0 mA <= A_- <= 100 mA (into 1.5 KOhms load)

OPERATION OF TIMER 2

As mentioned earlier, timer 2 operates in the dual 8 bit continuous mode to produce a square-wave with variable duty cycle. This continuous operation mode is selected by setting the bits 3 and 5 of the control register 2 to zero (CR23 and CR25 = 0). As timer 2 should not be reinitialized when data is written

to the latches CR22 and CR24 are set to logic one. See table B.2 and B.3 for programming details. t_{on} of this square wave determines the interpulse interval IPI, while T determines the repetition frequency of the final FNS waveform (see Figure B.10). Figure B.10a shows the relationship between the value of the least significant byte (L) and the most significant byte (M) and the times t_{on} and T of the square wave generated by timer 2.

On-time of square wave $t_{on} = L * t_2$; (I)

Period of square wave $T = 1/f = (L+1) * (M+1) * t_2$; (II)

Note, that the on-time defines the time interval between the negative and positive pulse of the final FNS waveform (ie. IPI). The period T of the square wave generated by timer 2 defines the period of the FNS waveform. For FNS applications this period lies typically in the range of 10 ms to 100 ms (ie. from 100 Hz to 10 Hz repetition frequency, see chapter 2). In equation (I) and (II) 'L' and 'M' are both Byte variables which can range from 0 to 255. The value of the cycle time T_2 has to be selected in such a way that the specifications are met,

ie.: maximum IPI (ie. t_{on}) $> 4000\ \mu s$ (A)

period T : $10\ ms < T < 100\ ms$ (B)

Unfortunately T is a function of IPI ! Substitute for L from (I) into (II) :

$$T = (1 + IPI/t_2) * (M + 1) * t_2 =$$
$$= (IPI + t_2) * (M + 1) ;$$

T must be variable from 10 ms to 100 ms for all values of IPI. Even in the worst case, when IPI = t_2, T will have to extend over the maximum period of 100 ms, defined by M = 255.

Hence : $T_{max} = 2 * t_2 * 256 > 100$ ms ;

Hence : $t_2 > 195$ µs ;

From (A) and (I) : $t_2 > 4000$ µs / 255 ;

ie.: $t_2 > 16$ µs ;

Therefore an external clock frequency of 1/195µs = 5128 Hz is needed for timer 2. This signal can be obtained by dividing the internal clock frequency of the SPERRY personal computer (4.77 MHz) by 1024, giving a frequency of 4658 Hz, or a cycle time t_2 of 214.6750 us. Using this clock frequency interval IPI and the frequency of the square wave generated by timer 2 can be varied within the following limits :

IPI : 214.7 µs ---> 5000 µs ;

frequency (=1/T) : 9 Hz ---> 100 Hz ;

From equation II it can be seen, that the frequency of the square wave produced is dependent on both, the value for L and M. Thus the increment with which the frequency can be adjusted is dependent on the value of L. For L = 1 the change in frequency by changing M is very small. For increasing L however the resulting step change of the frequency gets greater.

For a given t_{on} (and thus L) the frequency f of timer 2 can be changed by varying the value M. If the desired frequency f lies half way between two discrete possible values of f a maximum error results. The percentile error can be expressed by the following relationship :

$$f_{error} = 1/2 * 100\% * (f(M) - f(M+1)) / f(M) ; \quad (III)$$

where f(M) is the pulse frequency for a given L and M, and f(M+1)

is the pulse frequency for the same value of L but M increased by one. Thus the difference f(M) - f(M+1) is the step size with which the frequency can be changed by varying M and keeping L constant. From equation (II) and (I) f(M) can be expressed as a function of the interpulse interval IPI, M and t_2 :

$$f(M) = 1 / (IPI/t_2 + 1) * (M + 1) * t_2 \quad ; \qquad (IV)$$

this equation can be solved for M :

$$M = (1 / (IPI/t_2 + 1) * f * t_2) - 1 \quad ; \qquad (V)$$

Equation (III) can be rewritten and simplified with equation (IV) and (V) as :

$$f_{error}(IPI,f,t_2) = 1/2*100\% * (f*(IPI+t_2))/(1+f*(IPI+t_2)); \quad (VI)$$

Figure B.12 gives a graphic representation of this error function.

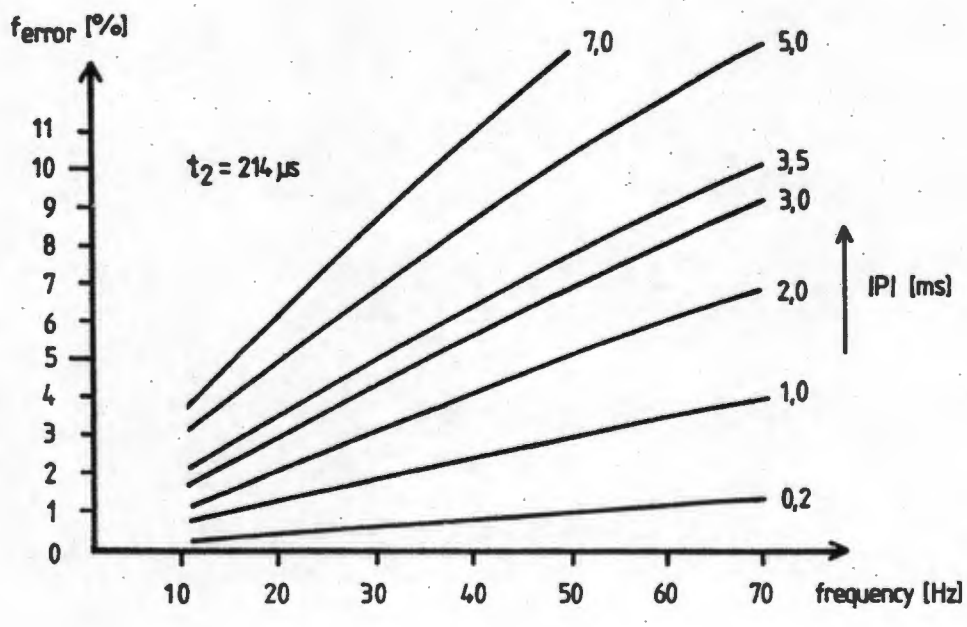


Figure B.12 Maximum frequency error as a function of IPI and f

The x-axis shows the desired pulse repetition-frequency f, the y-axis the worst-case error for IPI ranging from 0.2 ms to 7 ms. It can easily be seen that this error rises with f and IPI. However this error is limited to +- 10% for f ranging from 10 to 70 Hz and IPI ranging from 0.2 to 3.5 ms.

Given the relationships (I) and (II) the low-byte (L) and high-byte (M) which have to be loaded into timer 2 to select t_{on} and T can be calculated as follows:

$$L = IPI / t_2 ; \qquad \text{(VII)}$$

$$M = (1 / (1/T * (L+1) * t_2)) - 1 ; \qquad \text{(VIII)}$$

where t_2 = clockperiod of timer 2 (ie.214.6750 µs) and 1 / T is equal to the repetition-frequency f of the pulses generated by timers 1 and 3.

OPERATION OF TIMER 1 AND 3

Timers 1 and 3 are programmed as single shots in the 16 bit mode. This mode is identical to the continuous mode with three exceptions. The first of these is obvious from the name - the output returns to a low level after the first time out and remains low until another counter initialization cycle occurs (ie. via a gate input goes low or data is written to the latches). Secondly the counter function is independent of the level at the gate input once the gate input has gone low and this is interpreted as reinitialization by the timer. Thirdly the special condition of N = 0 (N is the 16 bit data-word in the timer latch) which leads to a disabled timer output. Figure B.13 shows the signal generation in single shot operation mode.

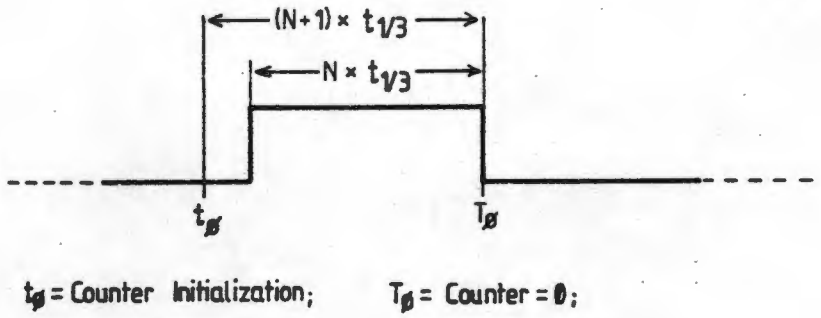


Figure B.13 Single shot operating mode

As timer 1 and 3 are triggered by the output of timer 2 (see Figure B.10) counter initialization takes place after a negative transition at the gate input (or a reset high). Both timers are controlled via an external clock source with a period $t_{1/3}$ of 6.7086 us (4.77 MHz / 32). Thus the minimum pulsewidth which is also equal to the minimum increment of timer 1 and 3 is 6.7 µs. The data-word N necessary for a desired pulsewidth t_- or t_+ can be calculated as follows :

$$N = t_- / 6.7086\ \mu s \quad ; \quad (IX)$$

$$\text{or} \quad N = t_+ / 6.7086\ \mu s \quad ;$$

The maximum pulsewidth which can be generated is 6.7 us * 65535 = 439.6 ms; For FNS waveforms the pulsewidth will range from 0 to 800 us for timer 3 and 0 to 4 ms for timer 1. (NOTE : The current stimulation software uses only the low byte of the 16-bit word N for definition of the pulse widths ! Thus with the current software only negative and positive pulse widths up to 1710 µs can be produced. See procedure 'settimers' in appendix H)

The counter initialization (ie. triggering) of the single shots can also be controlled via a write to latches command (W).

In this operation mode it is possible to generate FNS pulses and pulse trains under the complete control of software timing loops (see procedure impedance_check in Appendix H).

AMPLITUDE CONTROL

To ensure maximum flexibility of the FES-controller the amplitudes of the positive and the negative pulse are independently variable. Figure B.14 shows the principle of the amplitude control.

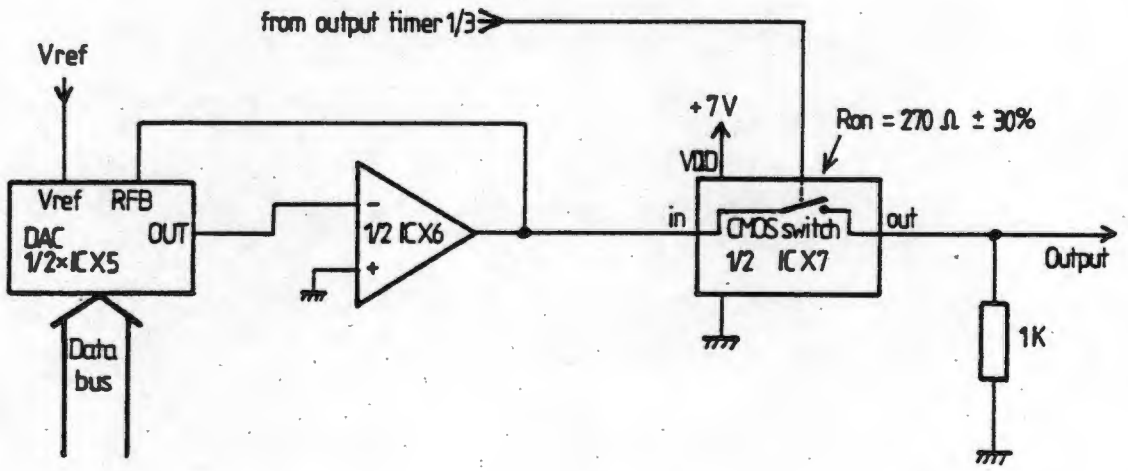


Figure B.14 Amplitude control of one stimulation channel

For each stimulation channel two 8 bit digital to analog converters (IC X5, where x is the number of the channel) contained in a single 20 pin package, are used to generate two independently controllable voltage sources. Both DAC's work in the unipolar binary operating mode (see Figure B.14, where one DAC is shown). These voltage sources deliver 0 to + Vref (Volts) to the inputs of a CMOS analog switch (ICX7) which is controlled by the output of timer 1 and 3 respectively. To reduce crosstalk

between the analog switches and improve the shape of the output pulse the outputs of the CMOS switches are loaded with 1 KOhm resistors. The on resistance of the CD 4066 analog switch is typically 270 Ohms + 30% , thus attenuating the voltage at the output of the analog switch by the ratio 1/(1+0.27) = 0.78 . To reach + 5 Volts into a 1000 Ohms load the output of the D/A converter must deliver 5V/0.78 = 6.35 V. The voltage reference (Vref) of the D/A converter is therefore about - 6.35 Volts. The negative reference voltage is regulated with a variable voltage regulator (see Figure B.3 IC 19 adjustable with VR1). To avoid latching up of the input of tha analog switch the power supply of IC X7 has to be at least 7 Volts (see regulator IC 20 in Figure B.3). To accommodate the + 30% variation of the on-resistance of the CMOS switches, the positive and negative pulse amplitude can be adjusted independently in the isolation amplifier. Note that for each channel two positive going pulses with variable amplitude (0 to + 5 V) and timing are generated. The output of timer 3 controls the negative going pulse and timer 1 the positive going charge balancing pulse.

The output of timer 3 will be inverted and added to the output of timer 1 in the isolation amplifier in order to decrease the hardware requirements on the plug in board for the PC.

Two additional D/A channels are provided for the control of a motor for the support of the patient in the initial stages of a strengthening program (see figure B.15)

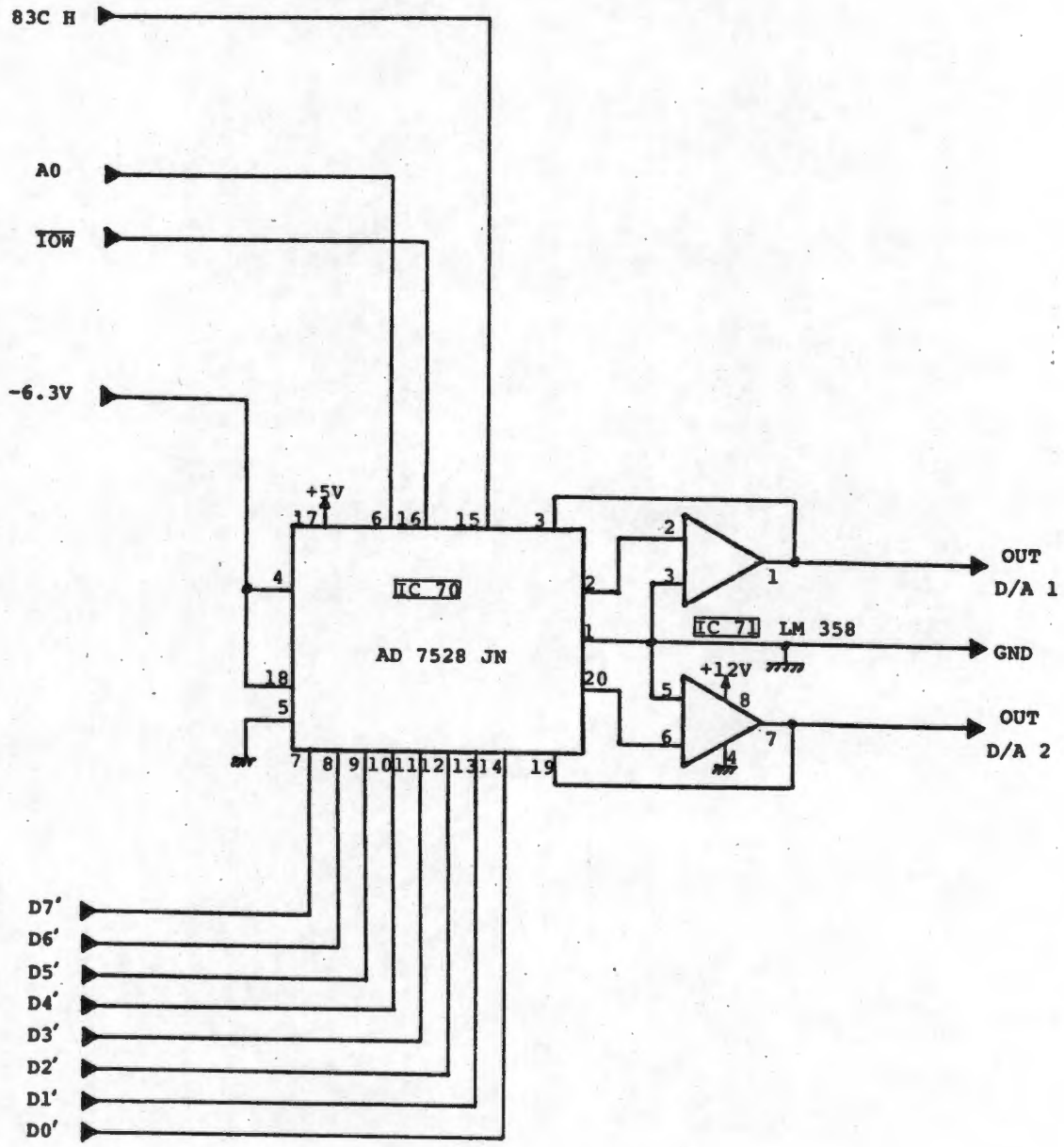


Figure B.15 Circuit diagram stimulation waveform generation board (sheet 3 of 3)

ADDRESS SPACE OF THE STIMULATION GENERATOR BOARD

The base address of the stimulation board can be selected with jumpers JMP 1 and JMP 2 (see Figure B.3 and B.16). (JMP 1 represents a weight of + 80 Hex and JMP 2 a weight of + 40 Hex to the address 800 Hex. Thus four baseaddresses can be selected : 800 Hex, 840 Hex, 880 Hex, 8C0 Hex. The 6 timers and 14 D/A converters are situated in the address-space above the base-address (B) of the waveform generation board as follows:

A = B + 00	timerchip 1	B + 34	D/A channel 3 +
A = B + 08	" 2	B + 35	" 3 -
A = B + 10	" 3	B + 36	" 4 +
A = B + 18	" 4	B + 37	" 4 -
A = B + 20	" 5	B + 38	" 5 +
A = B + 28	" 6	B + 39	" 5 -
B + 30	D/A channel 1 +	B + 3A	" 6 +
B + 31	" 1 -	B + 3B	" 6 -
B + 32	" 2 +	B + 3C	D/A motor 1
B + 33	" 2 -	B + 3D	D/A motor 2

Registers of the 68B40 timer chip

A + 0	CR3 (CR20=0)/CR1		
A + 1	CR2		
A + 2	msb	timer 1	positive pulse (+)
A + 3	lsb	timer 1	
A + 4	msb	timer 2	pulse frequency
A + 5	lsb	timer 2	
A + 6	msb	timer 3	negative pulse (-)
A + 7	lsb	timer 3	

For example, the msb of timer 3 of the timer chip number 4 has

the I/O port address : B + 18 + 6 , ie. 8C0 + 18 + 6 = 8DE (Hex) if the base address B is 8C0 (Hex).

PROGRAMMING OF THE 68B40 TIMER CONTROL WORDS

As described earlier three write only registers in the MC 68B40 are used to modify the timer operation. Table B.2 and Table B.3 sum up the control register programming.

The following programming sequence is recommended :

1. select control register 1 (CR1) by writing a one into the LSB of control register 2 (CR2).
   ie. in PASCAL : Port [A+1] := 1;
2. select preset mode for all timers by writing a one into LSB of CR1.
   ie. in PASCAL : Port [A] : = 1;
3. preset timer 1, 2, 3 with initial values.
4. select CR3 via LSB CR2 = 0
   ie. in PASCAL : Port [A+1] := 0;
5. output control word CR3 for timer 3
   ie. in PASCAL : Port [A] := CR3;
6. output control word CR2 for timer 2 and select CR1 with LSB of CR2 = 1.
   ie. in PASCAL : Port [A+1] := CR2 + 1;
7. output control word CR1 for timer 1 and set all timers in operation (LSB CR1 = 0).
   ie. in PASCAL : Port [A] := CR1;

For normal operation the following control words must be selected:

Timer 1 and 3 ..... B0 (Hex) ie. single shot mode, external clock, 16 bit mode, gate or reset causes counter initialization, all timers operate.

ie. CR1 = B0 ;
CR3 = B0 ;

Timer 2 ..... 95 (Hex) ie. continuous dual 8 bit mode, gate or reset causes counter reinitialization, CR1 may be written into.

ie. CR2 = 95 ;

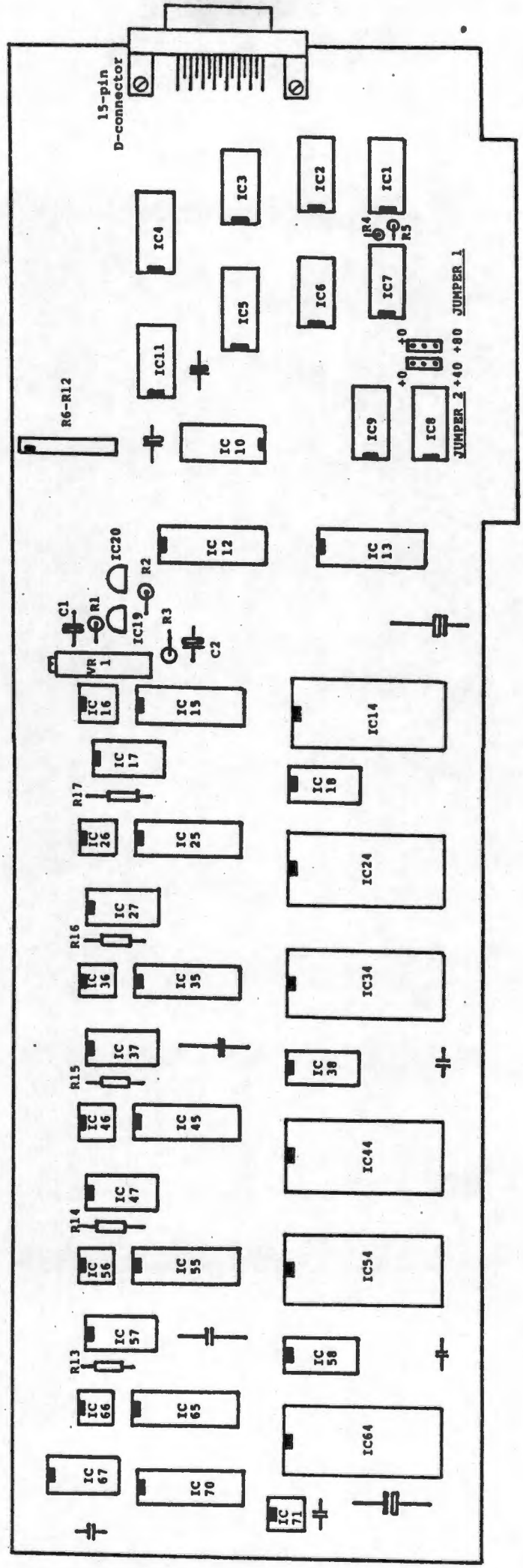
Timer 1 and 3 can also be triggered via software to produce pulse trains etc.. For this mode the output of timer 2 is disabled and timer 1 and 3 counter-initialization takes place when data is written to the respective timer.

CR1, CR3 = A0 (Hex); CR2 = 15 (Hex);

MECHANICAL CONSTRUCTION AND CONNECTIONS

The waveform generation board is assembled on a double sided printed circuit board which fits into a full length expansion slot of the IBM Personal Computer. The outputs are accessible at a female 15-pin D-connector at the rear mounting bracket. Figure B.16 shows the component layout of the stimulation waveform generation board

Figure B.17 Component layout stimulation waveform generation board (see next page)


15-pin
D-connector
IC1
IC2
IC3
IC4
IC5
IC6
IC7
IC8
IC9
IC10
IC11
IC12
IC13
IC14
IC15
IC16
IC17
IC18
IC19
IC20
IC24
IC25
IC26
IC27
IC34
IC35
IC36
IC37
IC38
IC44
IC45
IC46
IC47
IC54
IC55
IC56
IC57
IC58
IC64
IC65
IC66
IC67
IC70
IC71
R1
R2
R3
R4
R5
R6-R12
R13
R14
R15
R16
R17
C1
C2
VR 1
JUMPER 1
+0
JUMPER 2
+0
+40
+80

Pin assignment 15 pin D-connector

pin number	assignment
1	out D/A 1
2	channel 6 +
3	channel 5 +
4	channel 4 -
5	channel 3 -
6	channel 2 -
7	channel 1 +
8	GROUND
9	out D/A 2
10	channel 6 -
11	channel 5 -
12	channel 3 +
13	channel 2 +
14	channel 4 +
15	channel 1 -

timer 3 of each timerchip generates the - output and timer 1 of each timerchip generates the + output.

PARTS LIST FES-STIMULATOR BOARD

I C 's :

IC 1	74LS 05	6 inverters open collector
IC 2/9	74LS 30	8 input NAND gate
IC 3	74LS 74	dual D-flip/flop
IC 4	74LS 174	8 D-flip/flop

IC 5	CD 4040	12 stage counter
IC 6	74LS 00	4 NAND gates
IC 7/18/38/58	74LS 04	6 inverters
IC 8/10	74LS 138	1 out of 8 decoder
IC 11	74LS 02	4 NOR gates
IC 12/13	74LS 244	8 non-inverting busbuffers
IC 14/24/34/44/54/64	MC 68B40	Motorola timer
IC 15/25/35/45/55/65/70	AD 7528	dual 8-bits D/A
IC 16/26/36/46/56/66/71	LF 358	dual OP-amplifier
IC 17/27/37/47/57/67	CD 4066	quad analog switch
IC 19	L 337	adjustable neg. regulator
IC 20	L 317	adjustable pos. regulator

S O C K E T S

7 x 8 pin

16 x 14 pin

4 x 16 pin

9 x 20 pin

6 x 28 pin

APPENDIX C : ISOLATION AMPLIFIER

The pulses coming from the waveform generation board are combined to form a biphasic or monophasic waveform which is then amplified in the isolation unit. This unit contains six separate isolation amplifier channels each assembled on one single sided Euro card and mounted in a 19 inch rack. The isolation unit receives pulses from the waveform generation board and returns analog signals to the A/D converter board. The isolation amplifier has to fulfill the following functions :

- combine the two monophasic pulses coming from the waveform generation board into a biphasic (or monophasic) stimulation waveform;
- act as a voltage controllable current source, which is able to drive at least 100 mA into a load of 1.5 KOhm;
- amplify with a low level of distortion;
- achieve total isolation between all stimulation channels and ground, thus avoiding interference between neighbouring channels and ensuring the patient's safety;
- limit the output voltage and current to safe values;
- measure the electrode impedance before or during stimulation;

As already discussed in chapter 4.2.3. a push pull power amplifier and a pulse transformer are used for isolation and voltage amplification. Figure C.1 shows the block diagram of one channel of the isolation amplifier.

The design of the pulse transformer plays a major part in the performance of the isolation amplifier. For the analysis of the influences of various transformer parameters it is necessary to obtain a model or equivalent circuit of the pulse transformer.

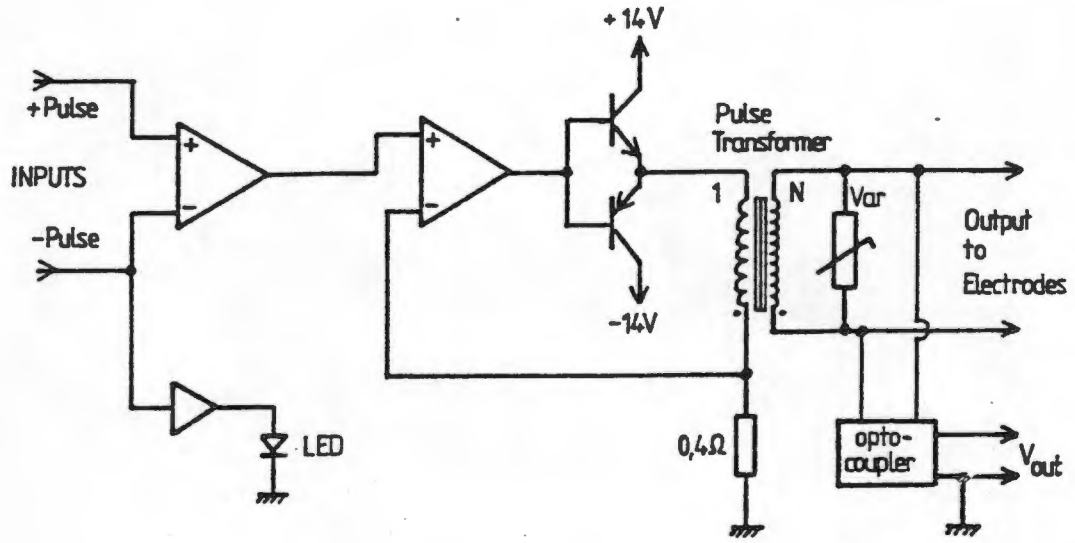


Figure C.1 Isolation amplifier (one channel)

EQUIVALENT CIRCUIT OF A PULSE TRANSFORMER

Figure C.2 shows a equivalent circuit diagram of a pulse transformer (Millman and Taub, 1965). The elements of the circuit and methods to measure them are described. The influences of the circuit elements on pulse wave shape are discussed.

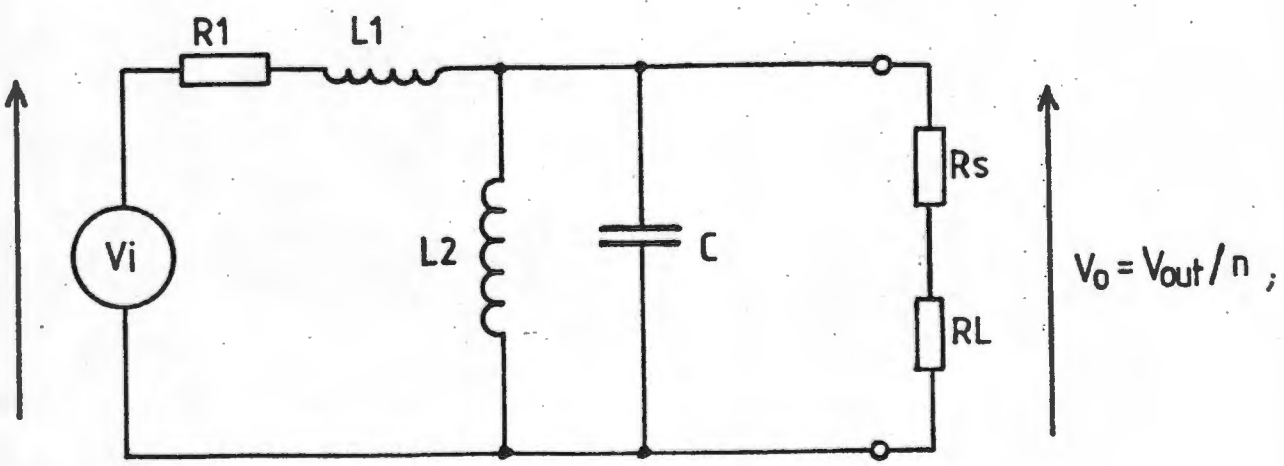


Figure C.2 equivalent circuit pulse transformer

- Np is the number of primary turns.
- Ns is the number of secondary turns.
- n is the turns ratio Ns/Np.
- Vi is a voltage source connected to the primary side of the transformer.
- Vo is the output voltage of the equivalent circuit.
- Vout is the voltage at the secondary of the transformer.
- R1 is the resistance of the primary winding.
- L1 is the leakage inductance.
- L2 is the magnetizing inductance of the primary winding.
- C is the leakage capacitance.
- R2 represents the combination of the load resistance RL and the secondary winding resistance Rs transformed to the primary side of the transformer (ie $R2 = (RL+Rs)/n^2$ (1))

Other important constants describing a given transformer are:

- A1 factor, which describes the permeability of the core material in nano Henry (nH) per winding squared.
- A_e , the cross-sectional area of the core.
- A_{min} , the minimum effective cross-sectional core area.

The primary inductance L2 can be calculated from the A1 factor and the number of primary turns Np as follows :

$$L2 = A1 * Np^2 \quad [nH] ; \qquad (2)$$

The leakage inductance L1 is due to the leakage flux, that is the flux which links one but not both windings. Leakage inductance is proportional to Np^2, inversely proportional to the A1 factor and proportional to the distance between the primary and secondary.

small number of primary turns, small spacing between primary and secondary and a core with a high Al factor are essential. The leakage inductance L1 may be measured with a Q meter or an impedance bridge provided that the transformer secondary is shorted. Of course the resistance R1, which is in series with L1 must be taken into account. Another method of measuring the leakage inductance L1 of a given transformer is to short the secondary, shunt the primary with a capacitance C_1 and measure the resonant frequency f_1 of the combination L1,C_1 (see Figure C.3).

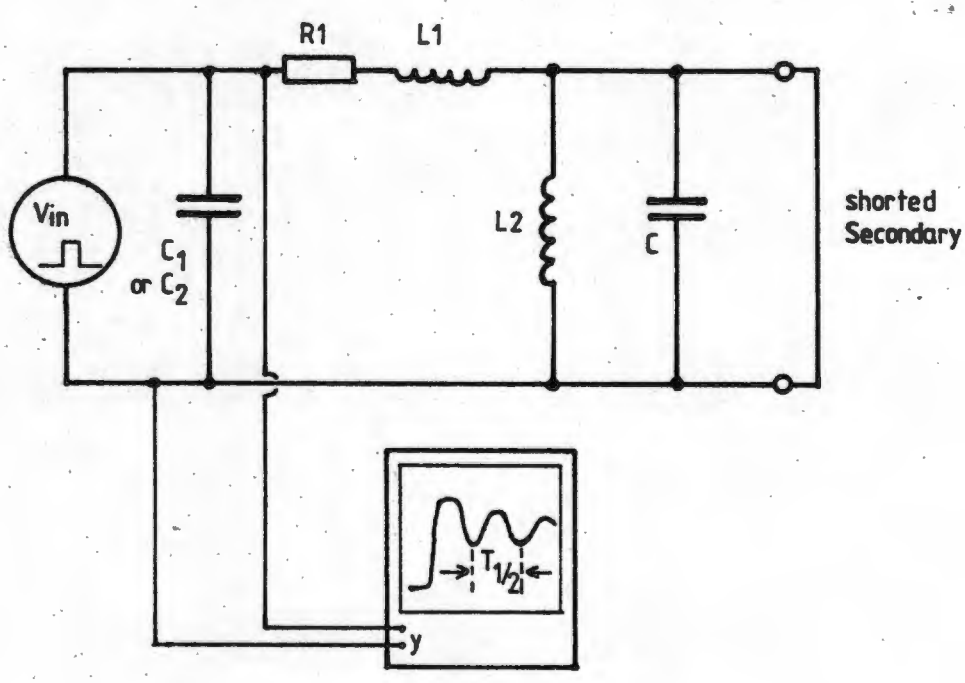


Figure C.3 Measurement of the leakage inductance L1

In order to eliminate the effect of the transformer and other unknown external capacitances which are in shunt with C_1, the above measurement is repeated with a second capacitor C_2. A pulse generator is connected to the primary and the waveform is viewed on an oscilloscope. The resonance frequencies can be calculated from the period of the 'ringing' wave of the step response (ie. $f_{1/2} = 1 / T_{1/2}$).

From the two resonant frequencies f_1 and f_2 the leakage inductance L1 can be calculated as follows (Millman and Taub, 1965) :

$$L1 = (f_1^2 - f_2^2) / [(2*Pi*f_1*f_2)^2 * (C_2 - C_1)] ; \quad (3)$$

The leakage capacitance C is proportional to n^2 and inversely proportional to the distance between the primary and secondary windings. Thus the leakage capacitance C decreases with distance between primary and secondary whereas the leakage inductance L1 increases. For a given transformer the leakage capacitance can be measured as shown in Figure C.4 by measuring the frequency at which L2 and C are in resonance;

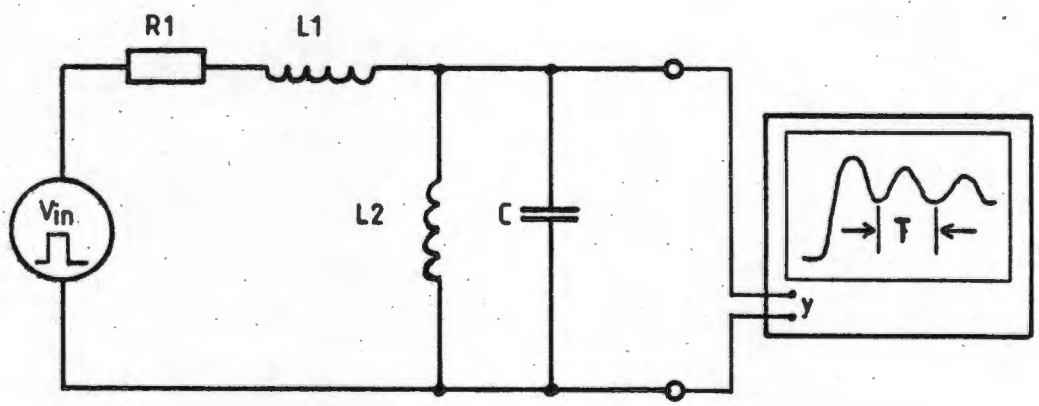


Figure C.4 Measurement of the leakage capacitance C

A pulse generator is connected to the primary. The output voltage on the open circuited secondary is viewed on an oscilloscope. The resonance frequency of the combination L2 and C can be determined from the period T of the 'ringing' waveform of the step response (ie. f = 1/T). Provided that L2 >> L1 the leakage capacitance C

can then be calculated as :

$$C = 1 / (4 * Pi^2 * L2 * f^2) \; ; \quad (4)$$

where L2 is the inductance of the primary.

PULSE RESPONSE OF THE PULSE TRANSFORMER

From the equivalent circuit (Figure C.2), the rise-time response of a pulse transmitted through the transformer can be calculated. Figure C.5 gives an example of distortion due to the pulse transformer to a typical pulse used in FNS (overdamped situation).

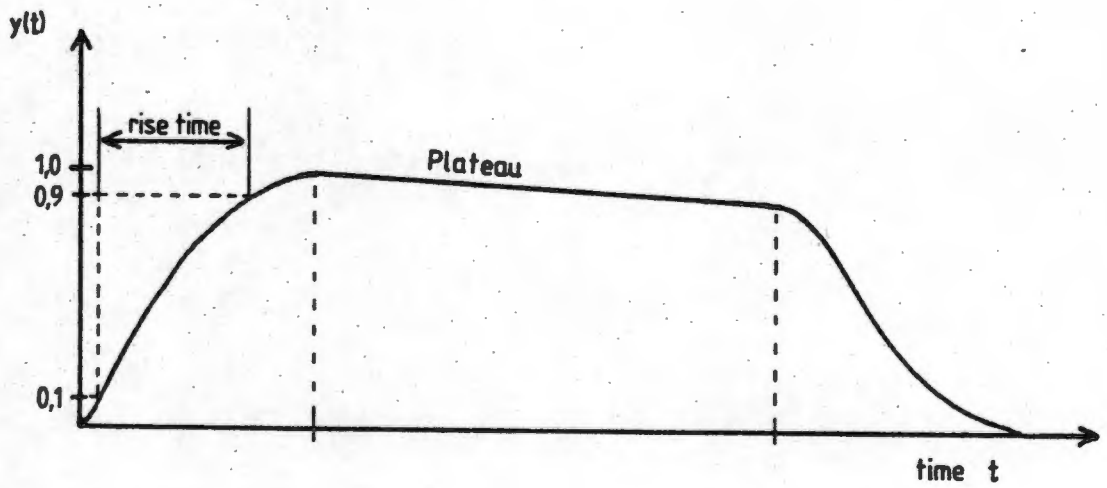


Figure C.5 Pulse distortion due to the pulse transformer

RISE-TIME RESPONSE

For calculation of the rise time response the following variables are defined (see Figure C.2 for components of the equivalent transformer circuit) :

$$R2 = (RL+Rs)/n^2 \qquad (5)$$

$$a = R2 / (R1 + R2); \qquad (6)$$

$$T = 2 * Pi * (L1*C*a)^{1/2}; \qquad (7)$$

$$k = (R1/L1 + 1/(R2*C)) * T/4*Pi ; \qquad (8)$$

$$x = t/T ; \qquad (9)$$

$$y = Vo/(n*a*Vout) \qquad (10)$$

where k is the damping factor and t is the time variable;

To achieve a voltage of +- 150 Volt at the transformer output when the maximum voltage at the input is limited by the power supply voltage of the power amplifier (see Figure C.1) to approximately +- 12 Volt a turns ratio n of at least 12.5 is required. The electrode impedance RL can vary between 300 and 1500 ohms. A practical value for the primary winding resistance R1 is 0.3 ohms and for the secondary winding resistance Rs is 60 ohms. Values for the leakage inductance L1 ranged from 1.9 µH to 160 µH. Values for the leakage capacitance C ranged from 130 pF to 320 pF (see Tables C.1 and C.3). In practice the impedance of the electrodes (RL) on the output of the pulse transformer will damp the circuit sufficiently so k will become greater than 1.

From equations (5) to (8) it can be concluded that the damping factor k is smallest for the greatest value of RL , and the smallest values for n, R1, L1, and C. Choosing the relevant values from the above-mentioned ranges, k can be calculated with equations (5) to (8) to be

$$k = 5.967 ;$$

where : n=12.5, R1=0.3 ohms, RL=1500 ohms, L1=1.9 µH, C=130 pF;

Thus k is well above 1 and the overdamped solution for the rise time response given by Millman and Taub (1965) is applicable :

$$y = 1 - (4k^2/(4k^2-1)) * e^{-Pi*x/k} + (1/(4k^2-1)) * e^{-4*Pi*k*x} ;$$

and if $4k^2 >> 1$

$$y = 1 - e^{Pi*x/k} ; \qquad (11)$$ (Millman,Taub, 1965)

From equation (11) it is possible to determine the rise time from 10 % to 90 % of the maximum value of y = f(x). With use of equation (9) equation (11) is solved for t :

$$t = - 2 * k * (L1 * C * a)^{1/2} * \ln (1-y) ; \qquad (12)$$

With equation (12) t can be determined when y is equal to 0.1 and 0.9 . The difference between these two times is the rise time t_{rise}. Hence :

$$t_{rise} = (\ln (0.9) - \ln (0.1)) * 2 * k * (L1 * C * a)^{1/2} =$$
$$= 4.394 * k * (L1 * C * a)^{1/2} ; \qquad (13)$$

It can be seen that in order for the output to rise rapidly, the leakage inductance L1 and the leakage capacitance C must be kept small. The rise time may also be reduced by reducing 'a ' , but a small 'a ' will result in a highly attenuated output voltage. As decided earlier, if the interwinding and interlayer distances are kept small, the leakage inductance will be small but the leakage capacitance will increase, thus decreasing L1 at the expense of C. The best combination of values for C and L1 must therefore be determined experimentally.

THE PLATEAU OF THE PULSE

The response during the plateau of the pulse (see figure C.5) is obtained from the equivalent circuit (see figure C.2) by neglecting the effect of the leakage inductance L1 and leakage capacitance C (At low frequencies wL2 is small compared with 1/wC, and wL1 is small compared with R1) (Millman and Taub, 1965). Thus the output y(t) is given by :

$$y = e^{-(R * t) / L2} ; \qquad \text{(Millman and Taub, 1965)}$$

where $R = R1*R2 / (R1+R2)$;

For Values of $R * t / L2$ much less than unity, the output can be approximated by :

$$y = 1 - R * t / L2 \quad ; \quad (14) \qquad \text{(Millman and Taub, 1965)}$$

Hence the plateau of the output pulse will slope downward and the percent slope P is given by :

$$P = 100\ \% * R * t_p / L2 \quad ; \qquad (15)$$

where t_p is the pulse width;

It can be seen, that for increasing L2 the slope decreases.

INFLUENCE OF THE CROSS-SECTIONAL AREA OF THE CORE

We have assumed that the inductance L2 is a constant. This assumption is valid as long as the core material does not begin to saturate. For a ferrite core the permeability is fairly constant for flux densities B up to B_{max} which is of the order of 0.15 to 0.6 Tesla. Saturation occurs if B exceeds the above value

B_{max}. The flux B can be calculated as :

$$B = (R2/(R1+R2) * Vout/n * t_p) / (Np * A_{min}) ; \quad (16)$$

(after Millman,Taub 1965)

Thus if all other variables are given the core area A_{min} can be calculated :

$$A_{min} = (R2/(R1+R2) * Vout/n * t_p) / (Np * B_{max}); \quad (17)$$

where B_{max} is the maximum flux density for a given core material, and t_p is the maximum pulse width which has to be transmitted without saturation.

SELECTION OF A SUITABLE PULSE TRANSFORMER

With the above equations it is possible to select a suitable transformer core and obtain guidelines regarding the winding and construction of the transformer. The iterative procedure of selecting and testing a transformer core is described below.

The pulses which have to be transformed have the following characteristics :

pulse width : 0 to 800 µs ;
pulse amplitude : 150 V ; ie. 100 mA into a 1500 Ohms load;

Because of its high permeability and high resistivity, ferrite core material is chosen. The high resistivity means that the skin effect due to eddy currents is very small, thus allowing fast pulse rise times. The maximum flux density of the SIFERRIT N27 core material of Siemens is 0.51 Tesla. The available Al permeability factors depend on the core size and range from 1500

to 3500 nH/turns^2.

The minimum crossectional core area is calculated with equation (17) :

$$A_{min} = (R2/(R1+R2) * Vout/n * t_p) / (Np * B_{max});$$

To simplify the calculation, R1 is assumed to be much smaller than R2 .

Thus $A_{min} = (Vout/n * t_p) / (Np * B_{max});$ (18)

To achieve a voltage swing of +- 150 volts on the secondary with a power current amplifier configuration (see figure C.1) which is supplied by +- 12 Volt DC, a step up ratio of at least 12.5 (150 V / 12 V) is neccessary. However, losses in power transistors due to collector emitter saturation voltage and particularly the current sensing resistor Rs, make a step up ratio of at least 30 necessary. From equation (17) it can be seen that A_{min} decreases with rising turns ratio n and number of turns in the primary.

The value of the primary magnetizing inductance L2 can be determined with equation (15) under the assumption that the plateau of the transformed pulse may fall a maximum of 5 % during an 800 us long pulse. Hence :

$$L2 = R * 800\ \mu s / 0.05 = 0.3\ Ohms * 800\ \mu s / 0.05 = 4.8\ mH ;$$

where R = R1*R2 / (R1+R2) with the primary winding resistance R1 = 0.3 Ohms and R2 = 1500 Ohms/n^2 (see equation (1)).

Assuming an A1 factor of 2000 the number of primary turns N_p can be calculated with equation (2) as :

$$N_p = (L2 / A1)^{1/2} = 49\ turns ;$$

Thus from equation (18) the minimum cross sectional core area was calculated as

$$A_{min} = 157 \text{ mm}^2 ;$$

with Vout = 150 V;
t_p = 800 µs;
n = 30;
Np = 50;
B_{max} = 0.51 Tesla;

Thus the Siemens SIFFERIT core EC 52 which has an A_{min} of 134 mm^2 was initially choosen.

The performance of the transformer was tested with the constant current amplifier. Figure C.6 shows the measurement of the maximum pulse width due to core saturation. The primary winding consists of one layer. Insulation between primary and secondary is a 0.3 mm thick layer of PVC tape. The primary (50 turns) is wound in one layer 0.66 mm diameter enamelled wire. The secondary consists of 1500 turns 0.2 mm diameter enamelled wire wound on top of the primary.

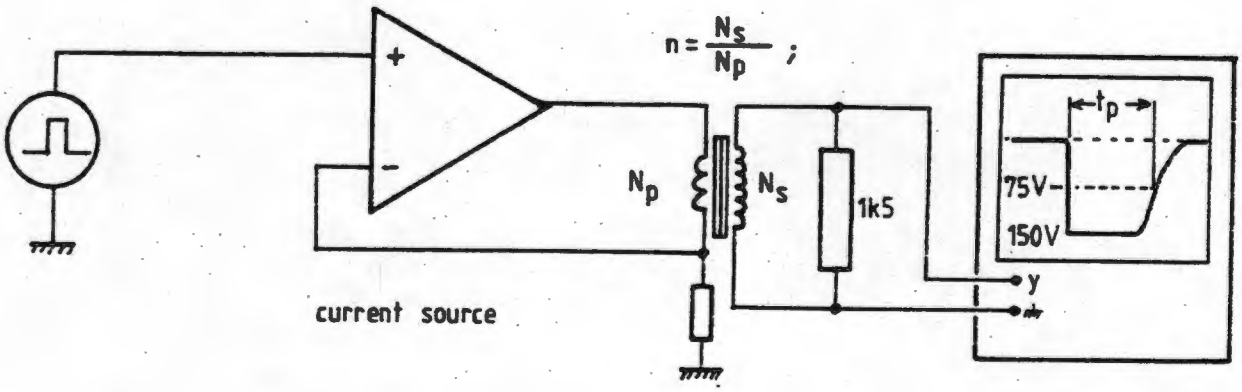


Figure C.6 Test circuit for measurement of maximum pulse width t_p

The maximum pulse width of a 150 V pulse into a load of 1500 Ohms before saturation was determined to be 500 µs. This is due to

the fact that the EC-52 core has an A_{min} of 134 mm^2 instead of the calculated 157 mm^2 which is needed to reproduce an 800 us long pulse without core saturation under the same test conditions. In addition the values of the leakage inductance and capacitance were determined. These results are summarized in Table C.1 :

Table C.1 Experimental Data SIFFERITE EC-52 core

AL	[nH/turn^2]	3400
primary turns Np		50
secondary turns Ns		1500
leakage L1	[µH]	2.52
magnatizing L2	[mH]	19
leakage capacit.	[pF]	142
A_{min}	[mm^2]	134
primary R1	[Ohms]	0.2
secondary Rs	[Ohms]	50
Turns ratio n		30
LOAD = 1.5 K; Vout = 150 V; max Pulsewidth t_p	[µs]	500
Vout = 50 V; max Pulsewidth t_p	[µs]	1060
Risetime	[µs]	24

The rise time is the time the pulse needs to rise from 10% to 90% of the output voltage swing.

On the basis of these experimental results and the fact that the necessary minimum core area can be decreased with a rising turns ratio it was decided to increase the turns ratio to 50 and to choose a core with a greater core area in order to be able to transform pulses longer than 500 µs. Figure C.7 shows the mechanical dimensions of the transformer core finally chosen. It is a Siemens SIFFERIT core ETD 44/22/15 which has a core area A_{min} of 172 mm^2 (Siemens ordering code B 66365-G-X127).

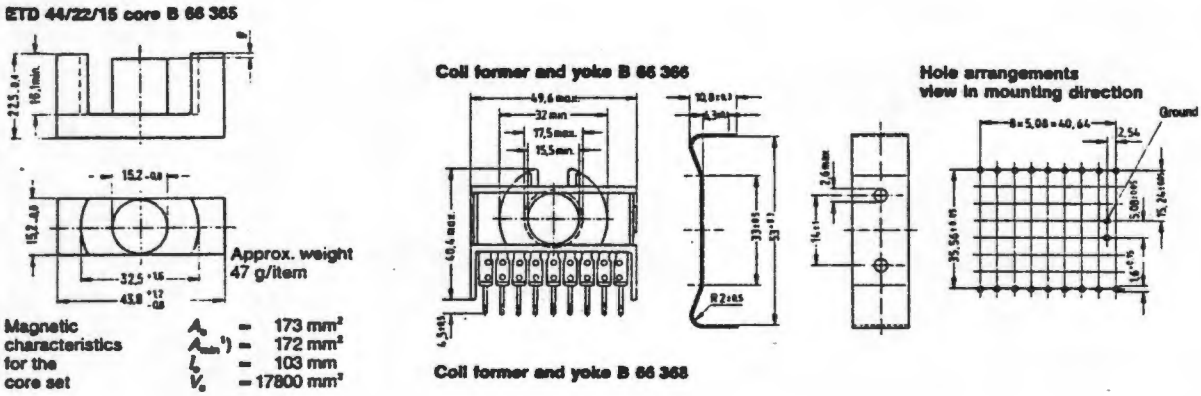


Figure C.7 Siemens SIFFERIT core ETD 44/22/15

Several combinations of primary turns and step-up ratios were tested in order to confirm the theoretical relationships. Table C.2 gives the results of these measurements.

Table C.2 Measurements ETD 44/22/15 core

	t_p max [µs] 150 V into 1500 ohms	Rise time (10% --> 90%)
Np = 40 (1) Ns = 1000 n = 25	480 µs	4 µs
Np = 80 (2) Ns = 2000 n = 25	760 µs	15 µs
Np = 40 (3) Ns = 2000 n = 50	820 µs	8 µs
Np = 100 (4) Ns = 2500 n = 25	850 µs	27 µs
Np = 40 (5) Ns = 2000 n = 50	800 µs	143 µs

(5) primary wound next to secondary

From these results it can clearly be seen that the combination of 40 primary turns and a step-up ratio of 50 is optimal regarding maximum pulse width and rise time of the pulse. The last transformer (number (5) in Table C.2) was wound to investigate the influence of placing the primary next to the secondary on the coil former. The rise time of transformer (5) is approximately 18 times greater than that of transformer (3). According to the theory mentioned earlier, the leakage inductance and thus the rise time of a transmitted pulse rises with the distance between the primary and secondary (see equation (13)). This was confirmed by measurement of the leakage inductances of transformer (3) as 7.8 µH and transformer (5) as 160 µH (see Figure C.3 for measurement procedure). Table C.3 shows the results of the measurements of transformers (3) and (5).

Table C.3 Characteristics of transformers (3) and (5)

	TRANSFORMER (3) (secondary wound on primary)	TRANSFORMER (5) (secondary wound next primary)
Np (turns)	40	40
Ns (turns)	2000	2000
R1 (Ohms)	0.2	0.2
Rs (Ohms)	60	60
L1 (µH)	7.8	160
C (pF)	130	134
a	0.76	0.76
T (µs)	0.174	0.801
k	170	761
t_{rise} (µs)	20	430
t_{rise} (measured)	8	134

The variables a, T, the damping factor k, and the rise time t_{rise} were calculated with equations (5) to (8) and (13) for an electrode resistance of 1500 Ohms (ie. RL in Figure C.2 is 1500 Ohms). The results of the actual measurements of the rise time

were about 3 times better than the calculated values.

Figure C.8 shows the final mechanical arrangement of the pulse transformer using a SIFFERIT ETD 44/22/15 core as shown in figure C.7 .

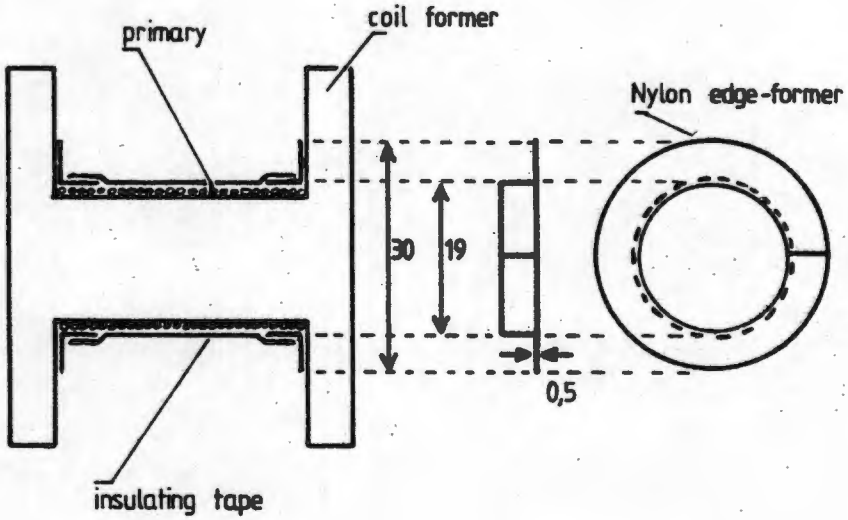


Figure C.8 Pulse transformer and insulation between primary and secondary

The primary consists of one layer with 38 turns 0.8 mm diameter enamelled wire (22 Standard wire gauge). Insulation of the primary from the secondary is ensured by two nylon formers and a 0.5 mm thick layer of insulation tape. The nylon formers prevent physical contact between the two windings on both ends of the winding space of the coil former. The secondary consists of 2000 turns of 0.2 mm diameter enamelled wire which are wound in the same direction as the primary on top of the insulation layer. The two halves of the EDT 44/22/15 core are inserted into the coil former and fixed with two stainless steel snap-in yokes (see Figure C.7).

THE VOLTAGE CONTROLLED CURRENT AMPLIFIER

Figure C.9 shows the principle of the voltage controlled amplifier including the pulse transformer.

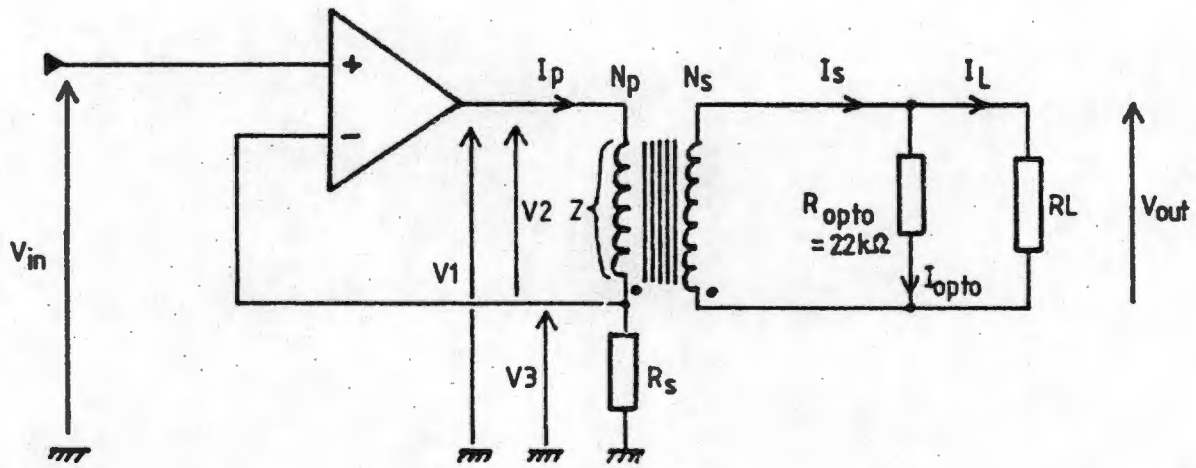


Figure C.9 Current amplifier

The amplifier includes a push-pull power output stage which drives the primary of the pulse transformer. The voltage drop V3 over the shunt resistor R_s is proportional to the current flowing through the primary. Assuming an ideal transformer, the primary current I_p is equal to the secondary current I_s multiplied by the turns ratio

$$I_p = n * I_s \ ;$$

where $n = N_s/N_p$;

The impedance Z is equal to the secondary load resistance transformed by $1/n^2$ to the primary, thus

$$Z = (R_L * R_{opto}/(R_L+R_{opto})) \ / \ n^2 \ ; \quad (20)$$

The voltage V2 is defined by the primary current I_p which flows through the impedance Z as

$$V2 = Z * I_p \; ; \qquad (21)$$

The voltage V3 is given by the primary current I_p which flows through the shunt resistor R_s as

$$V3 = R_s * I_p \; ; \qquad (22)$$

The voltage V1 can be described by the transfer function of a non-inverting operational amplifier as

$$V1 = Vin * (1 + Z/R_s) \; ; \qquad (23)$$

Assuming an ideal transformer the current I_s flowing through the primary is

$$I_p = n * I_s \; ; \qquad (24)$$

The current through the secondary I_s is distributed between R_L and R_{opto}. R_{opto} is the resistance of the opto-coupler stage which is provided for measurement of the output voltage. Its value is 22 KOhms. Thus the current I_L through the electrodes is

$$I_L = I_s * 22000 / (22000 + R_L) \; ; \qquad (25)$$

From Figure C.9 it is obvious that

$$V1 = V2 + V3 \; ; \qquad (26)$$

Substituting (21), (22), and (23) into (26) we have :

$$Vin * (1 + Z/R_s) = Z * I_p + R_s * I_p \; ; \qquad (27)$$

and substituting I_p from equation (24) and rearranging :

$$I_s = Vin / (n * Rs) \; ; \qquad (28)$$

Thus the current I_s through the secondary is independent of the secondary load resistance which is the combination of R_L and R_{opto}. Substitution for I_s from equation (25) gives :

$$I_L = (Vin * 22000) / (n * R_s * (R_L + 22000)) ; \quad (29)$$

Hence the current I_L flowing through the resistance R_L (ie. the stimulating current through the electrodes) is slightly dependent on the electrode resistance R_L. However this influence is negligible for realistic electrode impedances which range from 200 Ohms to 1500 Ohms. For example if Vin = 2 V, the current I_L ranges from 93.4 mA to 88.3 mA for the electrode impedance varying between 200 Ohms and 1500 Ohms, (ie. a variation of only 5.5 % .

CIRCUIT DESCRIPTION OF THE ISOLATION AMPLIFIER

Figure C.10 shows the complete circuit diagram of the isolation amplifier.

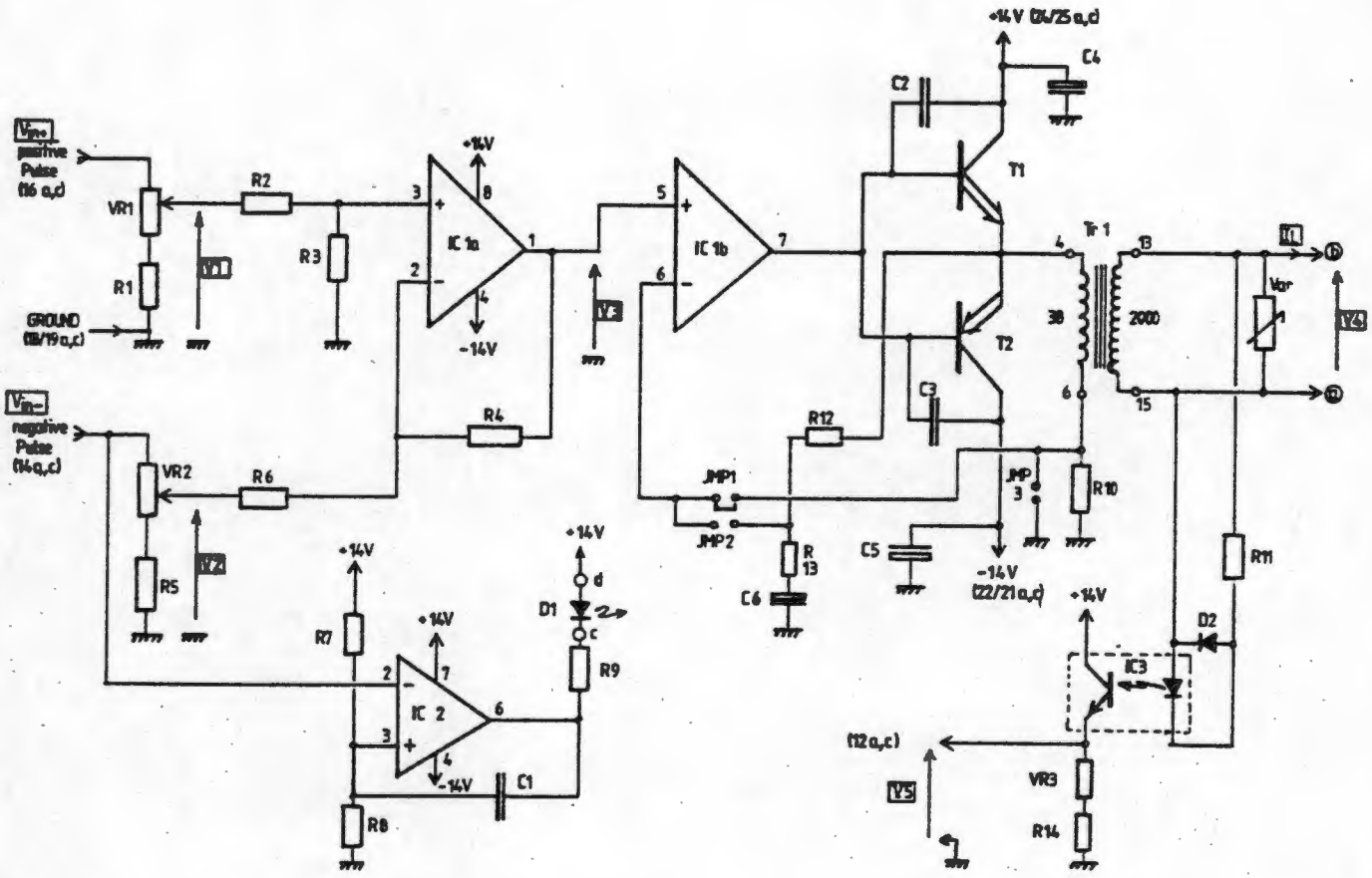


Figure C.10 Circuit diagram isolation amplifier

Figure C.11 shows the signal at various points in the isolation amplifier. The two positive going input voltages V_{in+} (b) and V_{in-} (a) are combined with the differential amplifier IC1a (Figure C.10) into a biphasic FNS voltage V3 (c). For R2 = R6 and R3 = R4, the transfer function of the differential amplifier IC1a is defined by :

$$V3 = (V1 - V2) * R4 / R6 ; \quad (30)$$

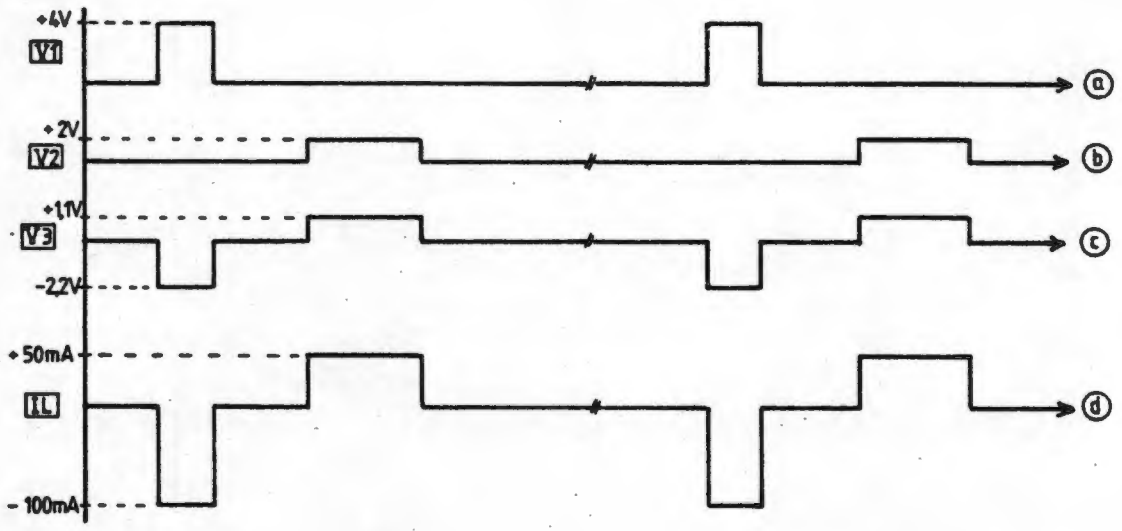


Figure C.11 Signals in the isolation amplifier

Hence the gain of the first stage is set by R4 and R6. The second part of the amplifier consists of operational amplifier IC1b driving a darlington push-pull output stage. The output current of the transformer secondary is given by equation (28) as :

Is = V3 / (n * R10) ;

The actual current through the electrodes with an impedance RL is given by equation (29) as:

IL = (V3 * 22000) / (n * R10 * (RL + 22000)) ; (31)

In order to determine the required gain of the first stage, we substitute V3 from equation (30) into equation (31). Hence :

IL = (V1 - V2) * R4/R6 * 1/(n*R10) * 22000/(22000 + RL) ;

R4/R6 = (IL/(V1 - V2)) * n * R10 * (22000+RL)/22000 ;

The requirement for the current IL through the electrodes is 100 mA for an input (V1 or V2) of 5 Volt. with n = 53, R10 = 0.4 Ohms

and RL = 1500 Ohms the ratio R4/R6 can be calculated as :

$$R4 / R6 = 0.45 ;$$

As the voltages V1 and V2 can be calibrated with VR 2 and Vr1 respectively between 2.5 and 5 Volts, the amplification factor (R4 / R6) of IC1a has to lie between 0.45 and 2.26/2.5 = 0.9 . So R4 was chosen as 27 KOhms and R6 as 47 KOhms giving an amplification factor of R4 / R6 = 27/47 = 0.574 .

The maximum current flowing through the primary of the pulse transformer and hence through the transistors TR 1 and TR 2 is defined by equation (25) as

$$I_p = I_s * n = 100 \text{ mA} * 53 = 5.3 \text{ Amperes} ;$$

The darlington power transistors (TR 1 = BD 649, TR 2 = BD 646) can sink up to 8 Amperes continuous current which gives a 40 % safety margin.

The heat sinks of TR 1 and TR 2 are designed for the worst case situation of operation, that is a FNS waveform with maximum frequency (100 Hz) and pulse width (800 µs) at half the maximum amplitude (50 mA), because at half the maximum amplitude, the Power dissipated in the transistor, ie. the product of voltage across and the current through the transistor, reaches a maximum. For an electrode current of 50 mA, the current through the primary of the pulse transformer and thus through the transistor TR1 or TR2 is 50 mA * n = 2.65 A (turns ratio n = 53). The collector emitter voltage is then approximately 7 V. Thus the maximum power dissipated by TR1 or TR2 is

$$P = 800 \text{ us} * 100 \text{ Hz} * 7 \text{ V} * 2.65 \text{ A} = 1.5 \text{ W}$$

Thus a small heat sink with a thermal resistance of 20 degree

Celsius per Watt is sufficient (ie. the maximum temperature of the transistor is then approximately 30 degrees Celsius above room temperature).

The capacitors C2 and C3 (Figure C.10) are needed to avoid high frequency oscillation of the amplifier due to phase shifts introduced by the transformer impedance Z (see Figure C.9) which is part of the feedback loop. The values of C2 and C3 were determined experimentally.

The output voltage of the secondary is limited by a metal oxide varistor which essentially has the characteristics of a back to back zener diode. The S 20 K 95 SIEMENS varistor limits the output voltage to less than +- 200 V for output currents up to +- 100 mA. The varistor can dissipate a maximum power of 1 Watt continuously. In a worst case situation (ie. no electrodes connected), all the power generated by the isolation amplifier is dissipated in the varistor. In the case of a biphasic waveform with a negative and positive pulse width of 500 µs (this is the maximum pulse width the pulse transformer can reproduce at a voltage swing of +- 200 V) and a repetition frequency of 50 Hz, the power to be dissipated by the varistor can be calculated as:

P = (2 * 500µs * 50Hz) * 200V * 0.1 A = 1.0 watts ;

As this is the maximum power rating of the varistor, the isolation amplifier should not be driven without electrodes (ie. open circuit) over long periods of time. However, the isolation amplifier was successfully tested in open circuit mode at a pulse repetition frequency of 100 Hz and 500 us symmetrical biphasic waveform for 10 minutes without increasing the case temperature of the varistor above the specified limits.

MEASUREMENT OF THE OUTPUT VOLTAGE

The output voltage of the isolation amplifier is measured with an opto-coupler stage in order to ensure complete isolation between the output and the rest of the circuitry. Thus by measuring the voltage V4 across the electrode at a known current, the electrode impedance can be calculated with Ohms' law. The ligh emitting diode (LED) in the opto-coupler IC3 is connected through a 22 KOhms resistor to the output of the isolation amplifier. Thus at an output voltage V4 = 200 V a current of 9.0 mA flows through the LED. The transfer factor of the opto-coupler is 100 +- 20 % , ie. a current flowing through the opto-couplers' LED initiates an equal current flow through the opto-couplers' photo transistor. The current through the photo transistor creates a voltage V5 over VR3 and R14. The voltage V5 can be adjusted with VR3 to give a sensitivity of 2V per 100 V at the output of the pulse transformer.

The rise time (10% - 90 %) of the combination isolation amplifier, and opto-coupler is 30 µs for a 100 mA pulse into a load of 1500 Ohms. Thus it is easily possible to measure the output voltage V4 and hence calculate the electrode impedance with a single 100 µs long pulse.

For LED currents above 1 mA (ie. output voltages V4 > 25 Volts) the linearity of the opto-coupler is within 5% . The voltage transfer characteristic of the opto-coupler stage can be decribed by the following linear equation :

$$V5 = V4 * K1 + K2; \quad (32)$$

The constants K1 and K2 were determined experimentally in combination with the analogue to digital converter board and the "impedance_check" procedure (see Appendix H). The opto-coupler

voltage V5 was sampled and the actual output voltage V4 determined at the same time with a storage oscilloscope. A linear curve described by equation (32) was fitted through the set of data points thus obtained.

The electrode impedance can thus be determined to within +- 10% over a range from 200 to 2000 Ohms with a 100 mA test pulse.

OPERATION INDICATOR LIGHT

A signalling LED (light emitting diode) is provided to indicate that the respective stimulation channel is active. Unfortunately it is not possible to drive a LED directly with the FNS waveform, because of the low duty cycle (less than 1 %). A LED driven by such a waveform would hardly emit light at reasonable current levels flowing through it. Thus it was decided to trigger a monostable Schmitt-trigger with the input voltage Vin- (see circuit diagram Figure C.10). When triggered this monostable switches the LED D1 on for approximately 15 ms, thus giving a duty cycle of 15ms / 30 ms = 50 % . The on time is determined by the values of C1, R8 and R7.

The trigger threshold V_T is set by R7 and R8 to 0.65 Volts (given the relationship : $V_T = 12\ V * R8 / (R8+R7)$). The current through the LED is limited by the resistor R9 and the maximum output current of IC2 to approximately 20 mA. Super bright red LEDs are used to increase readability in bright environments.

CALIBRATION OF THE ISOLATION AMPLIFIER

A special calibration procedure is included in the software package of the FNS-controller which is described in Appendix H.

USE OF THE ISOLATION AMPLIFIER AS A VOLTAGE AMPLIFIER

The isolation a,plifier is a voltage controlled current source, but it can be reconfigured as a voltage controlled voltage source. To do so only two jumper settings have to be changed. The complete system then has to be recalibrated. The jumper settings for the two modes are (see component layout Figure C.12) :

Current source mode : JMP 1 closed ;
JMP 2 open ;
JMP 3 open ;

Voltage source mode : JMP 1 open ;
JMP 2 closed ;
JMP 3 closed ;

In voltage controlled mode the output of the amplifier with a 1000 Ohms load is given by :

$$V4 = (V_{in+} - V_{in-}) * G * a * n * (1 + R12/R13) =$$

$$= (V_{in+} - V_{in-}) * 62.47 ;$$

where a is the attenuation factor due to primary and secondary winding resistances (see equation (6)), n is the turns ratio, and G is the gain of the differential input stage (G = 0.57);

Thus with V_{in+} and V_{in-} being maximum +5 Volts, the maximum output voltage into a 1000 Ohms load is +- 312.2 Volts. However, as discussed earlier, the varistor limits the output voltage to +- 200 Volts.

CALIBRATION OF THE ISOLATION AMPLIFIER IN VOLTAGE CONTROLLED MODE

As there is no software routine provided to calibrate the isolation amplifier in the voltage controlled mode, the isolation module has to be calibrated separately as follows :

- connect a pulse generator to the negative input of the isolation module (pin 14 a,c of the edge connector);
- set the pulse width to 300 us and the repetition frequency to 50 Hz;
- set the pulse amplitude to + 2.5 Volts;
- connect an oscilloscope to the output of the isolation amplifier (ground is point a , see component layout Figure C.14) ;
- calibrate the output voltage with VR 2 to -100 Volts;
- disconnect the pulse generator from the negative input and connect it to the positive input (pin 16 a,c of the edge connector);
- calibrate the output voltage with VR 1 to + 100 Volts;

Note: In the voltage controlled mode it is not possible to measure the current through the electrodes. Thus it is not possible to calculate the electrode impedance .

MECHANICAL CONSTRUCTION

Each of the six isolation amplifier channels is assembled on a single sided EURO card (size 100 mm x 160 mm) isolation module. Figure C.12 shows the component layout screen of one isolation module.

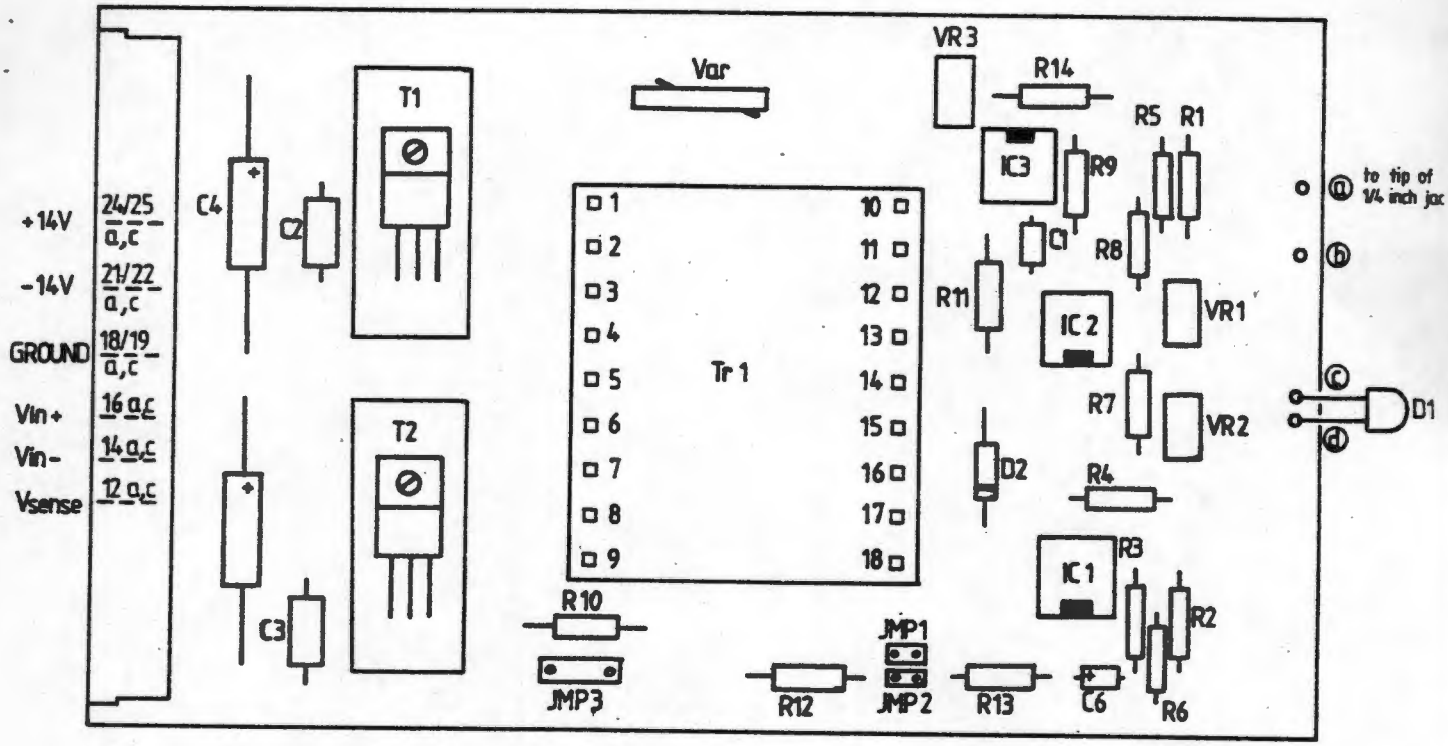


Figure C.12 Component layout isolation amplifier module

PARTS LIST ISOLATION AMPLIFIER

R1	=	2K2	
R2	=	47K	
R3	=	27K	
R4	=	27K	
R5	=	2K2	
R6	=	47K	
R7	=	680K	
R8	=	39K	
R9	=	470	
R10	=	0R4	(2* 1R0 parallel with 1R8)
R11	=	22K	
R12	=	12K	
R13	=	3K9	
R14	=	180	
R15	=	S 20 K 95 Siemens Varistor (code SIOV-S20K95)	
VR1	=	2K2	
VR2	=	2K2	
VR3	=	470	
C1	=	100 nF	
C2	=	100 pF	
C3	=	100 pF	
C4	=	10 uF/25V	
C5	=	10 uF/25V	
C6	=	4.7 uF/25V	

D1 = super bright red LED
D2 = 1 N 4003

TR1 = BD 649
TR2 = BD 646

IC1 = LF 353
IC2 = LM 741
IC3 = 4N 26

T1 = ETD 44/20/15 SIFFERITE CORE (code B66365-G-X127)
with coil former code B66366-A1018-T1, and
two steel yokes code B66366-A2000

JMP 1,2,3 = jumper

64 pin DIN 41 612 edge connector
Siemens ordering code V42254-B-1200-B640

1/4 inch jack stereo (3-pin) socket for panel mounting;

The lines to the power supply and the inputs are connected to a 62 pin edge connector (DIN 41 612). The output is connected to a 1/4 inch (6.35 mm) socket which is mounted to a 55mm x 128 mm aluminium front plate (see Figure C.13)

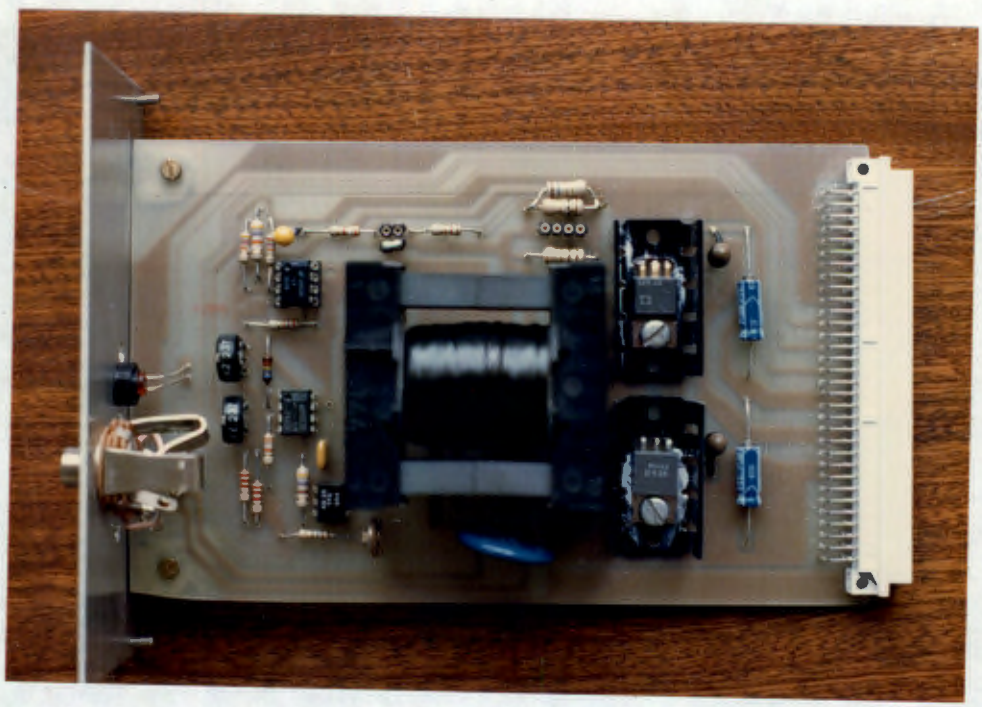


Figure C.13 Isolation amplifier module (one channel)

Six of these cards plug into a 19 inch (482 mm) wide rack. The power supply lines are connected to a 3 pin Cannon connector at the rear of the rack. The connections to and from each isolation module lead to a 25 pin D-connector at the rear of the rack. The 19 inch rack is mounted at the back of the paracycle. A 20 lead 10 m long cable connects the isolation amplifier with the waveform generation board and the analogue to digital converter board in the SPERRY Personal computer. The external power supply of the isolation amplifier is connected via a 3 lead 10 m long cable to the Cannon connector at the rear of the isolation amplifier rack.

POWER REQUIREMENTS OF THE ISOLATION AMPLIFIER

The isolation amplifier modules are supplied with a +- 14 Volt power supply. Each isolation module (ie. one channel) draws an average current of +- 300 mA when supplying a 800 µs, 100mA current pulse at a repetition frequency of 100 Hz (ie. in a worst case condition). However the peak pulse current is about 5 Amperes as shown earlier. A regulated power supply was designed with standard adjustable regulators which can supply +- 14 Volts at a continuous current of 2 Amperes for all 6 channels. To supply the necessary peak currents, the +- 14 V output lines are each buffered with 4700 µF capacitors in the isolation amplifier rack. The DC power supply lines can be disconnected from the isolation amplifier with a double-pole double-throw relay which can be controlled via a TTL compatible input by the computer.

For safety reasons, a toroidal mains transformer was used which allows a spacing of 15 mm between the primary and the secondary windings. The leakage between primary and secondary was

measured as being 50 µA.

The power supply is assembled in a separate earthed aluminium box which is connected with a 10 m long cable to the isolation amplifier.

TESTING OF THE ISOLATION AMPLIFIER SAFETY

The patient leakage current was measured as shown in Figure C.14.

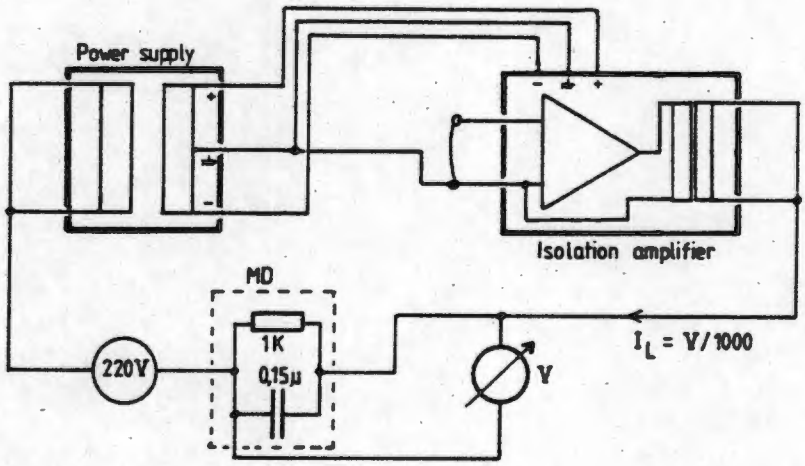


Figure C.14 Measurement of the patient leakage current (IEC, 1977)

The measuring device MD is adapted from the IEC (IEC, 1977). The patient leakage current was determined to be 32 µA which is well below the maximum of 500 µA allowed for Class II equipment.

APPENDIX D : THE Analogue to Digital Converter board

As shown in Figure 4.1 one of the functions of the ADC board is to sense the voltage supplied to the electrodes in order to check (calculate) the electrode impedance, and to measure various control variables. Additionally it was intended to use the ADC board for multi-channel EMG analysis, this being the reason for designing a 12-bit ADC. The dynamic range of a 12-bit ADC is 24 bit higher than that of an 8-bit ADC.

The design of the ADC board is based on the successive approximation analogue to digital converter chip AD 574 A chip of Analog Devices. This device contains a complete 12-bit ADC including the voltage reference and bus interface in one 28 pin dual in line package.

THE AD 574 A FROM ANALOG DEVICES

Producers of A/D converters (like Analog Devices) concentrate more and more upon the successive approximation technique because of its inherent excellent compromise between speed , accuracy and cost.

The AD 574 A chip, around which the ADC board is designed, is a complete 12 bit successive approximation A/D converter. The AD 574 A design is implemented with two LSI chips in one 28 pin ceramic package each containing both analog and digital circuitry, resulting in the maximum performance and flexibility at the lowest cost.

Figure D.1 shows the block diagram of the ADC card. See also circuit diagram Figure D.3 and component layout Figure D.4

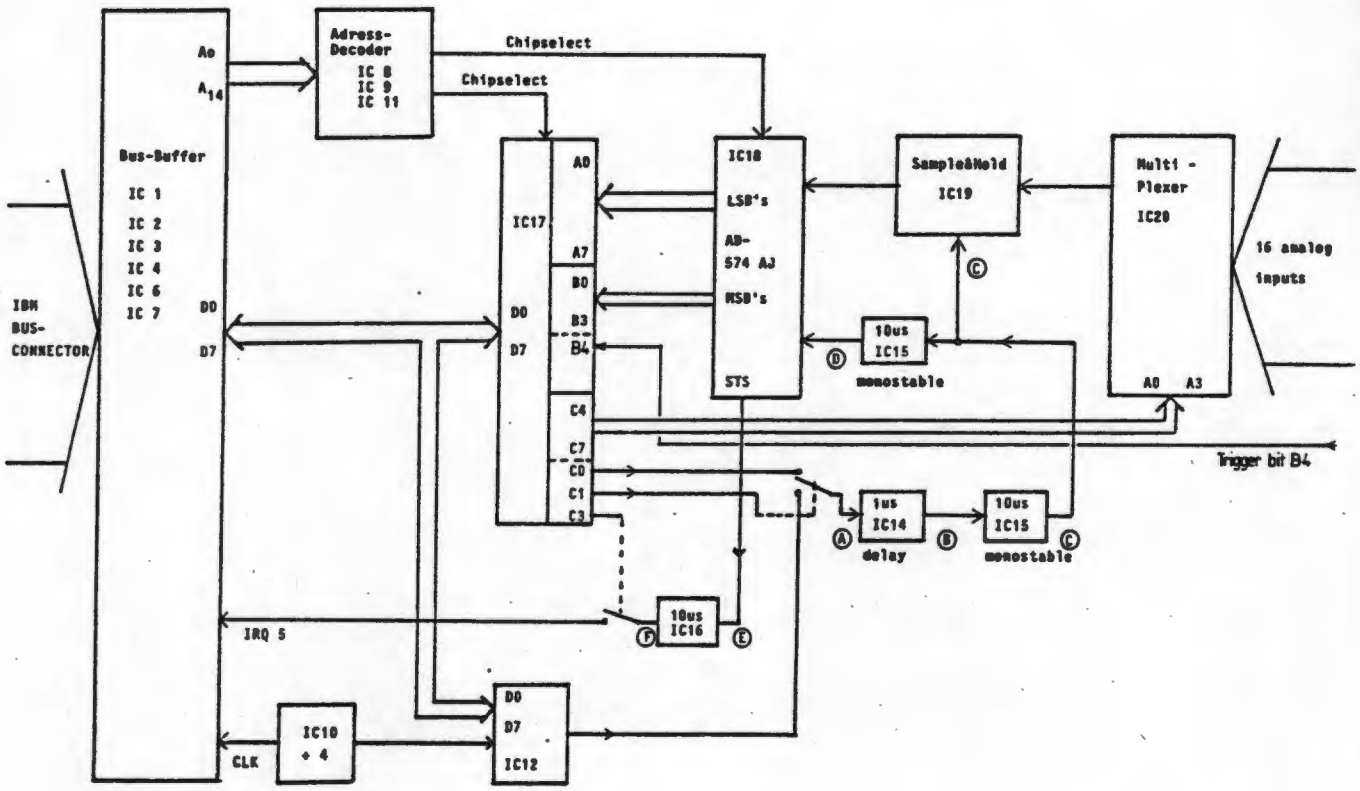


Figure D.1 Block diagram of the ADC board.

BUS BUFFERING AND PROGRAMMABLE PARALLEL INTERFACE (PPI)

In order to keep the load imposed by the ADC board on the IBM bus as small as possible,all the bus lines to and from the IBM bus connector are buffered with 74LSXXX IC's (ie IC 1,2,3,4,6,7). The address decoder (IC 8,9,11) detects the I/O port address of the ADC board, which can be selected as 700 Hex or 780 Hex. As the programmable timer 8354 works only up to 1.5 MHz, the 4.77 MHz IBM system clock (CLK) is divided by 4 (IC 10). The AD 574 AJ converter is connected to the data bus via input port A (8 LSB's) and half of port B (4 MSB's) of the Intel PPI 8355-A5 (IC 17). Port C of the PPI is programmed as an 8 bit output (bit C4 to C7 for multiplexer channel selection; bits C0 to C3 as control lines).

MULTIPLEXER AND SAMPLE & HOLD

To capture the analog data of 16 independent channels, a CMOS multiplexer (IH 6116, IC 20) is used. Note that each analog input Is tied to ground via a 22 K Ohms resistor. Thus the input impedance is equal to 22 K Ohms. To minimise crosstalk between the channels and conversion errors due to the input impedance, the output impedance of the signal source connected to an input should be as small as possible (< 10 Ohms) and unused channels should be tied directly to ground. The input channel number is transferred to the multiplexer via bits C4 to C7 of port C of the PPI.

Because the A/D converter takes about 25 µs to digitize the input signal, changes in this signal level during the actual conversion process could result in errors so that the final digital value would never truly represent the data level prevailing at the instant at which the conversion command was transmitted. A sample and hold chip (LF398, IC 19) is therefore introduced into the Link between the multiplexer and the A/D converter, which makes a fast acquisition of the varying signal and holds this signal during the conversion. For the selected acquisition time of 10 µs the sample error of the LF 398 is about 0.005 %.

As described earlier, the analog to digital conversion is performed by the Analog Devices A/D converter AD 574 AJ. Typical conversion time is 25 µs with an error of 0.025 % (ie. one LSB in 12 bits). With use of the AK version of the AD 574 A this error can be reduced to 0.012 % (ie. 1/2 LSB in 12 bits). At the end of conversion the STS line of the converter goes high and, if enabled with bit C3=1 of port C of the PPI, an interrupt request is sent to the IBM via the IRQ 5 line.

TIMING-DIAGRAM

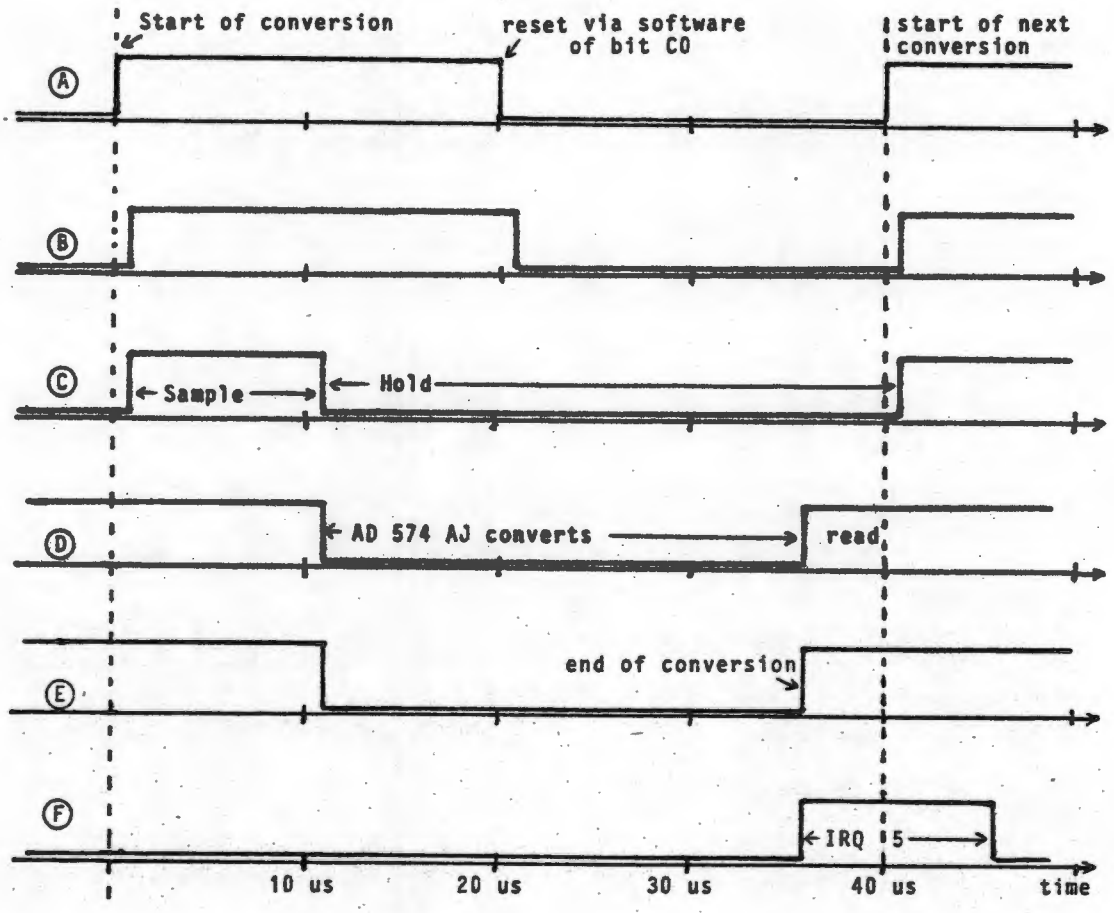


Figure D.2 Timing diagram of a single A/D conversion.

The curves A to F correspond with the signals at the points A to F in the block-diagram (Figure D.1). Note that it makes no difference whether the signal A is generated by setting bit C0 of port C of the PPI via software or by the output of the programmable timer (IC12). Initiation of the conversion via bit C0 is selected if bit C1 of port C is high. If C1 is low, the timer output frequency is used to initiate conversion. Either way the conversion sequence starts with the positive transition at point A, which is the input to the 1 μs delay-block (IC 14).
Conversion sequence:

- setting of the multiplexer channel via bits C4 to C7 of port C of the PPI. (If the timer initiates the conversion, the input channel must have been selected at the last time when the interrupt routine was serving IRQ 5.)
- positive transition of signal A;
- after a 1 µs delay (IC 14) which gives the multiplexer time to settle, sampling is initiated (signal C).
- after 10 us (monostable I of IC 15) sampling ends and conversion is started (monostable II of IC 15), signal D.
- reset of bit C0 (only if the software trigger is used).
- end of conversion is signalled by STS of the converter going high . This triggers a 10 µs long Interrupt request 5 (IRQ 5) pulse (IC 16), signal F .
- the result of the conversion can now be read from ports A and B of the PPI.

It is obvious that the overall conversion time from selecting the multiplexer channel to end of conversion is about 40 µs. This allows a maximum conversion rate of 22000 samples a second. The conversion rate under control of the timer is lower (11 KHz) as the interrupt service routine of the IBM's 8088 processor is somewhat slow (30µs).

SOFTWARE-CLOCK VERSUS TIMER-CLOCK

As already mentioned one of the advantages of the software-clock (or better termed software initiation of conversion) is its higher speed. However the exact conversion rate depends on the execution speed of the user program which is a function of the the CPU clock frequency used in the PC. The clock frequency of the IBM PC is 4.77 MHz. But other PC's differ considerably (eg. the Sperry PC can run at 7.16 MHz). The use of the on-board

programmable timer (see Figure D.1 ,IC 12) to generate a timing signal which triggers sampling avoids this problem as the timer input frequency is always 4.77 MHz / 4 = 1.1925 MHz, regardless which PC is used. Another advantage of the timer-clock and interrupt is that at low sampling rates the CPU is not tied up in timing loops and therefore can perform other tasks at the same time, e.g. online signal analysis or signal presentation on the computer screen.

BINARY OUTPUT CODE OF THE ADC BOARD

The ADC board allows the selection of a single unipolar range, (0 V to +10 V) i.e. 0 Volts input gives digital output 0, and 10 Volts input gives digital output 4095 (= $2^{12}-1$) as the result of the A/D conversion. Thus a change of 2.441 mV (= 10V/4096) in the input voltage changes the digital output of the converter by 1 bit.

For expressing bipolar analog quantities in its two bipolar ranges (-5V to +5V, -10V to +10V) the ADC board uses the binary offset code representation:

- 5 V (-10V)	--->	0
+- 0 V	--->	2048
+ 5 V (+10V)	--->	4095

SPECIFICATIONS

A/D converter AD 574 AJD :

Resolution	:	12 bit
Conversion time	:	25 µs (AD 574 AJD)
	:	40 µs (Multiplexer, S&H, AD 574 AJD)
Input voltage	:	- 5 V to + 5 V
	:	- 10 V to + 10 V
	:	0 V to + 10 V
Linearity	:	11 bit, ie. 0.025 %
Offset	:	adjustable to zero
Linearity drift	:	0.5 ppm/C
Offset drift	:	10 ppm/C
Gain drift	:	50 ppm/C

Sample & Hold LF 398 :

settling time	:	< 10 µs
Sample&Hold error	:	< 0.005 %

Multiplexer IH 6116

settling time	:	< 1 µs
input impedance	:	22 K

JUMPER SELECTION

Board address : Jumper J3 selects the board address to be either 700H or 780H.

Voltage range : Jumper J2 selects the voltage range to be either 10V or 20V.

Biploar/monoploar mode : Jumper J1 selects either bipolar mode (-5V to +5V or -10V to +10V) or monopolar mode (0 to 10V).

CALIBRATION OF THE ADC BOARD

MONOPOLAR MODE

- Select monopolar mode using J1;
- Select 10V range using J2;
- apply a highly stable voltage at input channel 0 and check this input voltage with a digital voltmeter;
- run the sample program (which is supplied with the PC-26 board) and print the values of channel 0 on the screen;
- apply +1,22 mV to channel 0;
- trim VR3 (offset) to give the first transition (0 to 1)
- apply +9,9963 volts to channel 0;
- trim VR2 to give the last transition (4094 to 4095)

BIPOLAR MODE

- select bipolar mode using J1;
- select 10V range using J2;
- apply a highly stable voltage to input channel 0 and check this input voltage with a digital voltmeter;
- run the sample program (which is supplied with the PC-26 board) and print the value of channel 0 on the screen;
- apply -4,9988 volts to input channel 0;
- trim VR1 to give the first transition (0 to 1);
- apply +4,9963 volts to input channel 0;
- trim VR2 to give the last transition (4094 to 4095);

Note : When changing from monopolar to bipolar mode (or vice versa), the ADC board has to be recalibrated!

Table D.1 Port addresses and functions of the ADC board

	ADDRESS	FUNCTION
PORT A	700H or 780H	bits A0 to A7 are the 8 least significant bits of the digital output (A0 is the LSB).
PORT B	701H or 781H	bits B0 tO B3 are the 4 most significant bits of the digital output (B3 is the MSB). bit B4 is used as a TTL trigger input (pin 18 of the 25-D conn.)
PORT C	702H or 782H	bits C4 to C7 are the channel address for the multiplexer (C7 is the MSB). bit C0 software-clock bit C1 softswitch software-clock (C1=1), timer-clock (C1=0) bit C3 interrupt enable (C3=1)

PROGRAMMING OF THE PARALLEL INTERFACE

The three ports of the programmable parallel interface (PPI) chip 8255 must be set up as follows :

Port A : Input (8 LSB's of the A/D)

Port B : Input (4 MSB's of the A/D)

Port C : Output (Input channel,Interrupt control,Software clock)

The ports of the PPI are set to the above modes by sending the control-word 92 Hex to the PPI (Port 703 (or 783) Hex).

PROGRAMMING THE TIMER (8253)

The on-board timer is used to generate a signal for triggering of conversion. The INTEL 8253 timer chip contains three timer blocks which can be independently programmed. In the following section the programming of the timer blocks is described . See the INTEL data sheet for more details.

Port addresses :		
	Counter 0	0784H or 0704H
	Counter 1	0785H or 0705H
	Counter 2	0786H or 0706H
	Control Word Register	0787H or 0707H

Counters 0 and 1 are connected in cascade. It is necessary to program each counter with a mode and quantity.

Control Words :

data bits	D7	D6	D5	D4	D3	D2	D1	D0		
Counter 0	0	0	1	1	0	1	0	0	= 52	= 34H
Counter 1	0	1	1	1	0	1	0	0	= 116	= 74H

```
          counter   RL1  RLO        mode
                      1               1
                      1               1
                      1               1-->rate generator
                      1
                      1
                      1
                      1-->load LS byte first, then MS byte
```

ie. to program counters, output control word first, followed by least significant (LS byte) of the counter, followed by most significant (MS) byte of counter eg.

```
OUT  0787H,  34H
OUT  0784H,  LS byte        Counter 0
OUT  0784H,  MS byte

OUT  0787H,  74H
OUT  0785H,  LS byte        Counter 1
OUT  0785H,  MS byte
```

USING THE TIMER

The system clock (4,77MHZ) is divided by 4 and then serves as the clock input for counter 0.

ie. Input Clock Frequency to counter 0 = 1,176 MHZ

It is illegal to program the counters with the value 1.

If a minimum frequency of 10Hz is adequate, counter 1 can always be programmed with the value 2.

ie. Low byte counter 1 = 02

High byte counter 1 = 00

To determine the value for counter 0 (Count0) for a specific sampling frequency Fs, the following formula is used

$$Count0 = Round\ (\ 1.176 * 10^{6}\ )\ /\ (\ 2 * Fs\ )\ ;$$

To find the high byte and low byte values for counter 0, the Turbo Pascal functions HI and LO can be used.

ie. High byte counter 0 = HI(Count0)

Low byte counter 0 = LO(Count0)

SOFTWARE

Bit C1 of Port C of the PPI is used to select either the Software clock (bit C0) or the Timer clock : C1 = 0 --> Timer clock;

C1 = 1 --> Software clock;

When using a software clock, bit C0 of Port C of the PPI is used to initiate a conversion (positive edge triggered). For port address details see Table D.1 .

SEQUENCE FOR A/D CONVERSION :

I. output channel number and clear bit C0 (bit C1 stays 1) :
ie. in TURBO PASCAL : PORT[$702] = (channel SHL 4) + 2;
(NOTE : PORT [address] is a TURBO PASCAL function to access the physical I/O port at the given I/O address)

II. output channel number and initiate conversion (ie set C0=1):
ie. in TURBO PASCAL : PORT[$702] = (channel SHL 4) + 3;

III. wait 40 us till end of conversion

IV. read A/D value from Ports A and B, masking 4 MSB's of Port B :
ie. in TURBO PASCAL :

```
ADSAMPLE := (Port[$701] AND $0F) * 256 + Port[$700];
```

In Turbo Pascal this conversion sequence is part of the FUNCTION ADSAMPLE (see listing). This FUNCTION is called with the channel number as it's argument, and it returns the value of the selected channel. The 40 us wait state is realized with a short FOR - DO loop.

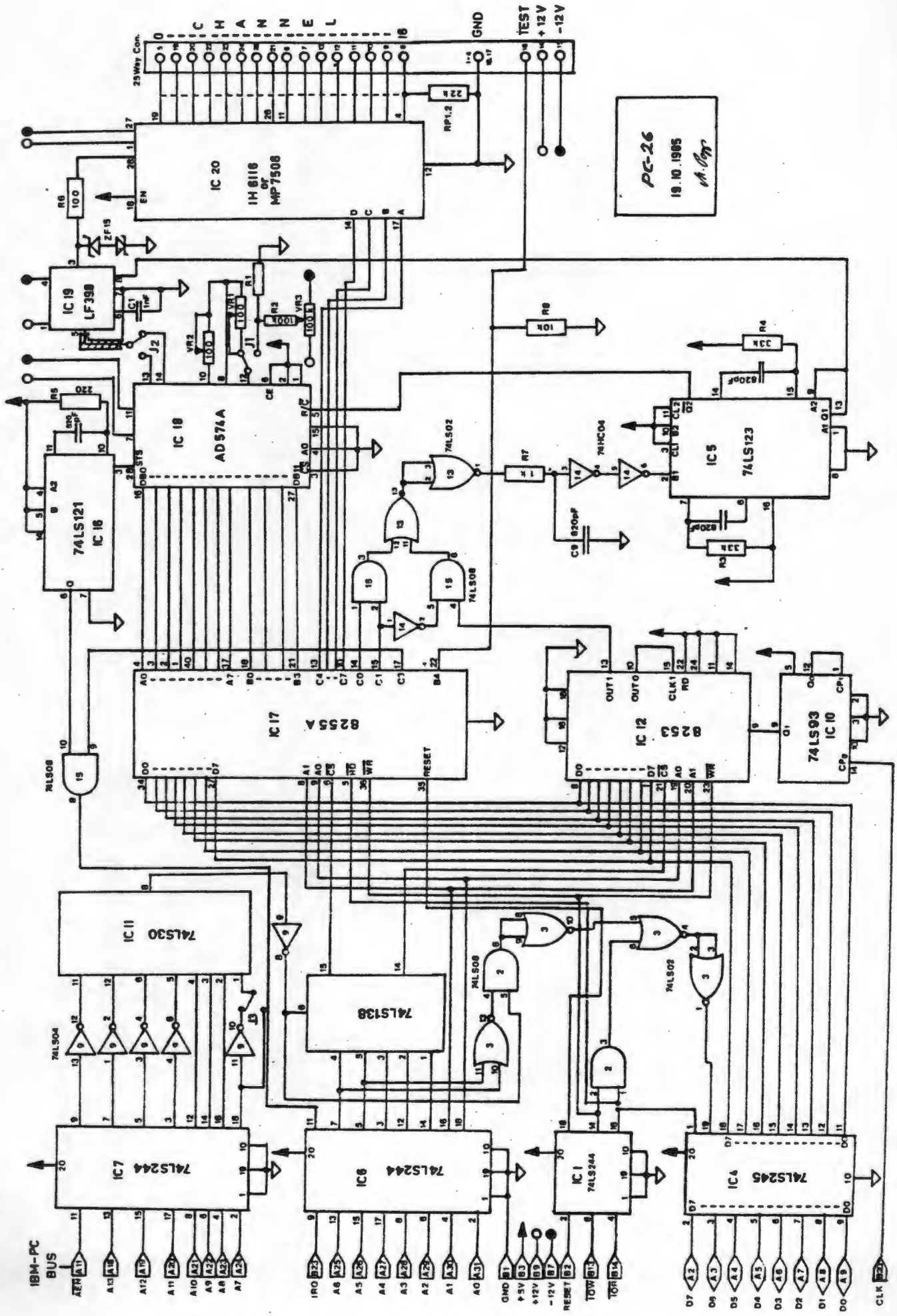
TRIGGER BIT B4

Bit B4 of the 8255 PPI can be used as a TTL compatible input to start data collection. Bit B4 is wired to pin 18 of the 25-D connector. Note that any voltages exceding the TTL range of 0 to 5V can damage the 8255 PPI !

Mechanical design

The ADC board is assembled on a double-sided through-hole-plated PC board. The 16 analogue inputs are accessible via 25-pin D-connector at the rear of the board. The ADC board is now locally manufactured and is available from the company EAGLE ELECTRIC in Cape Town under product code number PC-26 (Popp, 1986).

Figure D.3 Circuit diagram ADC board (PC-26)



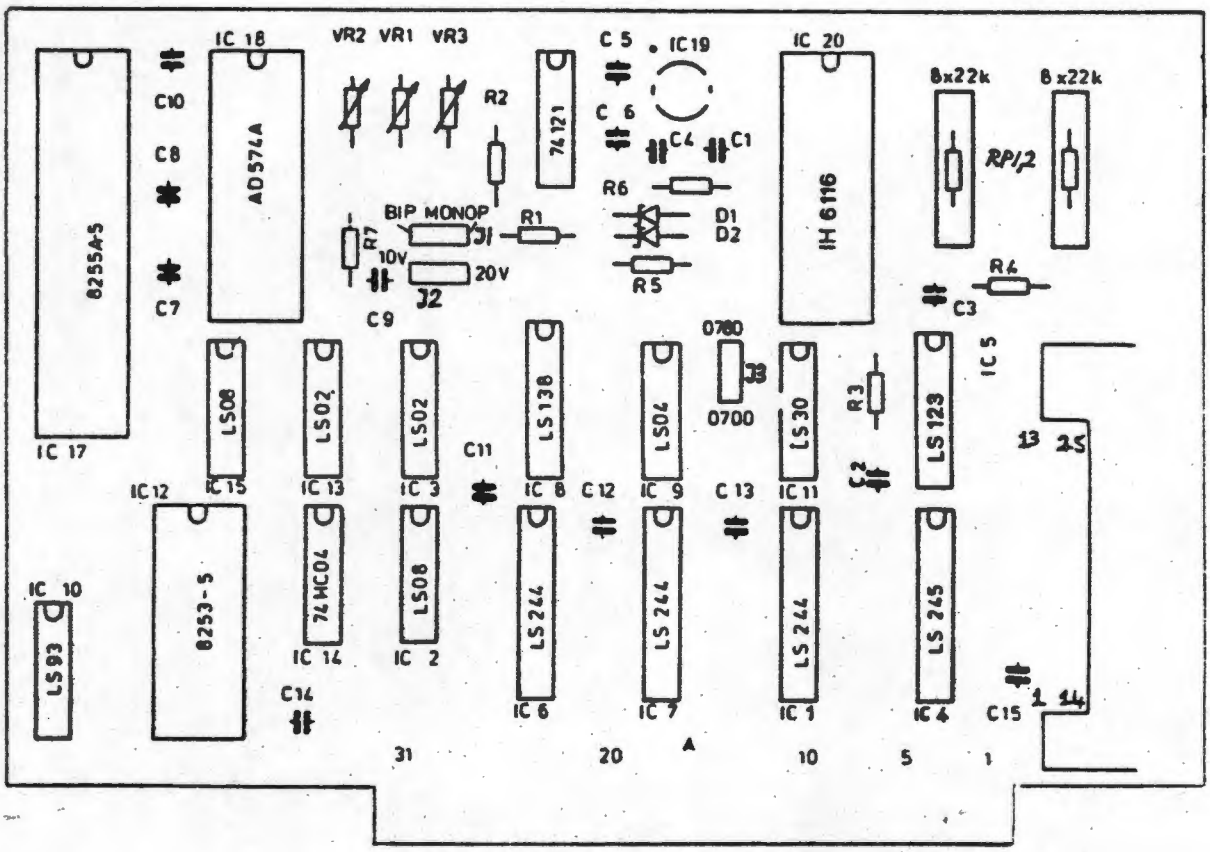


Figure D.4 Component layout ADC board (PC-26)

Signals at the 25 PIN D-CONNECTOR

PIN		FUNCTION
1	=	GND
2	=	GND
3	=	GND
4	=	GND
5	=	Channel 0
6	=	channel 8
7	=	channel 9
8	=	channel 15
9	=	channel 14
10	=	channel 13
11	=	channel 12
12	=	channel 11
13	=	channel 10
14	=	+ 12 V
15	=	- 12 V
16	=	GND
17	=	GND
18	=	bit B4 of 8255 PPI

19 = channel 1
20 = channel 2
21 = channel 7
22 = channel 3
23 = channel 4
24 = channel 5
25 = channel 6

PARTLIST ADC BOARD

CONNECTOR

1x 25 pin D connector for PC-moniting

IC's :

IC 1	LS 244
IC 2	LS 08
IC 3	LS 02
IC 4	LS 245
IC 5	LS 123
IC 6	LS 244
IC 7	LS 244
IC 8	LS 138
IC 9	LS 04
IC 10	LS 93
IC 11	LS 30
IC 12	8253-5
IC 13	LS 02
IC 14	74 HC 04 (National)
IC 15	LS 08
IC 16	74121
IC 17	8255A-5
IC 18	AD 574 A (Analog Devices)
IC 19	LF 398
IC 20	IH 6116 (INTERSIL)

DIODES

D 1,2 ZF 15 or ZPD 15

RESISTORS

R 1	470
R 2	100k
R 3	33k
R 4	33k
R 5	22k
R 6	100
R 7	1K
R 8	10K

RP 1,2 8x 22k resistor network

VARIABLE RESISTORS

VR 1,2	100	(Bourns)
VR 3	100k	(Bourns)

CAPACITORS

C 1	1000 pF	(polystyrene)
C 2,3,4	820 pF	
C 9	1.5 nF	

all other capacitors are 10 nF (ceramic) or 4.7 µF/15V (tantalum) block-capacitors.

APPENDIX E : PULSE-WIDTH MODULATED CONTROL FOR THE PARACYCLE MOTOR

The purpose of the PWM motor controller is to control a 24 Volt wheelchair motor which is connected via gears and chain to the crank of the paracycle. During the so called assist mode the motor drives the mobile paracycle and the subject's legs. If the subject's muscles are strong enough to drive the paracycle it should be possible to impose a variable load onto the subject using the motor.

Problems:
- finding a suitable wheelchair motor for 24 Volt and 200 watt power. The motor should also be able to operate as a controllable dynamic brake.
- regulation of the motor speed with high efficiency.
- finding a suitable switching device which can handle currents up to 30 amperes DC and 110 amperes pulsed.

Figure E.1 shows the circuit diagram of the pulsewidth-modulated motor controller.

The operational amplifier OP1 and OP2 generate a 20 kHz triangular waveform. OP3 acts as a comparator comparing the triangular waveform with the DC level which is determined by the speed control input and the back EMF of the motor. If the motor speed is reduced because of increasing load the back EMF falls and the voltage at the non-inverting input of OP4 rises. If the speed control voltage at the inverting input of OP4 is kept constant, the DC level on the noninverting input of OP3 rises. The effect is that the rising comparator level leads to a prolonged on-time of the pulses at the output of OP3 and the current through the motor increases thus counteracting the drop in motor speed.

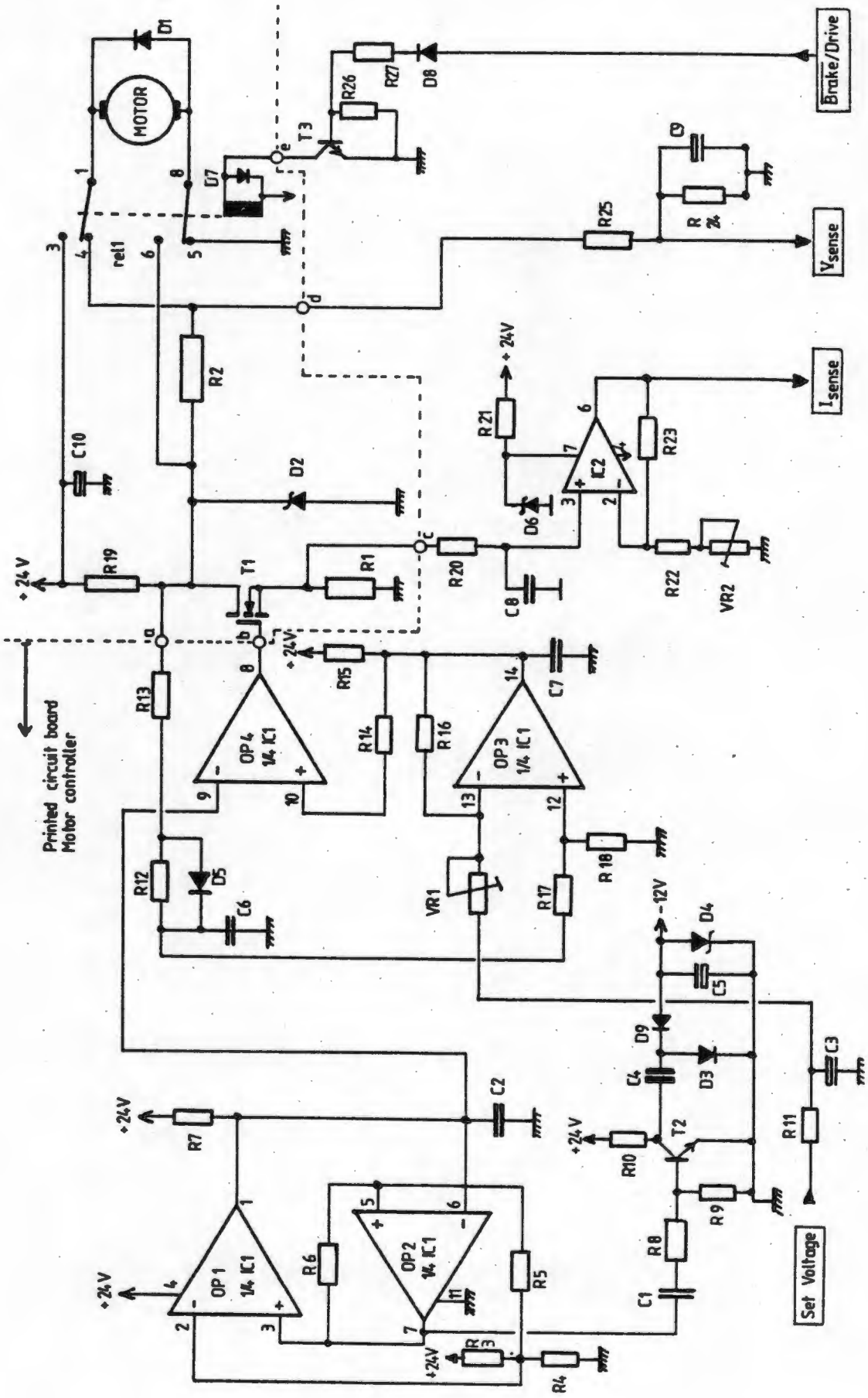


Figure E.1 Circuit diagram pulse-width modulated motor controller

A Siemens (SIPMOS) VMOS power FET (T1) was selected as the power switch for the following reasons:

- ease of interfacing to low power IC circuitry;
- freedom from second breakdown makes a VMOS-FET device highly suitable for driving inductive loads like a motor;
- high current ratings (DC 40 A, pulsed 140 A);
- low drain/source on impedance of 35 milli Ohms results in high efficiency;

When switching inductive loads the peak voltage spike must be limited to a value below the breakdown voltage of the switching device (BUZ-13 : V_{DSmax} = 50 V). The spikes caused by switching off the inductive motor load is effectively handled by the free-wheeling diode D1. The low impedance of the diode causes the current to have a long decay time, at a low voltage. A BYS 26 is used as free-wheeling diode D1. To prevent short peakes reaching the maximal V_{DS} of the MOS-FET an additional zener diode is used. The zener diode D2 responds in picoseconds and protects the VMOS from supply transients as well as inductive spikes.

As the motor resistance is only 0.7 Ohms when the motor stands still, the switching device must be able to handle 24V/.7 Ohms = 34 Amperes. The BUZ 13 can handle 40 A DC current and 140 A pulsed current (300µs on, 2% duty cycle). For this motor one BUZ-13 is therefore sufficient to stay within the safe operating area.

MOTOR DRIVES THE SUBJECT

The voltage at the drain of the MOS-FET is measured with respect to ground. This voltage is divided by a factor of 5 and integrated with a 1 uF capacitor. The actual voltage across the motor terminals can be calculated as:

$$V_{motor} = V_{battery} - 5 * V_{Vsense} ;$$

The current through the motor is measured as a voltage drop over a 25 milli Ohms resistor (R1) in the source lead of the MOS-FET. This voltage is amplified by a factor of 8 thus giving 200 mV per 1A motor current. As the voltage across the current sensing resistor is negative if the motor acts as a brake the amplifier IC2 must be supplied with a symmetrical voltage. T2 drives a diode pump which generates this -12 V from the clock frequency (20 kHz) of the PWM.

MOTOR ACTS AS A BRAKE

The transistor T3 drives a relay (rel 1) which enables the user to switch between the state in which the motor drives the paracycle and the patient and the state in which the motor acts as a variable load for the patient. If the patient drives the motor an EMF voltage is generated across the motor's terminals. This EMF can be shorted to ground with the PWM's switching MOS-FET and an external load resistor in order to brake the motor's movement. The degree of braking can be controlled with the PWM's speed control input. The breaking power can be calculated from the generated EMF and the sensed current.

RANGE OF VOLTAGE TO SET THE MOTOR VOLTAGE/BRAKE VOLTAGE:

	set voltage		motor voltage
motor drives patient :	0 V	--->	+ 24 V
	+ 5.6 V (*)	--->	0 V
motor brakes patient :	0 V	--->	maximum brake effect;
	5.6 V (*)	--->	no brake effect;

(*) adjustable with VR 2 ;

SENSITIVITY OF MOTOR VOLTAGE SENSING OUTPUT

motor drives patient V_{sense} = (V supply - V motor) / 5 ;

motor brakes patient V_{sense} = V motor / 5 ;

SENSITIVITY OF MOTOR CURRENT SENSING OUTPUT

motor braking or driving patient : I_{sense} = 200 mV / A (*) ;

(*) adjustable with VR 1 ;

Partlist PWM motor controller :

Resistors :
R1 = 0.025 Ohms (wire bridge)
R2 = 1 Ohm / 200 Watt (load resistor)
R3 = 560 K
R4 = 220 K
R5 = 1.5 M
R6 = 1 M
R7 = 22 K
R8 = 1 K
R9 = 47 K
R10 = 470
R11 = 1 K
R12 = 1 M
R13 = 100 K
R14 = 100 K
R15 = 100 K
R16 = 47 K

R17 = 100 K
R18 = 120 K
R19 = 220
R20 = 2.2 K
R21 = 1 K
R22 = 22 K
R23 = 180 K
R24 = 500
R25 = 2 K
R26 = 47 K
R27 = 1 K

Variable Resistors

VR 1 = 22 K
VR 2 = 5 K

Capacitors

C1 = 0.47 uF
C2 = 56 nF
C3 = 4.7 uF
C4 = 10 uF / 25 V
C5 = 10 uF / 16 V
C6 = 0.1 uF
C7 = 0.1 uF
C8 = 56 nF
C9 = 1 uF / 10 V
C10 = 10 uF / 25 V

Integrated circuits

IC1 = TL 084 (Texas Instruments)
IC2 = LM 308

Transistors

T1 = BUZ 13 (Siemens SIPMOS)
T2 = 2N 2222
T3 = 2N 2222

Diodes

D1 = BYS 26 (Siemens)
D2 = Zener diode 43 V / 10 Watt
D3 = 1N 4002
D4 = BZY 12 (zener diode 12V)
D5 = 1N 4002
D6 = BZY 12 (zener diode 12V)
D7 = 1N 4002
D8 = 1N 4002
D9 = 1N 4002

Relay

rel 1 = double pole double throw relay, 24 V coil voltage, 30 A max. switching current

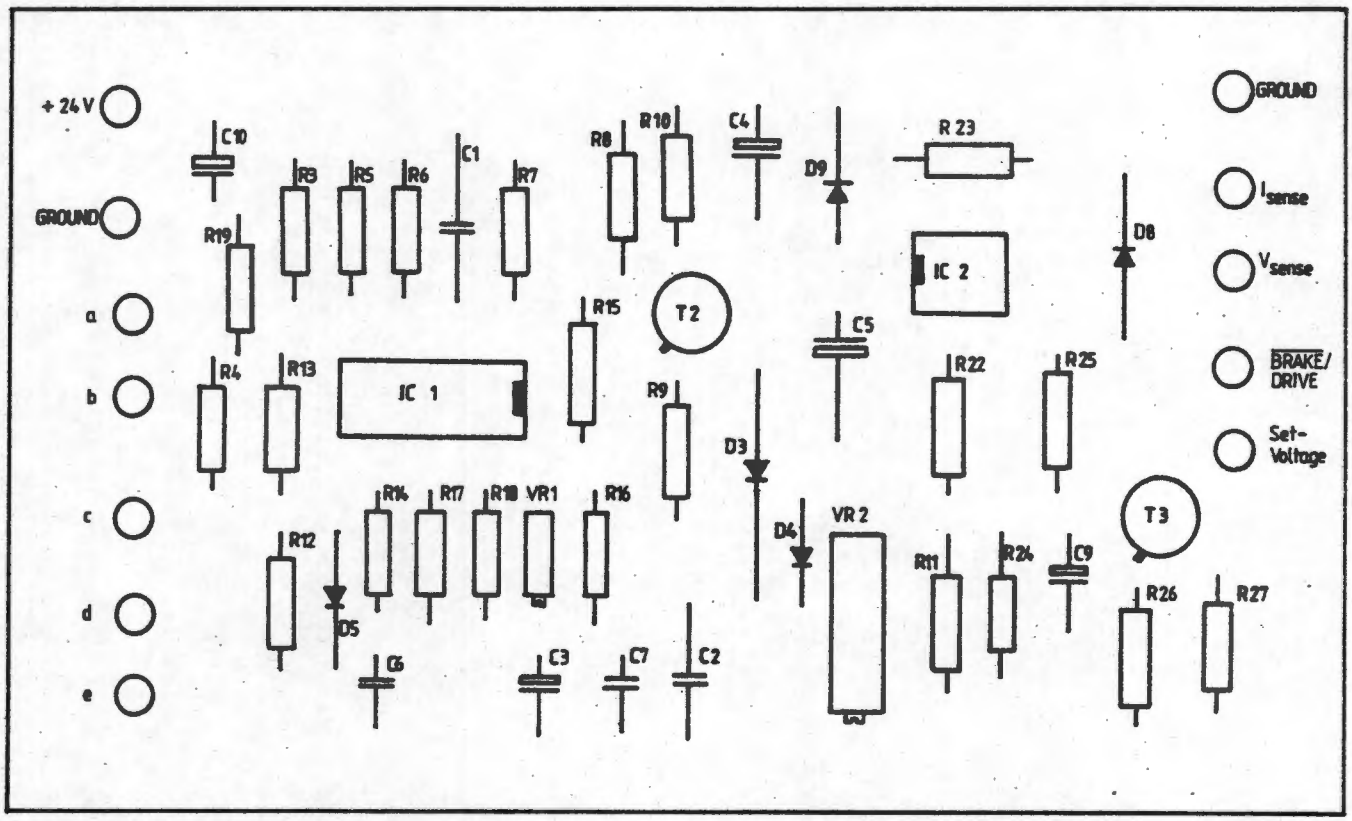


Figure E.2 Component layout motor controller

APPENDIX F : THE SHAFT ENCODER INTERFACE

The interface for the incremental optical shaft encoder of Hewlett & Packard was designed and construced by Peter Kolb and Gisela Hefftner (Kolb 1986). The main technical features are documented in the following.

ADDRESS DECODING

addresses 77C --> 77F or 7FC -->7FF selected by JMP1

8255 (PPI) addressing - Port A = 77C or 7FC (shaft angle)
Port B = 77D or 7FD (speed)
Port C = 77E or 7FE (general usage)
Control word = 77F or 7FF

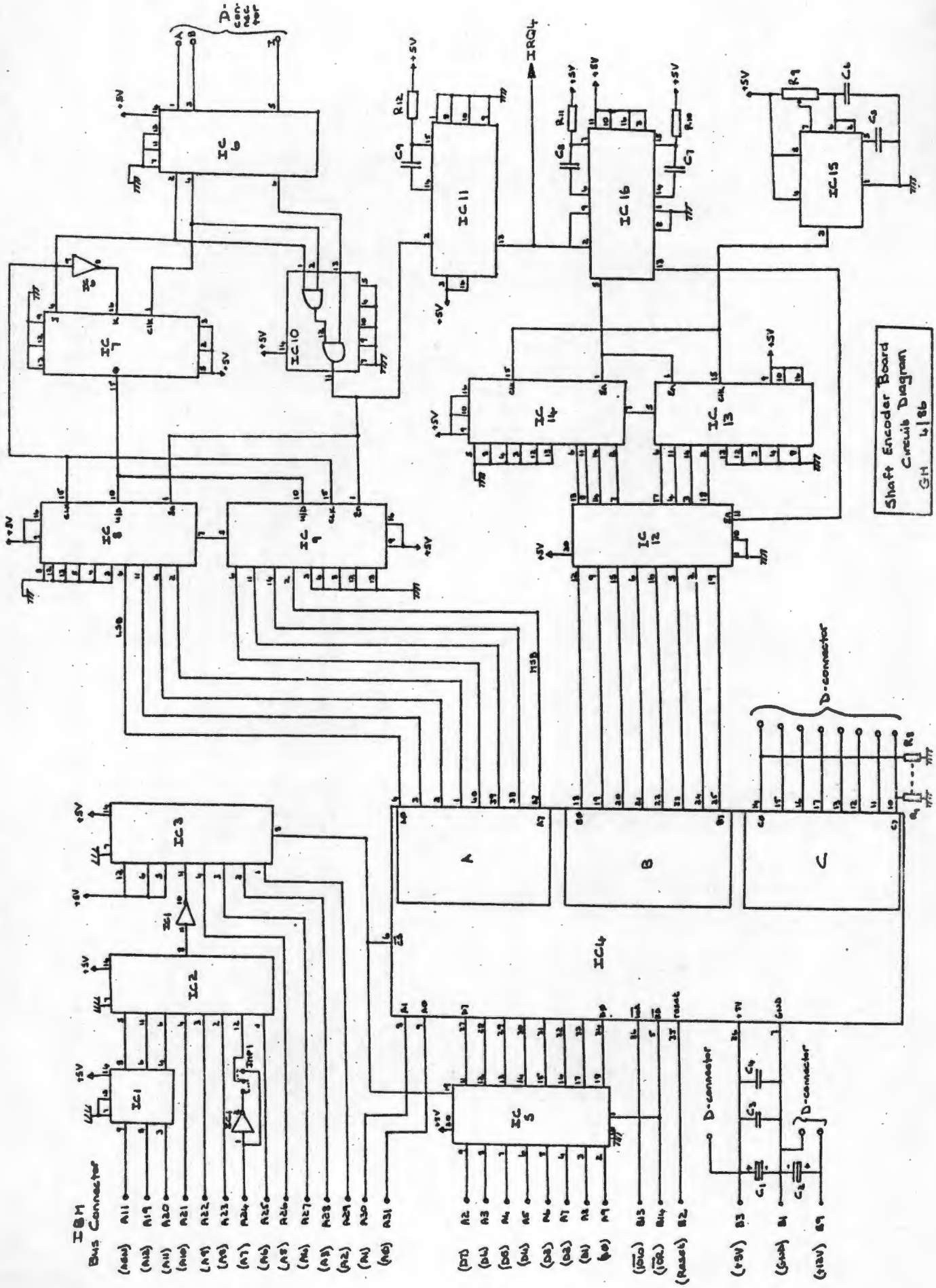
initialization : control word = $93 (Ports A and B input, C7 to C4 output and C3 to C0 input).

SHAFT ENCODER INTERFACE - CIRCUIT DESCRIPTION

Three input signals, A, B and I are obtained from the shaft encoder. A and B are pulses obtained from two optical sensors spaced 90 degrees apart with respect to the windows in the optical disk which rotates with the shaft.

Two 4-bit counters (IC8, IC9) record the angle. The index pulse C resets the counters once per revolution. The counters are clocked by signal A. Signal B, in conjunction with A, is used to determine the direction of rotation, this information being stored in a JK flip-flop (IC7). The output of the JK flip-flop controls the up down count mode of the 4-bit counters.

Figure F.1 Circuit diagram Shaft encoder interface (overleaf)


Shaft Encoder Board
Circuit Diagram
GH 4/86
IC 1
IC 2
IC 3
IC 4
IC 5
IC 6
IC 7
IC 8
IC 9
IC 10
IC 11
IC 12
IC 13
IC 14
IC 15
IC 16
IRQ4
+5V
D-connector
IBM
Bus Connector

SPEED DETERMINATION

The 555 Timer (IC15) is set at a frequency of 50 Hz. The index signal is used to reset the two counters and to enable the latches. Thus the speed is updated once every revolution when the index is passed.

PARTS LIST SHAFT ENCODER INTERFACE

Integrated Circuits

IC1	74LS04	Hex inverters
IC2,3	74LS30	8 input NAND gates
IC4	8355A	Programmable peripheral interface
IC5	74LS245	TRI-STATE octal buffers
IC6	74C14	Hex Schmitt Triggers
IC7	74C76	Dual J-K Flip-flops
IC8,9	4029	4-bit up/down counters
IC10	74C08	Quad 2-input AND gates
IC11	74LSS123	Dual retriggerable one shots
IC12	74C373	TRI-STATE Octal D-type latch
IC13,14	4029	4-bit up/down counters
IC15	LM555	
IC16	74LS123	Dual retriggerable one shots

Capacitors

C1,C2	10 uF tantalum
C3,C4,C6	0.1 uF
C5	0.01 uF
C7,C8,C9	820 pF

Resistors

R1...R8	22K
R9	200K linear pot.
R10,R11	22K
R12	82K

15 PIN D-CONNECTOR - SHAFT ENCODER BOARD :

pin number	assignment
1	GROUND
2	A
3	C0
4	C2
5	C4
6	C6
7	GROUND
8	+ 5V
9	I
10	B
11	C1
12	C3
13	C5
14	C7
15	+ 12V

APPENDIX G : WIRING OF THE UCT FNS PARACYCLE SYSTEM

In the following the information is provided to combine the various components of the FNS-controller and the Paracycle. The plug layouts of the various expansion boards are summarized and a circuit diagram of the complete Paracycle - FNS-contrlloer system is shown.

15 pin D-connector Waveform generation board :

pin number	assignment
1	out D/A 1
2	channel 6 +
3	channel 5 +
4	channel 4 -
5	channel 3 -
6	channel 2 -
7	channel 1 +
8	GROUND
9	out D/A 2
10	channel 6 -
11	channel 5 -
12	channel 3 +
13	channel 2 +
14	channel 4 +
15	channel 1 -

25 pin D connector of isolation amplifier rack :

pin 1,2 = Ground

pin 5 =	ch 1 +	(white/red)	
pin 6 =	ch 1 -	(white/blue)	
pin 7 =	Vout ch 1	(red/black)	2 V for 100 V at output
pin 8 =	ch 2 +	(yellow/red)	
pin 9 =	ch 2 -	(yellow/blue)	
pin 10 =	Vout ch 2	(red/brown)	"
pin 11 =	ch 3 +	(green)	(the Vout's of every stimulation channel are connected to the re- ective channel of the A/D board ie. stimula- tion ch 1 to channel 1 of the A/D board etc.)
pin 12 =	ch 3 -	(green/red)	
pin 13 =	Vout ch 3	(red/violett)	
pin 14 =	ch 4 +	(light blue)	
pin 15 =	ch 4 -	(blue)	
pin 16 =	Vout ch 4	(red)	
pin 17 =	ch 5 +	(yellow)	
pin 18 =	ch 5 -	(brown)	
pin 19 =	Vout ch 5	(violett)	"

pin 20 = ch 6 + (pink)
pin 21 = ch 6 - (grey)
pin 22 = Vout ch 6 (orange) "

(the colors in brackets refer to the colors of the 20 lead 0.22 mm cable which connects the iso-amplifier to the waveform generation board in the IBM PC)

Power supply Canon connector :

1 = + 14 V
2 = - 14 V
3 = GROUND

PWM Motor Controller connector : (DIN 180 degree 5 pole)

pin 1 = set the motor voltage (red), to DA 1 of stim-board;

pin 2 = Ground (black);

pin 3 = output of motor current I (brown); to channel 7 A/D;

pin 4 = brake/drive (green), (brake=0,drive=5V) to DA 2 stim-board

pin 5 = output motor voltage V (white); to channel 8 A/D;

(colors refer to the connecting cable to the A/D board or the stimulation board)

ADC BOARD (PC-26) 25 PIN D-CONNECTOR

pin	function
1	GROUND
2	GROUND
3	GROUND
4	GROUND
5	Channel 0
6	Channel 8
7	Channel 9
8	Channel 15
9	Channel 14
10	Channel 13
11	Channel 12
12	Channel 11
13	Channel 10
14	+ 12 V

15	- 12 V
16	GROUND
17	GROUND
18	bit B4 of 8255A PPI
19	Channel 1
20	Channel 2
21	Channel 7
22	Channel 3
23	Channel 4
24	Channel 5
25	Channel 6

15 pin D-connector - Shaft Encoder Board :

pin number	assignment
1	GROUND
2	A
3	C0
4	C2
5	C4
6	C6
7	GROUND
8	+ 5V
9	I
10	B
11	C1
12	C3
13	C5
14	C7
15	+ 12V

Hewlett & Packard Shaft Encoder - Connections

lead	function
1 (dark blue end)	A (green)
2	+ 5V (blue)
3	GROUND (purple)
8	B (grey)
10	I (white)

PARACYCLE SIGNAL CABLE : 25 pin D-connector on paracycle

colour	pin	function
NC	1	-
NC	2	-
yellow	3	-
red	4	-
white	5	right hand switch (white)
purple	6	right hand switch (white)
black	7	GROUND
white + blue	8	shaft encoder - GROUND (purple)
yellow + green	9	shaft encoder - I (white)
red + brown	10	shaft encoder - B (grey)
brown	11	shaft encoder - +5V (blue)
red + black	12	shaft encoder - A (green)
NC	13	-
NC	14	-
orange	15	left hand switch + speed pot. (white)
pink	16	speed pot. (shield)
turquoise	17	speed pot. (black)
grey	18	left hand switch (red)
red + blue	19	motor controller - GROUND
green	20	motor controller - GROUND (black)
blue	21	motor controller - brake/drive (green)
green + red	22	motor controller - I out (brown)
yellow + red	23	motor controller - V out (white)
white + red	24	motor controller - input voltage setting (red)
NC	25	-

PARACYCLE SIGNAL CABLE : non-paracycle end connections

colour	connected to :-
NC	-
NC	-
yellow	-
red	-
white	optional safety box (active high)
purple	shaft encoder board - pin 8 (+5V)
black	safety box (GROUND)
white + blue	optional shaft encoder board - pin 1 (GROUND)
yellow + green	shaft encoder board - pin 9 (I)
red + brown	shaft encoder board - pin 10 (B)
brown	shaft encoder board - pin 8 (+5V)
red + black	shaft encoder board - pin 2 (A)
NC	-
NC	-
orange	shaft encoder board - pin 8 (+5V)
pink	shaft encoder board - pin 7 (GROUND)
turquoise	ADC board - pin 7 (channel 9)
grey	shaft encoder board - pin 3 (CO)
red + blue	shaft encoder board - pin 1 (GROUND)
green	stimulator board - pin 8 (GROUND)
blue	shaft encoder board - pin 5 (C4) (cw = $93)
green + red	ADC board - pin 21 (channel 7)
yellow + red	ADC board - pin 6 (channel 8)
white + red	waveform generation board - pin 1 (out D/A 1)
NC	-

Figure G.1 shows the connections between the various components of the FNS-controller / Paracycle system.

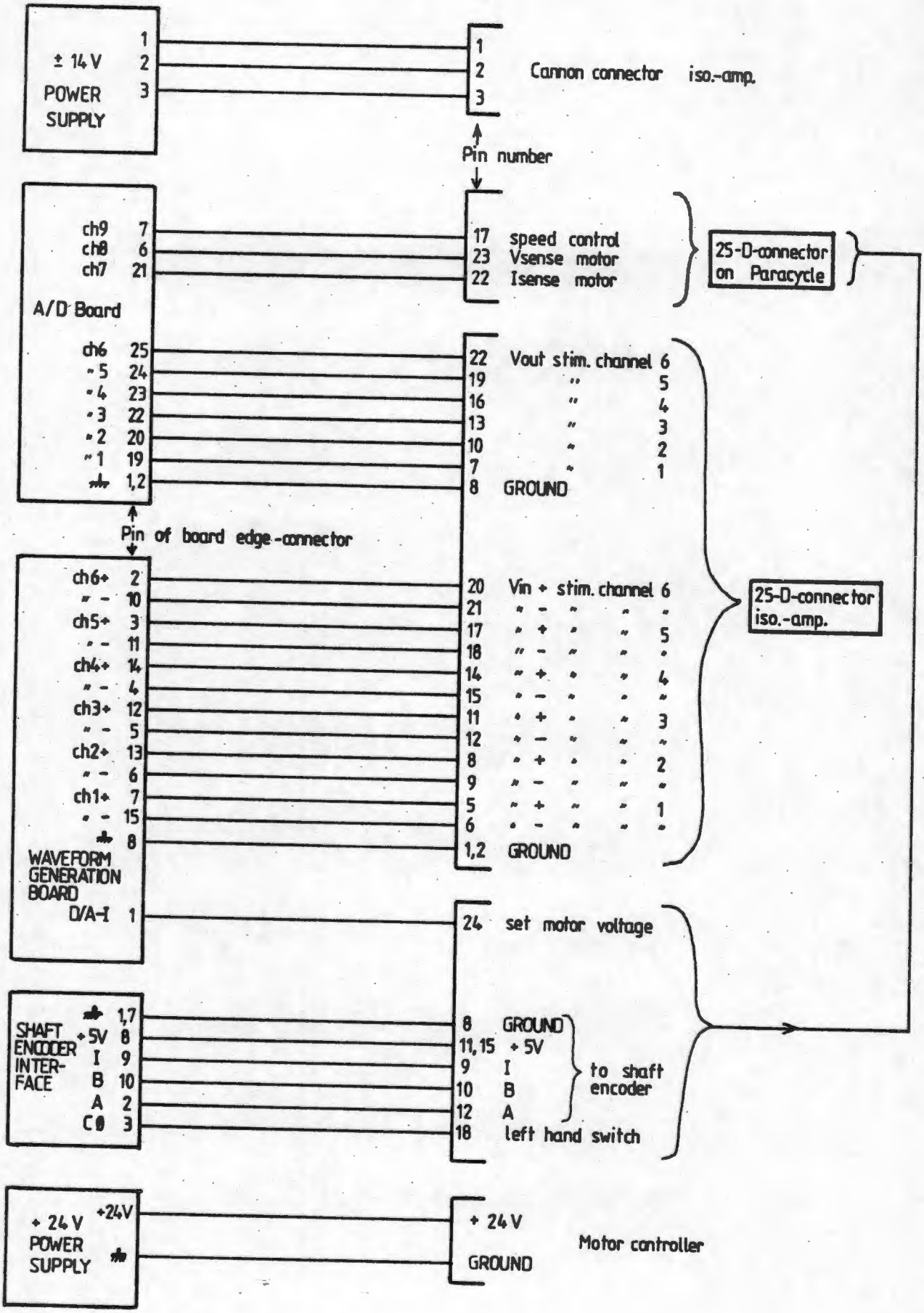


Figure G.1 FNS-controller / Paracycle system

APPENDIX H : THE SOFTWARE PACKAGE FOR THE FNS CONTROLLER

The software package for the FNS controller is written in TURBO PASCAL Version 3.01A supplemented with the TURBO GRAPHIX TOOLBOX utilities of Borland International Inc. . The TURBO GRAPHIX TOOLBOX allows the development of high resolution monochrome graphics for the IBM Personal computer and compatible computers using the standard IBM colour graphics adapter card. Prerequisite for the understanding of this documentation is a good working knowledge of TURBO PASCAL and the TURBO GRAPHIX TOOLBOX.

The FNS-2 program runs on the IBM Personal Computer and fully compatible computers. The minimum requirement in terms of memory capacity is 256 KByte of RAM. A standard IBM colour graphics adapter and a colour graphics screen must be installed (This package does not run with a Hercules graphics card or the IBM monochrome adapter card).

As about a third of the program code is located in overlay procedures the program execution speed is greatly dependent on disk access times. To run the FNS-2 program from a hard disk or even a RAM disk greatly reduces disk access times and disk wear. To run the package from a RAM-disk requires a minimum of 384 KByte memory of RAM.

Documentation of the FNS controller software package includes:

- an index of the names of source code and help files ;
- an index of the names of the TURBO GRAPHIX files necessary for compilation;
- a general overview of the program structure ;
- an index of all the TURBO GRAPHIX procedures used ;
- an alphabetical listing of the procedure and function names ;
- an index of the procedures and functions in sequence of their appearance in the source code;
- an annotated listing of the complete source code ;

NAMES OF SOURCE CODE FILES :

FNS-2.PAS	the main program which contains the global types, variables and constants definitions;
TG-UTIL2.PAS	include file containing utility procedures;
EDITFILE.PAS	include file containing file handhandling procedures;
INPUTS.OVL	include file containing overlay procedures;
STIMULAT.OVL	include file containing the overlay procedure start_stimulation;

NAMES OF HELP TEXT FILES

MAIN.HLP	Help file for the main opening menu;
INP-STIM.HLP	Help file for menu input stimulation parameters;
WAVEFORM.HLP	Help file for frequency definition menu;
ENVELOPE.HLP	Help file for stimulation envelope definition;
P-WIDTH.HLP	Help file for pulse width definition;
STIM.HLP	Help file for stimulation process;

TURBO GRAPHIX INCLUDE FILES NECESSARY FOR COMPILATION

To compile the FNS-2.PAS program the TURBO PASCAL compiler and the following TURBO GRAPHIX INCLUDE files must be present on a floppy disk in system drive a :

TYPEDEF.SYS GRAPHIX.SYS KERNEL.SYS
WINDOWS.SYS POLYGON.SYS

For compilation of the program FNS-2.PAS all the source code files should be present on a floppy disk in system drive b . The compilation results in two files :

FNS-2.COM ..... the compiled code of the main program;

FNS-2.000 ..... the overlay file containing the compiled code of the overlay procedures;

To run the FNS-2.COM program the following files have to be present on the same disk as FNS-2.COM and FNS-2.000 :

ERROR.MSG ..... file with error messages of TURBO GRAPHIX TOOLBOX ;

8x8.FON ..... file containing high resolution font for the IBM graphics card;

4x6.FON ..... file containing TURBO GRAPHIX font;

If the help facility is wanted the above-mentioned help text files must also be present on the same disk.

GENERAL OVERVIEW OF THE PROGRAM STRUCTURE

Figure H.1 shows a simplified flow chart of the program. The procedure names are used to indicate which procedure calls what other procedure.

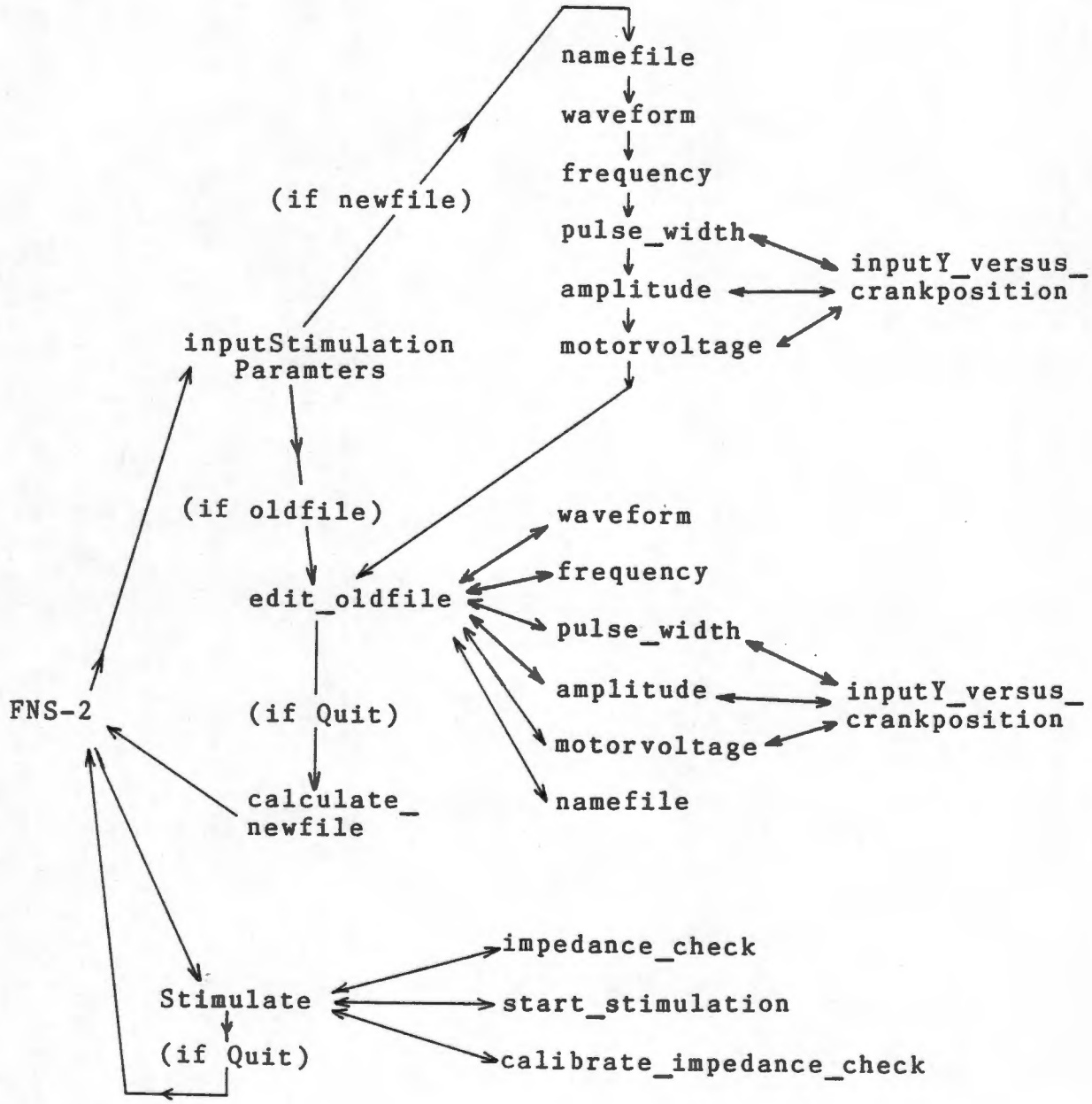


Figure H.1 Structure of the program

The main program FNS-2 calls either procedure inptStimulation-Parameters or procedure Stimulate.

If a new file name is selected the procedures namefile, waveform, frequency, pulse_width, amplitude and motorvoltage are called sequentially to enter the stimulation data for a new patient file. Procedures pulse_width, amplitude and motorvoltage call the procedure inputY_versus_crankposition which allows the user to define the respective data as a function of crank position. After all the data is entered the procedure edit_oldfile is called to allow for corrections to be made on the new patient file. Edit_oldfile is immediately called by inputStimulationParameters if the file selected for processing already exists (ie. if oldfile). Edit_oldfile calls on request by the operator any of the already mentioned procedures which allow the definition of stimulation parameters or other data. After a file has been edited (ie. edit_oldfile is quitted) the procedure calculate_newfile calculates the look up tables of the new patient file and the control is passed back to the main program FNS-2.

The procedure Stimulate calls on request by the user either procedure impedance_check (which allows the measurement of electrode impedances), start_stimulation or calibrate_impedance_check. If procedure Stimulate is left (ie. quitted by the user) control is transferred back to the main program.

TURBO GRAPHIX PROCEDURES USED IN THE SOFTWARE PACKAGE

The used TURBO GRAPHIX procedures and functions are listed alphabetically below. Please see the TURBO GRAPHIX manual for details.

ClearScreen
ClrEOL
DefineTextWindow
DrawBorder
DrawLine
DrawPolygon
DrawText
GotoXY
InvertWindow
MoveVer
ResetWindows
SelectScreen
SetAspect
SetBreakOn
SetColorWhite
SetLinestyle
SetWindowModeOff
SwapScreen

ClearWindowStack
CopyScreen
DefineWindow
DrawCircle
DrawLinW
DrawSquare
DrawTextW
InitGraphic
LeaveGraphic
ReDefineWindow
ResetWorlds
SelectWindow
SetBackground
SetClippingOn
SetHeaderOff
SetMessageOff
SetWindowModeOn

clip
DefineHeader
DefineWorld
DrawItem
DrawPoint
DrawStraight
GetWindow
InvertScreen
MoveHor
RemoveHeader
RestoreWindow
SelectWorld
SetBreakOff
SetColorBlack
SetHeaderOn
SetMessageOn
StoreWindow

LISTING OF THE PROCEDURES AND FUNCTION IN SEQUENCE OF APPEARANCE IN THE SOURCE CODE :

COMPILED CODE FILE	OVERLAY SIZE	PROCEDURE OR FUNCTION NAME	SOURCE CODE FILE NAME
FNS-2.COM		FNS-2	FNS-2.PAS
FNS-2.COM		DefineWindowIBM	TG-UTIL2.PAS
FNS-2.COM		getkey	TG-UTIL2.PAS
FNS-2.COM		puts	TG-UTIL2.PAS
FNS-2.COM		centre	TG-UTIL2.PAS
FNS-2.COM		wait	TG-UTIL2.PAS
FNS-2.COM		clrKbd	TG-UTIL2.PAS
FNS-2.COM		beep	TG-UTIL2.PAS
FNS-2.COM		readreal	TG-UTIL2.PAS
FNS-2.COM		readint	TG-UTIL2.PAS
FNS-2.COM		readstr	TG-UTIL2.PAS
FNS-2.COM		resetD_A	TG-UTIL2.PAS
FNS-2.COM		commandLine	TG-UTIL2.PAS
FNS-2.COM		DefaultDrive	TG-UTIL2.PAS
FNS-2.COM		log_drive	TG-UTIL2.PAS
FNS-2.COM		change_drive	TG-UTIL2.PAS
FNS-2.COM		DisplayDirectory	TG-UTIL2.PAS
FNS-2.COM		IOCheck	TG-UTIL2.PAS
FNS-2.COM		displayHelp	TG-UTIL2.PAS

```
FNS-2.COM            display_patientfile ..... TG-UTIL2.PAS
FNS-2.COM            initialize_window_patientfile . TG-UTIL2.PAS
FNS-2.COM            draw_cross ..... TG-UTIL2.PAS
FNS-2.COM            place_cross ..... TG-UTIL2.PAS
FNS-2.COM            delete_cross ..... TG-UTIL2.PAS
FNS-2.COM            draw_interpolation ..... TG-UTIL2.PAS
FNS-2.COM            xy_axis_360 ..... TG-UTIL2.PAS
FNS-2.COM            flash ..... TG-UTIL2.PAS
FNS-2.COM            arrowLeft ..... TG-UTIL2.PAS
FNS-2.COM            arrowRight ..... TG-UTIL2.PAS
FNS-2.COM            arrowUp ..... TG-UTIL2.PAS
FNS-2.COM            arrowDown ..... TG-UTIL2.PAS
FNS-2.COM            arrowLeftFast ..... TG-UTIL2.PAS
FNS-2.COM            arrowRightFast ..... TG-UTIL2.PAS
FNS-2.COM            arrowUpFast ..... TG-UTIL2.PAS
FNS-2.COM            arrowDownFast ..... TG-UTIL2.PAS
FNS-2.COM            inputY_versus_crankposition ... TG-UTIL2.PAS
FNS-2.000    1313    intro ..... INPUTS.OVL
FNS-2.000    3925    updateWindow_patientfile ..... INPUTS.OVL
FNS-2.000     963    Getdir ..... INPUTS.OVL
FNS-2.000     263    store_patient ..... INPUTS.OVL
FNS-2.000    1289    defWindows ..... INPUTS.OVL
FNS-2.000    2620    waveform ..... INPUTS.OVL
FNS-2.000              monophasic_pulse ..... INPUTS.OVL
FNS-2.000              biphasic_pulse ..... INPUTS.OVL
FNS-2.000     489    frequency ..... INPUTS.OVL
FNS-2.000    2834    pulse_width ..... INPUTS.OVL
FNS-2.000              constant_pulsewidth ..... INPUTS.OVL
FNS-2.000              editPulseWidth_Onechannel ... INPUTS.OVL
FNS-2.000    2107    amplitude ..... INPUTS.OVL
FNS-2.000              constant_amplitude ..... INPUTS.OVL
FNS-2.000              editAmplitude_Onechannel .... INPUTS.OVL
FNS-2.000     182    motorvoltage ..... INPUTS.OVL
FNS-2.000    1501    namefile ..... INPUTS.OVL
FNS-2.000     950    calculate_newfile ..... INPUTS.OVL
FNS-2.000    2223    calibrate_impedance_check ..... INPUTS.OVL
FNS-2.000              stim_output_voltage ..... INPUTS.OVL
FNS-2.000    1587    impedance_check ..... INPUTS.OVL
FNS-2.000              stim_output_voltage ..... INPUTS.OVL
FNS-2.000              TimerInitialisation ..... INPUTS.OVL
FNS-2.000    8937    start_stimulation ..... STIMULAT.OVL
FNS-2.000              ADSAMPLE ..... STIMULAT.OVL
FNS-2.000              TimerInitialisation ..... STIMULAT.OVL
FNS-2.000              settimers ..... STIMULAT.OVL
FNS-2.COM            Stimulate ..... EDITFILE.PAS
FNS-2.COM            edit_oldfile ..... EDITFILE.PAS
FNS-2.COM            inputStimulationParameters .... EDITFILE.PAS
```

SORTED RESULTS OF OVERLAY ANALYSIS

OVERLAY GROUP	SEGMENT SIZE(bytes)	CONTROLLING PROCEDURE or FUNCTION	SOURCE CODE FILE NAME
.000	182	motorvoltage	INPUTS.OVL
.000	263	store_patient	INPUTS.OVL
.000	489	frequency	INPUTS.OVL
.000	950	calculate_newfile	INPUTS.OVL

```
.000        963     Getdir ..................... INPUTS.OVL
.000       1289     defWindows ................. INPUTS.OVL
.000       1313     intro ...................... INPUTS.OVL
.000       1501     namefile ................... INPUTS.OVL
.000       1587     impedance_check ............ INPUTS.OVL
.000       2107     amplitude .................. INPUTS.OVL
.000       2223     calibrate_impedance_check .. INPUTS.OVL
.000       2620     waveform ................... INPUTS.OVL
.000       2834     pulse_width ................ INPUTS.OVL
.000       3925     updateWindow_patientfile ... INPUTS.OVL
.000       8937     start_stimulation .......... STIMULAT.OVL
```

SEGMENT SIZE LIMIT FOR A SINGLE OVERLAY PROCEDURE IS 8960 BYTES

******* Calculated file sizes ********

FILE NAME	FILE SIZE	DATE	TIME
FNS-2.COM	54409	86/07/10	00:02
FNS-2.000	33280	86/07/10	00:01

The sizes of the overlay porcedures were determined with the utility program OVERMAP which is part of the software package TURBO EXTENDER by TurboPower Software. The overlay analysis shows that OVERLAY PROCEDURE start_stimulation is already nearly of the maximum segment size which is 8960 Bytes. Thus future extensions of the procedure start_stimulation make a rearrangement of the overlay structure necessary. However it is recommended to consider the use of a utility program which allows the creation of TURBO PASCAL programs greater than 64 Kbyte code without the use of overlays. BIGTURBO of TurboPower Software is an example of such an utility package.

LISTING OF THE PROCEDURES AND FUNCTIONS OF FNS-2.PAS

The procedures and functions are listed alphabetically. The 'value parameters' and 'variable parameters' of each procedure and function are defined. The given page number is reffering to the annotated listing.

THE ANNOTATED LISTING OF THE SOURCE CODE OF THE FNS-CONTROLLER SOFTWARE PACKAGE FNS-2.PAS (see overleaf)

```
PROGRAM FNS_2;

  (*****************************************************************
   *                                                               *
   *       FUNCTIONAL NEUROMUSCULAR STIMULATION CONTROLLER         *
   *                                                               *
   *              FOR  THE  UCT  P A R A C Y C L E                 *
   *                                                               *
   *            Department of Biomedical Engineering               *
   *                 University of Cape Town                       *
   *                   UCT-Medical School                          *
   *                     7925 Observatory                          *
   *                  Republic of South Africa                     *
   *                                                               *
   *                                                               *
   *                Version  2 , July 7th,  1986                   *
   *                                                               *
   *                created by Matthias H. Popp                    *
   *                   on a Portable IBM PC                        *
   *                                                               *
   *             written in TURBO PASCAL Version 3.01A             *
   *           and TURBO GRAPHIXS Toolbox Version 1.05A            *
   *                                                               *
   *****************************************************************)

{$C-}


-- Include file A:TYPEDEF.SYS not found --
-- Include file A:GRAPHIX.SYS not found --
-- Include file A:KERNEL.SYS not found --
-- Include file A:WINDOWS.SYS not found --
-- Include file A:POLYGON.HGH not found --


CONST
  BaADGlb       = $700;          {base address of the A/D board}
  BaShftEncGlb = $77C;    {base address of the shaft encoder interface}
                          {N O T E   that in the procedure start_
                           stimulation  the ports are addressed
                           immediate to gain speed !!  }

  BaGlb      = $8C0;             {global board base address of stim board}

  Tclk2Glb   = 214.675052;       {global pulsewidth for timer 2 of 6840}

  pwGlb      = 6.7085952;        {global pulsewidth  of timer 1,3 of 6840}

  max_chGlb = 6;               {global maximal number of channels the FNS-2
                                program is designed for ie. 3 channels per leg}

  chGlb      = 3;                {global number channels per leg}

  xCircleGlb = 120;              {x-coordinate crank circle in procedure
                                 defWind.ovl and start_stimulation}
  yCircleGlb = 70;               {y-coordinate crank circle}
  radiusGlb  = 80;               {radius of crank circle}

  IOValGlb   : INTEGER = 0;        {global I/O error const}
  IOErrGlb   : BOOLEAN = FALSE;    {global error flag, see procedure IOCheck}

  flag_first_fileGlb : BOOLEAN = TRUE;  {global flag which is true as long
                                         as the first patient file is pro-
                                         cessed}
```

```
TYPE
  RegRec     - RECORD              {register record for routines using MS-DOS}
                                   {function calls, see eg. procedure Getdir}
                 AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : INTEGER;
               END;

  string_80 - STRING[80];
  string_20 - STRING[20];
  string_12 - STRING[12];
  string_8  - STRING[8];
  string_5  - STRING[5];
  char12arr - ARRAY [1..12] OF CHAR; {type for directory mask dir_maskGlb}
                                     {see Procedure getdir }

  array_zero  - ARRAY [1..64,1..2] OF INTEGER;
  array_one   - ARRAY [BYTE] OF BYTE;
  array_two   - ARRAY [1..max_chGlb,BYTE] OF BYTE;
  array_three - ARRAY [1..max_chGlb] OF INTEGER;
  array_four  - ARRAY [1..7] OF array_zero;
  array_five  - ARRAY [0..255] OF BOOLEAN;

  patient_record - RECORD
                     name : string_20;
                 initials : string_5;
                birth_day : string_8;
              injury_date : string_8;
             lesion_level : string_5;
                   ch_max : INTEGER;      {number of stimulated channels,
                                           ie. number of channels per leg
                                           times 2 ! }
    interpulse_interval : INTEGER;        {time interval between the
                                           negative going and the positive
                                           going pulse in micro-seconds}
          neg_pos_ratio : REAL;           {ratio of negative pulse width
                                           divided by positive pulse width}
        pulse_frequency : INTEGER;        {pulse repetition frequency in Hz}
         max_pulseWidth : INTEGER;        {max. pulse-width neg. pulse}
      max_motor_power : INTEGER;          {max. motor drive and break power
                                           this variable is not yet used}
            rpm_limit : INTEGER;          {not used; for future versions}
           flag_drive : BOOLEAN;          {TRUE for motor drives patient
                                           not used; for future versions}
      flag_amp_control : BOOLEAN;         {TRUE if pulse amplitude modula-
                                           tion is used for varying the
                                           stimulation level in respect
                                           to crank position}
      flag_pwm_control : BOOLEAN;         {TRUE if pulse width modulation
                                           is used for varying the stimula-
                                           tion level in respect to crank
                                           position}
         flag_biphasic : BOOLEAN;         {TRUE if biphasic pulses are used
                                           as stimulation waveform; FALSE
                                           for monophasic waveforms}

      set_motor_current,     {not used ; for future program versions}
      set_motor_voltage,     {contains motor voltage vs. crank position}
      set_motor_power,       {contains motor power vs. crank position}
      pedal_force_0,         {not used ; for future program versions}
      pedal_force_1,         {not used ; for future program versions}
      pedal_force_2,         {not used ; for future program versions}
      pedal_force_3,         {not used ; for future program versions}
      dummy1                 {not used ; for future program versions}
      : array_one;           {one dimensional array of 256 x one Byte }
```

```
        msb_t1,          {most significant data byte timer 1 of 6840 chip}
        msb_t2,          {most significant data byte timer 2 of 6840 chip}
        msb_t3,          {most significant data byte timer 3 of 6840 chip}
        lsb_t1,          {least significant data byte timer 1 of 6840 chip}
        lsb_t2,          {least significant data byte timer 2 of 6840 chip}
        lsb_t3,          {least significant data byte timer 3 of 6840 chip}
        pos_amplitude,      {positive pulse amplitude}
        neg_amplitude,      {negative pulse amplitude}
        pulsewidth       {pulse width of negative going stimulation pulse}
        : array_two;     {two dimensional array : 6 channels of 256 byte
                          values}

        stored_plotarray : array_four; {for storage of the amplitude or
                                        pulsewidth envelope for use by the
                                        screen editing procedure inputxy_
                                        versus_crankpostion}

        brake_drive : array_five;      {one dimensional array of 255
                                        boolean values : FALSE = brake,
                                        TRUE = drive patient}

    END;  {of patient_record}

VAR
          patientfile : FILE OF patient_record;
              patient : patient_record;
              RegsGlb : RegRec;    {global register record for MS-DOS calls}
           dir_maskGlb : char12arr;  {global directory mask for directory of
                                      the drive containing the patient files}
      logged_driveGlb : string_5;    {global variable defining the drive for
                                      storage of patient files; eg. 'b:'}
      system_driveGlb : CHAR;        {global variable defining the drive
                                      which contains the program system disk
                                      eg.  'a' (without the colon !) }
         file_nameGlb : string_20;   {global variable defining the current
                                      selected patient file, for example :
                                      'b:temmers.FNS'  }
      flag_newfileGlb : BOOLEAN;     {TRUE during entering the data of a
                                      file which was not found on the data
                                      disk containing the patient files}
           flag_XOGlb : BOOLEAN;     {global flag, TRUE indicates that a
                                      stimulation envelope (pulse width or
                                      amplitude has been defined at crank
                                      position zero degree }
                   ch : CHAR;        {global character variable; NOTE that
                                      this global variable does not carry
                                      the extender Glb which characterizes
                                      global variables.}
            chHalfGlb : INTEGER;     {global number of stimulated channels
                                      per leg}
                  a,c : PlotArray;   {global plotarray for proc. polygon}
      xCirclePointGlb : array_one;   {global crank circle array; see }
      yCirclePointGlb : array_one;   {proc. start_stimulation, defwind.ovl}
      oldCH1_Glb,
      oldCH2_Glb,  {global variables containing the actual pulse width of}
      oldCH3_Glb,  {each active channel during real time stimulation, see}
      oldCH4_Glb,  {procedure start_stimulation and procedure settimers  }
      oldCH5_Glb,
      oldCH6_Glb   : BYTE;

{-------------------------------------------------------------------}
```

Listing of FNS-2.PAS-include file TG-UTIL2.PAS

```
{$I tg-util2.pas}      {Turbo graphixs and other utilities}

      {*****************************************************
       *                                                   *
       *          Turbo Graphix Utilities Module           *
       *                                                   *
       *          For Paracycle Software Package           *
       *                                                   *
       *                                                   *
       *                by  M.H. Popp   7/86               *
       *                                                   *
       *****************************************************}


PROCEDURE DefineWindowIBM(i,X1,Y1,X2,Y2:integer);
BEGIN
  DefineWindow(i,Trunc(X1/79*XMaxGlb+0.001),Trunc(Y1/199*YMaxGlb+0.001),
                 Trunc(X2/79*XMaxGlb+0.5),Trunc(Y2/199*YMaxGlb+0.5));
END;

{*******************************************************************}

PROCEDURE getkey (VAR key : CHAR);

{THIS Procedure NEEDS THE COMPILER COMMMAND $C- IN THE MAIN-PROGRAM !
 Procedure getkey returns a character if a key is pressed, else it
 returns a null character}

BEGIN
  IF KeyPressed THEN READ (kbd,key)
    ELSE key := chr(0);
END;

{*******************************************************************}

PROCEDURE puts (x,y:INTEGER;  str:string_80);
                          {puts displays a string at the x,y co-
                           ordinates on the screen, 1,1 defines the
                           top left corner, 79,25 the bottom right
                           corner of the screen}
BEGIN                     {string_80 must have been declared of TYPE}
  GOTOXY (x,y);           {STRING [80] in the main program}
  WRITE(str);
END;

{*******************************************************************}

PROCEDURE centre(y :INTEGER; str :STRING_80);

VAR  i : INTEGER;                              {centre displays a string}
                                               {at the centre of line y }
BEGIN
   i := ROUND(40 - Length(str) DIV 2);
   IF i > 0 THEN
      BEGIN
      GotoXY(i,y);
      Write(str);
      END;
END; {centre}

{*******************************************************************}

PROCEDURE wait;              {wait waits until a key is pressed and
                              clears then the keyboard-buffer       }
   BEGIN
```

```
    REPEAT UNTIL KeyPressed;
    REPEAT
      READ(kbd,ch);
    UNTIL NOT KeyPressed;
  END;

{*********************************************************************}

PROCEDURE clrKbd;      {procedure to clear the input buffer}

BEGIN
  WHILE KeyPressed READ (kbd,ch);
END;

{*********************************************************************}

PROCEDURE beep;        {beep beeps !}

BEGIN
   Sound(880);
   Delay(100);
   Nosound;
   Sound(1109);
   Delay(100);
   Nosound;
END; {beep}

{*********************************************************************}

FUNCTION readreal: REAL;

{ function for entering reals with up to max
significant digits. Additionally the flashing cursor _  is created,
as the Turbo Graphix Toolbox does noramlly not display an flashing
cursor during reading a variable ! }

VAR
   J,value,max,K,l:INTEGER;
   I:REAL;
   x:CHAR;
   strng : string_20;
   cr,OK,point:BOOLEAN;

BEGIN
   strng:='          ';
   cr:=FALSE;
   J:=1;
   max:=8;       {max is the max number of significant digits}
   point:=FALSE;
   x := ^H;          {set character x to backspace}
   REPEAT
     ok := FALSE;
     REPEAT
       Write ('_',x);
       l := 0;
       REPEAT
         getkey(ch);
         IF ORD(ch) IN [8,13,46,48..57] THEN OK:=TRUE
            ELSE IF ORD(ch)<>0 THEN BEEP;
         l := l + 1;
       UNTIL OK OR (l>150);
       Write (' ',x);
       IF NOT ok THEN
         BEGIN
           l := 0;
           REPEAT
```

```
            getkey(ch);
            IF ORD(ch) IN [8,13,46,48..57] THEN OK:=TRUE
              ELSE IF ORD(ch)<>0 THEN BEEP;
            i := i + 1;
          UNTIL OK OR (i>150);
        END;
      UNTIL ok;

      CASE ORD(ch) OF
         46,48..57 : BEGIN
                       IF (J<max) AND ( (NOT(point) OR (ch<>'.'))) THEN
                         BEGIN
                           WRITE (ch);
                           IF ch='.' THEN
                           BEGIN
                              point:=TRUE;
                              max:=max+2;
                           END;
                           strng[J]:=ch;
                           J:=J+1;
                         END ELSE BEEP;
                     END;
                  13 : BEGIN
                         VAL(strng,I,K);
                         cr:=TRUE;
                         readreal := i;
                       END;
                   8 : BEGIN
                         IF J>1 THEN
                         BEGIN
                           IF strng[J-1] = '.' THEN point:=FALSE;
                           WRITE (ch);     {one backspace}
                           WRITE (' ');    {wipe out old character}
                           WRITE (ch);     {one backspace}
                           strng[J-1]:=' ';
                         END;
                         J:=J-1;
                         IF J>0 THEN
                         BEGIN
                           IF strng[J]='.' THEN point:=FALSE;
                           strng[J]:=' ';
                         END ELSE
                         BEGIN
                           J:=1;
                           BEEP;
                         END;
                       END;

        END; (* case *)
    UNTIL cr;
END; (* readreal *)

{*****************************************************************}

FUNCTION readint (max:INTEGER):INTEGER;

{ function for entering positive integers with maximum
max digits. Additinally the flashing cursor is displayed }

VAR
    J,value,K,i:INTEGER;
    i : REAL;
    x:CHAR;
    strng : string_20;
    cr,OK:BOOLEAN;
```

```
BEGIN
   x := ^H;          (set character x to backspace)
   IF max > 5 THEN max := 5;
   strng:='          ';
   cr:=FALSE;
   OK:=FALSE;
   J:=1;
   (max:=5;)
   REPEAT
     ok := FALSE;
     REPEAT
       Write ('_',x);
       l := 0;
       REPEAT
         getkey(ch);
         IF ORD(ch) IN [8,13,48..57] THEN OK:=TRUE
            ELSE IF ORD(ch)<>0 THEN BEEP;
         l := l + 1;
       UNTIL OK OR (l>150);
       Write (' ',x);
       IF NOT ok THEN
         BEGIN
           l := 0;
           REPEAT
             getkey(ch);
             IF ORD(ch) IN [8,13,48..57] THEN OK:=TRUE
               ELSE IF ORD(ch)<>0 THEN BEEP;
             l := l + 1;
           UNTIL OK OR (l>150);
         END;
     UNTIL ok;

     CASE ORD(ch) OF
         48..57 : BEGIN
                    IF J<max+1 THEN
                      BEGIN
                        WRITE (ch);
                        strng[J]:=ch;
                        J:=J+1;
                      END ELSE BEEP;
                  END;
         13 : BEGIN
                VAL(strng,I,K);
                cr:=TRUE;
                IF i > maxint THEN i := maxint;
                readint := ROUND (i);
              END;
         8 : BEGIN
               IF J>1 THEN
                 BEGIN
                   WRITE (ch);
                   WRITE (' ');
                   WRITE (ch);
                   strng[J-1]:=' ';
                 END;
               J:=J-1;
               IF J<1 THEN
                 BEGIN
                   J:=1;
                   BEEP;
                 END;
             END;
       END; (* case *)
   UNTIL cr;
END; (* readint *)
```

Listing of FNS-2.PAS-include file IG-UTIL2.PAS

```
{*********************************************************************}

FUNCTION readstr (max:INTEGER):STRING_20;

{ function for entering string with max characters;
  Additinally the flashing cursor is displayed }

VAR
   J,value,K,l:INTEGER;
   i : REAL;
   x:CHAR;
   strng : string_20;
   cr,OK:BOOLEAN;

BEGIN
   x := ^H;          {set character x to backspace}
   IF max > 20 THEN max := 20;
   strng:='                              '; {presetting strng with 20 spaces, if
                                              readstr should read longer strings
                                             than 20, strng must be preset accordingly}
   cr:=FALSE;
   OK:=FALSE;
   J:=1;
   REPEAT
     ok := FALSE;
     REPEAT
       Write ('_',x);
       l := 0;
       REPEAT
         getkey(ch);
         IF ORD(ch) IN [8,13,27,32,45,46,48..57,65..90,97..122] THEN OK:=TRU
            ELSE IF ORD(ch)<>0 THEN BEEP;
         l := l + 1;
       UNTIL OK OR (l>60);
       Write (' ',x);
       IF NOT ok THEN
         BEGIN
           l := 0;
           REPEAT
             getkey(ch);
             IF ORD(ch) IN [8,13,27,32,45,46,48..57,65..90,97..122] THEN OK
               ELSE IF ORD(ch)<>0 THEN BEEP;
             l := l + 1;
           UNTIL OK OR (l>100);
         END;
     UNTIL ok;

     CASE ORD(ch) OF
         32,45,46,48..57,65..90,97..122 :
                     BEGIN
                     IF J<max+1 THEN
                       BEGIN
                         WRITE (ch);
                         strng[J]:=ch;
                         J:=J+1;
                       END ELSE BEEP;
                     END;
         27 : BEGIN
                REPEAT
                  beep;
                  READ (kbd,ch);
                UNTIL NOT Keypressed;
              END;
         13 : BEGIN
                readstr := Copy (strng,1,J);
                cr:=TRUE;
```

```
            END;
          8 : BEGIN
                IF J>1 THEN
                  BEGIN
                    WRITE (ch);
                    WRITE (' ');
                    WRITE (ch);
                    strng[J-1]:=' ';
                  END;
                J:=J-1;
                IF J<1 THEN
                  BEGIN
                    J:=1;
                    BEEP;
                  END;
              END;
        END; (* case *)
    UNTIL cr;
END; (* readstr *)

{*****************************************************************}

PROCEDURE resetD_A;      {this procedure resets all the D/A's of
                          the waveform generation board in order
                          to stop stimulation immediately; NOTE that
                          the D/A controlling the motor must be set
                          to 254 as this sets the motor voltage to 0}
VAR  i,p: INTEGER;

BEGIN
  p := BaG1b + $30;
  FOR i := 0 TO 11 DO
    BEGIN
      Port [p+i] := 0; {set  Pulseamplitude to 0 V}
    END;
  Port [p+$0C] := 254;  {set motor voltage to 0 V}
  Port [p+$0D] := 0;    {set brake mode}
END;

{*****************************************************************}

PROCEDURE commandLine(i :INTEGER);    {displays a message in window 15
                                       on the bottom of the screen; the
                                       message tells the user what to
                                       do next !}
VAR    temp :INTEGER;

  BEGIN
  IF i  IN  [0..15] THEN
    BEGIN
    temp := GetWindow;      {store active window in temp}
    SelectWindow(15);
    SetBackground(0);       {clear window}
    CASE i OF
0:centre(25,'F1 - HELP          F2 - DIRECTORY       Q - QUIT');
1:centre(25,'Press  any  key  to  switch  DIRECTORY  off');
2:centre(25,'Press  any  key  to  switch  HELP  off');
3:centre(25,'F1 - HELP      F2 - Display PATIENT-FILE     Q - QUIT');
4:centre(25,'Press  any  key  to  switch  PATIENT-FILE  off');
5:centre(25,
  'F1 - HELP      F2 - PATIENT-FILE      F9 - DRAW LINE    Q - QUIT');
6:centre(25,'press  --->  Q   to  QUIT');
7:centre(25,' First define point at zero degrees !!');
8:centre(25,'INPUT  DATA   ,   PRESS  RETURN  FOR DEFAULT   !');
9:centre(25,
  'duplicate this envelope 180 degrees shifted to the right leg Y/N ?');
```

```
10:centre(25,
  'F1 - HELP      F2 - DIRECTORY      Q - QUIT     RETURN - DEFAULT');
11:centre(25,
  'PRESS --->  Q  to  QUIT          SPACEBAR  TO  REPEAT  ENTRY');
12:centre(25,'Input Patient file name !');
13:centre(25,'Read the message and wait 3 seconds  !');
     END; {case}
    InvertWindow;
    SelectWindow(temp);  {restore old active window}
  END;  {if}
END;  {procedure commandLine}

{*****************************************************************}

FUNCTION DefaultDrive : char;  {makes MSDos call to find out the
                                logged drive, it returns the drive
                                character (in lower case; see
                                TURBO TUTOR p. 20-12)}

BEGIN
  WITH RegsGlb DO
    BEGIN
      AX := $1900;
      MSDos(RegsGlb);
      DefaultDrive := chr((AX AND $FF)+$61);
    END;
END;

{*****************************************************************}

PROCEDURE log_drive (driveCharacter:CHAR);

BEGIN
  WITH RegsGlb DO              {MsDOS function call to select new drive}
    BEGIN                      {AX = $0E00 selects the MsDOS function}
      AX := $0E00;             {the drive to select is specified in DL}
      ch := upcase(driveCharacter); {DL=0 is drive a, DL=1 is drive b,..}
      DX := ord(ch) - $41;
      MsDOS (RegsGlb);         {see DOS reference manual of the SPERRY PC}
    END;
END;

{*****************************************************************}

PROCEDURE change_drive;     {procedure to change the drive for
                             storage of the patient files}
BEGIN
  DefineHeader (2,'Change  Logged  Drive');
  SelectWindow (2);
  SetBackGround (0);
  DrawBorder;
  commandLine(8);
  puts (20,8,'CHANGE LOGGED DRIVE FOR STORAGE OF PATIENT DATA');
  puts (20,12,'Current logged drive is : ');
  Write (logged_driveGlb);
  puts (20,14,'New   logged  drive  is : ');
  REPEAT
    READ (kbd,ch);
  UNTIL upcase(ch) IN ['A'..'C'];
  logged_driveGlb := ch + ':';
  file_nameGlb := logged_driveGlb + Copy(file_nameGlb,3,12);
END;

{*****************************************************************}

PROCEDURE DisplayDirectory (deltax,deltay:INTEGER);  {displays the
```

```
                                                    directory of
                                the current logged data drive which has
                                been stored in window 16 by procedure
                                Getdir}
    BEGIN
    CopyScreen;
    RestoreWindow(16,deltax,deltay);
    commandLine(1);
    wait;
    SwapScreen;
    SelectScreen(2);
    ClearScreen;
    SelectScreen(1);
    END;

{*********************************************************************}

PROCEDURE IOCheck;

            {This routine sets IOValGlB equal to IOresult, then sets
             IOErrGlb accordingly.  It also prints out a message on
             the 24th line of the screen and waits one second }
var
  Ch                        : Char;  {local character variable; has nothing
                                       to do with the global variable ch !}
begin
  IOValGlB := IOresult;
  IOErrGlb := (IOValGlB <> 0);
  GotoXY(30,23);
  if IOErrGlb then begin
     Write(Chr(7));
    case IOValGlB of
      $01   :  Write('File does not exist');
      $FO   :  Write
               ('Write disk error --> Disk is full ');
    else  Write('Unknown I/O error:  ',IOValGlB:3)
    end;
    beep;
    delay(1000);
    GotoXY(30,23);
    Write('                                                        ')
  end; {IF}
end; { of proc IOCheck }

{*********************************************************************}

PROCEDURE displayHelp(i:INTEGER);     {displays HELP number i in a
                                       text Window 14 lines by 40
                                       characters in chunks of 10
          lines; after 10 lines are displayed it waits until the
          user presses a key to display the next 10 help lines.
          Help files must have the file extender .HLP ! Help files
          can be edited with the Turbo Pascal editor, or as Word-
          star non document files. Most convenient way to edit Help
          files is to load Sidekick before running this program and
          to use the Sidekick Notepad for editing. The maximum numbers
          of characters per line is 40 !!!  There is no limitation
          to file length }

VAR  oldwindow,y : INTEGER;
     filename : string_20;
     hfile : text;
     help_line : STRING [42];

BEGIN
  IF i IN [0..10] THEN
```

```
    BEGIN
    y := 6;
    oldwindow := GetWindow;
    IF i<>4 THEN CopyScreen;
    DefineTextWindow(14,2,4,42,18,6);
    DefineHeader(14,'H E L P   W I N D O W');
    SelectWindow(14);       {Window defined in defWindows for HELP}
    SetHeaderOn;
    SetBackGround(0);
    DrawBorder;

    CASE i OF
      1 : filename := 'Main.hlp';
      2 : filename := 'Inp-stim.hlp';
      3 : filename := 'waveform.hlp';
      4 : filename := 'envelope.hlp';
      5 : filename := 'ampl.hlp';
      6 : filename := 'p-width.hlp';
      7 : filename := 'stim.hlp';
    END;
    filename := system_driveGlb + ':' + filename;
    ASSIGN (hfile,filename);
    {$I-}
    RESET (hfile);
    {$I+}
    IF IOresult = 0 THEN
      BEGIN
        WHILE NOT (EOF(hfile)) DO
          BEGIN
            IF y = 17 THEN
              BEGIN
                puts (3,18,'..... Press any key for more help !');
                wait;
                SetBackGround(0);
                DrawBorder;
                y := 6;
              END
            ELSE
              BEGIN
                READLN (hfile,help_line);
                puts (3,y,help_line);
                y := y + 1;
              END;
          END;  {while}
        CLOSE (hfile);
        puts (3,18,'Press any Key to switch Help off !');
        commandLine(2);
        wait;
      END  {IF}
     ELSE
      BEGIN
        puts (5,10,'Help File  ');
        Write (filename);
        puts (15,12,'MISSING !');
        beep;
        delay (3000);
      END;  {if & else}
    SwapScreen;
    CopyScreen;
    SelectWindow(oldwindow);    {select window which was active}
  END; {IF}
END; {displayHelp}

{*****************************************************************}

PROCEDURE  display_patientfile;
```

```
                        {displays the patient file which has been
                         stored in window 5 by procedures initialize_
                         window_patientfile (see below) and procedure
                         updateWindow_patientfile }
  BEGIN
    CopyScreen;                {save current screen}
    RestoreWindow(5,0,0);      {restore patientfile window}
    commandLine(4);
    wait;
    SwapScreen;                {restore old screen}
  END;

{*****************************************************************}

PROCEDURE  initialize_window_patientfile;     {sets up window 5 as
                                               window for display of
                                               the patient file}
VAR  temp_string : string_80;

  BEGIN
    SelectScreen(2);
    temp_string := 'Patient-file  ' + file_nameGlb;
    DefineHeader (5,temp_string);
    SelectWindow(5);
    SetHeaderOn;
    SetBackGround(0);
    DrawBorder;
    StoreWindow(5);     {store window frame on stack}
    ClearScreen;
    SelectScreen (1);
  END;

{*****************************************************************}

PROCEDURE draw_cross(x,y:INTEGER);     {draws a cross at world}
  BEGIN                                {coordinates x,y }
    DrawLine(x-3,y,x+3,y);
    DrawLine(x,y+2,x,y-3);
  END;

{*****************************************************************}

PROCEDURE place_cross(x,y:INTEGER; VAR flag_XOGlb:BOOLEAN);

              {places a small cross in world 2 world coordinates in
               window 2. x is the x-coordinate of the cursor (arrow)
               ranging from 0 to 64. y ranges from 0 to 127. The arrow
               can be moved horizontally in steps of 8 world coordinate
               points. The zero,zero point of the xy-axises has the
               world coordinates 60,21 . Thus the world coordinates of
               the cross can be calculated as X = 60 + x*8, and
               Y = y + 21 . These world coordinates are stored in the
               global plot-array c which is later used to draw the
               straight line interpolation between the entered data
               points (crosses). See procedure draw_interpolation.
               Window 2 and world 2 were defined in procedure
               xy_axis_360. }

VAR i,xtemp : INTEGER;

BEGIN
  StoreWindow(4);                  {store the moving window (arrow)}
  SelectScreen(2);                 {select virtual screen}
  SelectWorld(2);
  SelectWindow(2);
  IF x = 0 THEN flag_XOGlb := TRUE;      {set the flag_XO if x=0}
```

```
  xtemp := x;
  X := 60 + x*8;
  Y := 21 + y;
  c [xtemp+1,1] := X;
  c [xtemp+1,2] := Y;
  draw_cross(X,Y);
  CopyScreen;
  SelectScreen(1);
  RestoreWindow(4,0,0);
  SelectWindow(4);
END;

{*************************************************************************}

PROCEDURE delete_cross(x,y :INTEGER; VAR flag_XOG1b:BOOLEAN);

VAR                          {deletes a cross at coordinates x,y  }
  xtemp : INTEGER;           {for comment see procedure draw_cross}

BEGIN
  IF x = 0 THEN flag_XOG1b := FALSE;
  StoreWindow(4);
  SelectScreen(2);
  SelectWorld(2);
  SelectWindow(2);
  xtemp := x;
  x := 60 + x*8;
  y := 21 + y;
  c [xtemp+1,1] := 0;
  c [xtemp+1,2] := 0;
  SetColorBlack;
  Draw_cross(x,y);
  SetColorWhite;
  CopyScreen;
  SelectScreen(1);
  RestoreWindow(4,0,0);
  SelectWindow(4);
END;

{*************************************************************************}

PROCEDURE draw_interpolation(channel:INTEGER; VAR j:INTEGER;
                             VAR  done:BOOLEAN; flag_XOG1b:BOOLEAN);

        {draws straight line segments between the data points entered
         in procedure inputY_versus_crankposition. The maximum i=64
         data points are stored in the global array c in the following
         way :     c [i,1] = X world coordinate of the i'th point.
                   c [i,2] = Y world coordinate of the i'th point.
         To draw an polygon through j data points with the TURBO
         GRAPHIX procedure DrawPolygon it is neccessary that all j
         points are entered in position 1 to j in the array a.
         Therefore all the empty positions in array c are found
         (normally only a few data points are placed on the screen;
         empty crierion is that the X coordinate is. c[i,1] = 0) and
         array c is compressed into array a with maximum j entries.
         Thereafter the polygon through data points 1 to j of array a
         is drawn with procedure DrawPolygon and the data points are
         redrawn with draw_cross in window 2. Drawing of the polygon
         takes only place if a data point has been entered at crank
         position zero degree is. flag_XO must be TRUE}

VAR i,k,l,m : INTEGER;
          dt : REAL;
BEGIN
  IF flag_XOG1b = TRUE THEN
```

```
    BEGIN
      a[1,1] := c[1,1];
      a[1,2] := c[1,2];
      j := 2;
      WITH patient DO
        BEGIN
          FOR i := 2 TO 63 DO
            BEGIN
              IF c[i,1] <> 0 THEN
                BEGIN
                  a[j,1] := c[i,1];
                  a[j,2] := c[i,2];
                  j := j + 1;
                END; {if}
            END; {for i}
        END;  {with}
      a[j,1] := 564;        {x world coordinate of the last data point}
      a[j,2] := c[1,2];     {y world coord. first point = y last point}
      StoreWindow(4);
      RestoreWindow(2,0,0);     {restore x,y axis  from window stack,
                                 see Procedure xy_axis                 }

      SelectWorld(3);          {same as world 2 but y axis upside down}
      SelectWindow(2);         {this is necessary for Drawpolygon, see
                                Turbo Graphixs manual p. 146 and readme
                                file of the Turbo graphixs disk !!!}
      SetLineStyle(1);         {draw dotted line}
      DrawPolygon(a,1,j,0,0,0);  {draw line polygon from point 1 to j}
      SetLineStyle(0);           {reset line style to continuous line}
      SelectWorld(2);
      SelectWindow(2);
      FOR i := 1 TO j DO Draw_cross(ROUND(a[i,1]),ROUND(a[i,2]));
      CopyScreen;                {copy drawn screen to virtual screen}
      RestoreWindow(4,0,0);      {restore the moving arrow}
      SelectWindow(4);
      done := TRUE;
    END  {if flag_XOG1b = TRUE}

    ELSE
      BEGIN
        commandLine(7);
        beep;
        delay(2000);
        commandLine(5);
      END; {else & if}

END; {draw_interpolation}

{*************************************************************************}

PROCEDURE xy_axis_360(zero :INTEGER; yincrement : REAL;
                      channel : INTEGER; Header:string_80;
                      yname:string_5);

        {this procedure is called by inputY_versus_crankposition and
         draws the x-y coordinate system and labelling of the screen
         for entering data points versus crank position. Channel is
         the number of the channel for which data is entered. If
         channel=0 then the x-y axis for the motor voltage is drawn.
         yincrement is the labelling increment per tick on the y axis.
         yname is the labelling unit for the y axis. Zero is the y
         value at the left bottom corner of the x-y coordinate
         system. Header is the string written into the Header of
         window 2. }

VAR
```

```
  temp_string,temp_string1    : string_20;
  header_string               : string_80;
  i,j,k                       : INTEGER;


BEGIN
  DefineWindowIBM(2,1,10,79,185);
  DefineWorld (2,0,0,631,165);    {should be (1,0,0,639,175)  !!!!}
                                  {but header occupies 10 y pixels}
                                  {therefore Y2 only 165           }
  DefineWorld(3,0,165,631,0);     {y-flipped world 2 for procedure
                                   drawpolygon}
  DefineWindowIBM(4,9,162,10,166);
  DefineWorld (4,0,0,7,4);
  temp_string  := '';
  temp_string1 := '';

  WITH patient DO
    BEGIN
      IF channel > chGlb THEN     {chGlb is number of stimulated
                                   channels per leg}
        BEGIN
          k:= channel-chGlb;
          temp_string1 := '  Right Leg';
        END
       ELSE
         BEGIN
           k := channel;
           temp_string1 := '  Left Leg';
        END;  {else & if}
    END; {with}
  Str(k:1,temp_string);
  IF channel>0 THEN
    header_string := header + ' Channel ' + temp_string + temp_string1
   ELSE header_string := 'Definition of Motor Voltage';
  DefineHeader(2,header_string);

  SelectWorld(2);
  SelectWindow(2);
  SetHeaderOn;
  SetBackGround(0);
  DrawBorder;
  puts (15,4,header_string);
  DrawLine(56,21,56,158);
  DrawLine(56,158,60,155);     {draw y-axis + arrow}
  DrawLine(56,158,52,155);
  DrawLine(60,18,571,18);

  FOR i := 0 TO 31 DO          {draw x-axis segmentation every 16 point}
    BEGIN
      j := 60 + i*16;
      DrawLine (j,16,j,18);
    END;
  FOR i := 0 TO 7 DO           {draw x-axis segmentation every 64 points}
    BEGIN                      {and draw numbering from 0 to 360 degree}
      j := 60 + i*64;
      k := 45*i;
      DrawLine(j,14,j,18);
      Str(k:3,temp_string);
      DrawTextW(j-10,10,1,temp_string);
    END;
  DrawLine(571,14,571,18);
  DrawTextW(564,10,1,'360 degree');

  FOR i := 0 TO 15 DO          {draw y-axis segmentation every 8 points}
    BEGIN
```

```
    j := 21 + i*8;
    DrawLine(52,j,56,j);
    END;
  FOR i := 0 TO 8 DO              {draw y-axis segmentation every 16 points}
    BEGIN                         {and draw numbering from 0 to 100 mA      }
      j := 21 + i*16;
      k := ROUND(zero + i * yincrement);
      Str(k:4,temp_string);
      temp_string := temp_string + ' '+yname;
      IF i IN [0..7] THEN DrawLine (50,j,56,j);  {draw line for in [0..7]}
      DrawTextW(6,j,1,temp_string);
    END;
  DrawLine(49,148,56,148);        {draw line at x=100 mA or x=400 us}

  StoreWindow(2);                 {store x,y axis & numbering in
                                     window 1 of window stack     }
END; {xy_axis_360}


{*****************************************************************}

PROCEDURE Flash (Count,time:INTEGER);    {flashes the active window}
                                         {called by inputY_versus_
                                    crankposition to flash the little
                                    arrow. Count defines how often the
                                    respective window is flashed ie.
                                    inverted. time is the delay between
                                    two inversions}
VAR i:INTEGER;
BEGIN
   FOR i:= 1 To (Count*2) DO
      BEGIN
      Delay(time);
      InvertWindow;
   END;
END;

{*****************************************************************}

          {The following procedures are used to move the little
           arrow which is drawn in window 4 in two speeds over the
           screen ie. in window 2. The arrow can be moved maximum 63
           position in the horizontal and 127 positions in the
           vertical direction. All the arrow procedures check these
           boundary conditions. For details how to move windows
           please see TURBO GRAPHIXS manual p. 39 ff.
           ALL the arrow procedures are called by procedure inputY_
           versus_crankposition}

PROCEDURE arrowLeft(VAR x:INTEGER);
  BEGIN
    IF x > 0 THEN
      BEGIN
        MoveHor(-1,true); {left arrow}
        x := x - 1;
      END;
  END;

{*****************************************************************}

PROCEDURE arrowRight (VAR x:INTEGER);
  BEGIN
    IF x < 63 THEN
      BEGIN
        MoveHor(1,true);  {right arrow}
        x := x + 1;
```

```
      END;
  END;

{*************************************************************************}

PROCEDURE arrowUp (VAR y:INTEGER);
  BEGIN
    IF y < 127 THEN
      BEGIN
        MoveVer(-1,true); {up arrow}
        y := y + 1;
      END;
  END;

{*************************************************************************}

PROCEDURE arrowDown (VAR y:INTEGER);
  BEGIN
    IF y > 0 THEN
      BEGIN
        MoveVer(1,true);  {down arrow}
        y := y - 1;
      END;
  END;

{*************************************************************************}

PROCEDURE arrowLeftFast (VAR x:INTEGER);
VAR temp:INTEGER;
  BEGIN
    temp := x - 5;
    IF temp<-0 THEN
      BEGIN
        MoveHor(-x,TRUE);
        x := 0;
      END
     ELSE
      BEGIN
        MoveHor(-5,true);
        x := x - 5;
     END;
  END;

{*************************************************************************}

PROCEDURE arrowRightFast (VAR x:INTEGER);
VAR temp:INTEGER;
  BEGIN
    temp := x + 5;
    IF temp>-63 THEN
      BEGIN
        MoveHor(63-x,TRUE);
        x := 63;
      END
     ELSE
      BEGIN
       MoveHor(5,true);
       x := x + 5;
     END;
  END;

{*************************************************************************}

PROCEDURE arrowUpFast (VAR y:INTEGER);
VAR temp:INTEGER;
  BEGIN
```

```
    temp := y + 10;
    IF temp>=127 THEN
      BEGIN
        MoveVer(y-127,TRUE);
        y := 127;
      END
     ELSE
      BEGIN
        MoveVer(-10,true);
        y := y + 10;
      END;
  END;

{*************************************************************************}

PROCEDURE arrowDownFast (VAR y:INTEGER);
VAR temp:INTEGER;
  BEGIN
    temp := y - 10;
    IF temp<=0 THEN
      BEGIN
        MoveVer(y,TRUE);
        y := 0;
      END
     ELSE
      BEGIN
       MoveVer(10,true);
       y := y - 10;
      END;
  END;

{*************************************************************************}

PROCEDURE inputY_versus_crankposition (channel,selection:INTEGER);

  {this procedure lets the user enter stimulation envelopes versus
   crankposition (0..360 degree) for:
   pulsewidth    : selection = 0  called by editPulseWidth_Onechannel
   amplitude     : selection = 1  called by editAmplitude_Onechannel
   motorvoltage : selection = 2  called by motorvoltage

   The envelopes are defined by a entered set of data points which are
   drawn on the screen as small crosses and through which an inter-
   polated curve is drawn with procedure draw_interpolation. To enter
   the data points a small cursor (arrow) which is drawn in window 4
   is moved with the standard cursor keys of the IBM numeric key pad.
   NOTE that the Num Lock key must have been to be pressed to recognize
   the respective key codes !!!!! The little arrow can be moved in
   increments of one Y-screen-pixel and 8 X-screen-pixels on the screen.
   Data points are entered in window 2 which is defined in procedure
   xy_axis_360. World 2 defines a world to window 2 in a way that one
   world coordinate is equal one screen pixel, and the left bottom
   corner of window 2 is equal to the world coordinates 0,0 and the
   right top corner equal to world coordinates 635,165. For drawing
   the interpolation between data points world 3 is selected for window
   2. In world 3 the left bottom corner is 0,165 and the right top
   corner is 635,0 ie. the y-coordinates are upsidedown. This is due
   to the world coordinate system used by TURBO GRAPHIX procedure
   DrawPolygon, see page 146-147 TURBO GRAPHIX manual and the infor-
   mation in the readme file of the TURBO GRAPHIXS disk !
   A x-y coordinate axis system is drawn into window 2. The left
   bottom (zero point) of the x-y axis is world coordinate 60,21 ie.
   Xworld is 60 and Yworld is 21.
       The position of the little arrow is stored in the variables x
   and y. x ranging from 0 to 63 and y from 0 to 127. The world
   coordinates of the arrow tip are calculated in the procedure
```

```
  place_cross with the relationships : Xworld = 60 + x*8 and
  Yworld = 21 + y . If a data point is entered with procedure
  place_cross the respective world coordinates are entered into the
  array c in the following way :
  c [x,1] = X world coordinate of the x'th point = 60 +x*8 ;
  c [x,2] = Y world coordinate of the x'th point = 21 + y  ;
  Positions x of array c in which no data points are entered are set
  to zero ! To remove a data point from the screen the Backspace key
  is used. Procedure delete_cross removes the respective cross from
  window 2 and sets the respective position in array c to 0.
      After all the wanted data points are entered the envelopes are
  interpolated between the data points in the 'compressed array a'
  (see procedure draw_interpolation !) and placed into the respective
  arrays pulsewidth [channel,crankposition], neg_amplitude[channel,
  crankposition] or set_motor_power[crankposition]. See next block of
  comments for details !
      Previously entered data points are stored in the patient file
  in the array stored_plotarray in the same way as they are stored in
  the array a which is used by procedure draw_interpolation to draw
  a straight line interpolation between entered data points. If an
  old patient file is edited ie. flag_newfileGlb = FALSE the old
  data envelope is drawn onto the screen. These old data points can
  then be edited.
      If a new data file is processed (ie. flag_newfile = TRUE) the
  data entered for the left leg is automaticall shifted 180 degrees to
  the right and transferred to the respective channel of the right leg.
  This is possible because the activity patterns of respective leg
  muscle groups are usually 180 out of phase. The user of the software
  can define this 'shift 180 degrees and copy' option as well when
  editing an already existing data file (ie. flag_duplicate = TRUE).}

VAR
     d,x,y,z,i,j,k,l,m,n,temp : INTEGER;
  flag_XOGlb,done,flag_duplicate : BOOLEAN;
     pw_factor,y_incr,max_Byte : REAL;
                   temp_string : string_5;

BEGIN
  WITH patient DO
    BEGIN
      flag_duplicate := False;
      j:=2;     {variable pointing at the last entered point in array a}
      x:=0;     {variable indicating the x position of the arrow 0...63}
      y:=0;     {variable indicating the y position of the arrow 0...127}
      done := FALSE;
      flag_XOGlb := FALSE;
      FillChar(a,SizeOf(a),0);     {resetting arrays a and c}
      FillChar(c,SizeOf(c),0);
      commandLine(5);
      IF selection = 2 THEN channel := 7;

      CASE selection OF
        0 : BEGIN
              y_incr := max_pulseWidth/8;
              xy_axis_360 (0,y_incr,channel,'Define Pulse-width','us');
            END;
        1 : xy_axis_360 (0,12.5,channel,'Define Pulse-amplitude','mA');
        2 : BEGIN
              y_incr := 6;
              xy_axis_360 (-24,y_incr,0,'Define Motor Voltage','V');
              DrawLine (60,85,575,85);     {draw zero line into xy-axis}
              StoreWindow(2);              {and store it onto stack}
            END;
      END;  {case}

      IF flag_newfileGlb=FALSE THEN     {plotting the old envelope}
```

```
  BEGIN
    FOR i := 1 TO 64 DO
      BEGIN
        done := TRUE;
        flag_XOG1b := TRUE;
        c[i,1] := stored_plotarray[channel,i,1];
        c[i,2] := stored_plotarray[channel,i,2];
      END;  {for i}
    draw_interpolation(channel,j,done,flag_XOG1b);
    flag_XOG1b := TRUE;
  END; {if}

CopyScreen;                          {copy it to the virtual screen}

SetBreakOff;                         {dont error when window 4 hits}
SetMessageOff;                       {the edge of the screen }

SelectWorld(4);                      {select it's world}
SelectWindow(4);                     {select moveable window}
SetBackground(0);                    {give it a black background}
SetLineStyle(0);
DrawLine(0,2,7,2);
DrawLine(1,2,4,4);                   {draw leftpointed arrow}
DrawLine(1,2,4,0);

puts (60,4,'  0  degree,   0  ');

CASE selection OF
  0 : Write ('us');
  1 : Write ('mA');
  2 : BEGIN
        puts (60,4,'  0  degree, ');
        Write ('-24  V');
      END;
END;  {case}

done := FALSE;
flash(6,50);
REPEAT
  read(Kbd,Ch);          {read the keystroke}
  case ord(Ch) of
    72 : arrowUp(y);              { '8' }
    75 : arrowLeft(x);            { '4' }
    77 : arrowRight(x);           { '6' }
    80 : arrowDown(y);            { '2' }

    56 : arrowUpFast(y);          { Shift '8' }
    52 : arrowLeftFast(x);        { Shift '4' }
    54 : arrowRightFast(x);       { Shift '6' }
    50 : arrowDownFast(y);        { Shift '2' }

    13 : place_cross(x,y,flag_XOG1b);     { ENTER Key }
     8 : delete_cross(x,y,flag_XOG1b);    { Backspace Key }
    67 : draw_interpolation(channel,j,done,flag_XOG1b);  { F9 Key}
    32 : flash(4,50);                     { Spacebar }
    59 : BEGIN                            { F1 Key }
           StoreWindow(4);       {store moving arrow}
           displayHelp(4);       {display help 4 }
           RestoreWindow (4,0,0); {restore moving arrow}
         END;
    60 : BEGIN                     { F2 Key }
           StoreWindow(4);         {store moving window}
           RestoreWindow(5,0,0);   {displaying patient file}
           CommandLine(4);
           wait;
           SwapScreen;
```

```
            CopyScreen;
            RestoreWindow(4,0,0); {restore moving window}
          END;  {case 60}
     81 : draw_interpolation(channel,j,done,flag_XOG1b);
    113 : draw_interpolation(channel,j,done,flag_XOG1b);

  end; {case}

  GotoXY(60,4);                {display crank position in degrees}
  temp := ROUND (x * 5.625);
  Write (temp:3,'  degree');
  GotoXY(73,4);

  CASE selection OF
    0 : Write ((ROUND((y/127)*max_pulsewidth)):3,'  us');
    1 : Write (ROUND(y*0.787402):3,'  mA');
    2 : Write (ROUND(-24 + y*(24/64)):4,' V');
  END; {case}

until (upcase(Ch)='Q') AND done;

d:=1;
IF (selection<2) AND (channel<=chHalfG1b) AND (flag_newfileG1b=TRUE)
  THEN
  BEGIN
    d := 2;
    flag_duplicate := TRUE;
  END
ELSE IF (selection<2) AND (channel<=chHalfG1b) AND (flag_newfileG1b=
FALSE) THEN
  BEGIN
    commandLine(9);
    beep;
    REPEAT
      READ (kbd,ch);
    UNTIL upcase(ch) IN ['Y','N'];
    commandLine(5);
    IF upcase(ch)='Y' THEN
      BEGIN
        d:=2;
        flag_duplicate := TRUE;
      END;
  END; {else if}

FOR n:=1 TO d DO          {loop is executed twice if duplication}
 BEGIN                    {of just entered channel to the right }
                          {leg is neccessary}
FOR  i := 1 TO j DO
  BEGIN
    a[i,1] := (a[i,1]-60)/2; {recalculate x so that x IN [0..255]}
    a[i,2] := (a[i,2]-21);   {recalcuate y so that y IN [0..127]}
  END;
a[j,1] := 255;     {set x of the last point to 255}

{ negative pulsewidthes are defined in array pulsewidth[channel,
  crankposition] from 0 us to 255*pwG1b (=1710us); pwG1b being
  the increment of the timers 1 and 3 of the 6840 timer chip ie.
  6.7085952 us . thus eg. value 10 in array pulsewidth is equal to
  67 us. In plotarray a the pulsewidthes are defined from 0 - 0us
  to 127 - max_pulsewidth (defined in proc. pulsewidth), ie. one
  y-increment in array a equals max_pulsewidth/127 us; to get
  now the appropriate value in array pulsewidth the following
  calculation has to be performed :
   pulsewidth[channel,crankposition] := a[..]*max_pulsewidth/
   (127*pwG1b); the necessary factor is :
   correction := max_pulsewidth/(127*pwG1b);
```

```
  the pulsewidths are defined from 0 - 0 degree crank position,
  and 255 - 358.4 degree crank position. In procedure calculate
  newfile the arrays msbtx, lsbtx are calculated the other way
  round, ie. 0 - 358.4 deg. and  255 - 0 degrees  !!! This is
  due to the mechanical arrangement of the shaft encoder which
  decreases its output value from 255 to 0 if the crank moves
  clockwise from 0 degree to 358.4 degrees !
      The pulse widths of the negative going pulse (produced by
  timer 3 of th 68840 chip) are equal to the pulse widths defined
  i array pulsewidth. The positive pulse widths (produced by timer
  1 of the 68840 chip) are calculated via the neg_pos ratio (see
  variable defininition in patient record in file FNS-2.PAS).

  amplitudes are defined in array neg_amplitude[channel,crank-
  position] between 0..100mA, 0 - 0mA and 254 - 100mA;
  the array pos_amplitude is calculated from neg_amplitude via
  the neg_pos_ratio; eg. if neg_pos_ratio - 1 then the pos_amp-
  litudes are equal to the neg_amplitudes; if neg_pos_ratio -
  0.5 (ie. the positive pulsewidthes are double the neg_pulse
  width) the pos_amplitudes are half the neg_amplitudes in order
  to ensure charge balance. Thus the pos_amplitude - neg_ampli-
  tude * neg_pos_ratio !
  the array neg_amplitude is defined referring to the crank
  position as  0 - 358.4 degree and 255 - 0 degrees !!!

  motor_power is defined in array a as follows:
  0 is - max_motor_power, 64 is 0, 127 - + max _motor_power;
  (- meaning breaking the patient, + meaning driving the
  patient). array a is expanded to array set_motor_power by
  multiplication with 2 ie.
  0 - - max_motor_power, 128 - 0, 254 - + max_motor_power;
  The array set_motor_voltage is calculated in procedure calculate
  newfile from array set_motor_power which is defined reffering
  to the crank position as 0 - 358.4 degree, and 255 - 0 degrees ! )

pw_factor := max_pulseWidth/(127*pwGlb);
m := 0;                  {m is the pointer (0..255) in the
                          destination array}
FOR i := 1 TO j-1 DO
  BEGIN
    y_incr := (a[i+1,2] - a[i,2])/(a[i+1,1] - a[i,1]);
    l := 0;
    FOR k := ROUND(a[i,1]) TO ROUND(a[i+1,1]-1) DO
      BEGIN
        CASE selection OF
          0 : pulsewidth[channel,m] := ROUND(pw_factor*
                                       (a[i,2]+y_incr*l));
          1 : neg_amplitude[channel,255-m] := 2*ROUND(a[i,2]+y_incr*l:
          2 : set_motor_power[m] := 2*ROUND(a[i,2]+y_incr*l);
        END; {case}
        IF m=128 THEN z:=ROUND(a[i,2]+y_incr*l);
           {store y of midpoint which is y (x=0) of the -180
            degree shifted right leg channel}
        l := l+1;     {increment counter between 2 x-points}
        m := m+1;     {increment destination counter}
      END; {for k}
END;      {for i}

CASE selection OF   {interpolate 255 th. point}
  0 : pulsewidth[channel,m] := ROUND(pw_factor*
                               (a[i,2]+y_incr*l));
  1 : neg_amplitude[channel,255-m] := 2*ROUND(a[i,2]+y_incr*l);
  2 : set_motor_power[m] := 2*ROUND(a[i,2]+y_incr*l);
END; {case}

SelectWorld(2);
```

```
      SelectWindow(2);

      FOR i := 1 TO 63 DO          {storing plotarray c in patient}
        BEGIN
          stored_plotarray[channel,i,1] := ROUND (c[i,1]);
          stored_plotarray[channel,i,2] := ROUND (c[i,2]);
        END;

      IF (n=1) AND (flag_duplicate=True) THEN
        BEGIN
          FillChar (c,SizeOf(c),0);    {reset c array}
          FOR i := 1 TO 32 DO
            BEGIN
              IF stored_plotarray[channel,i,1]<>0 THEN
                BEGIN
                  c [i+32,1] := stored_plotarray [channel,i,1] + 256;
                  {shifts x screen coord. 256 points right}
                  c [i+32,2] := stored_plotarray [channel,i,2];
                END;
            END;
          FOR i := 32 TO 63 DO
            BEGIN
              IF stored_plotarray[channel,i,1]<>0 THEN
                BEGIN
                  c [i-32,1] := stored_plotarray [channel,i,1] - 256 ;
                  {shifts x screen coord. 256 points to the left}
                  c [i-32,2] := stored_plotarray [channel,i,2];
                END;
            END;
          a[1,1] := 60;

          a[1,2] := z + 21;    {z is midpoint of left channel}
          j := 2;              {set destination pointer in a = 2}
          FOR i := 2 TO 63 DO
            BEGIN
              IF c [i,1]<>0 THEN
                BEGIN
                  a[j,1] := c[i,1];
                  a[j,2] := c[i,2];
                  j := j + 1;
                END;
              a[j,1] := 564;       {first point=last point in array a}
              a[j,2] := a[1,2];
              c[1,1] := a[1,1];
              c[1,2] := a[1,2];
            END;
          channel := channel + chGlb;

        END;  {if duplicate}
      END; {for n}
      END; {with}
END; {inputY_versus_crankposition}

{*****************************************************************}
```

```
{$I Inputs.ovl}        {overlay procedures to input stimulation}
       {*********************************************************
        *                                                       *
        *    Overlay Procedures for the definition of           *
        *                                                       *
        *      Stimulation Parameters and Sequences             *
        *                                                       *
        *                                                       *
        *                By  H.M. Popp  7/86                    *
        *                                                       *
        *********************************************************}

{IN_CLUDE FILE Inputs.ovl

NOTE : this file is already 62 KByte long. Only 200 byte are free
for additional source code ! Split this file up if you want to enter
more code ! }

{*****************************************************************}

OVERLAY PROCEDURE intro;    {this is the modified introduction
                             display of the TURBO GRAPHIX demo
                             program tgdemo.pas which is supplied
                             with the TURBO GRAPHIXS disk }
var i:integer;
begin
  SetHeaderOff;
  DefineWorld(2,0,199,639,0);
  DefineWindowIBM(10,0,0,xmaxGlb,ymaxGlb);
  DefineWindowIBM(1,5,40,25,80);
  DefineWindowIBM(2,16,55,43,95);
  DefineWindowIBM(4,15,60,43,135);
  SelectWindow(1);
  DrawBorder;
  SelectWorld(2);
  SelectWindow(10);
  DrawTextW(65,55,5,'PARA ');
  StoreWindow(1);
  delay(100);
  SelectWindow(2);
  SetBackground(0);
  DrawBorder;
  SelectWorld(2);
  SelectWindow(10);
  DrawTextW(140,70,5,' CYCLE ');
  StoreWindow(2);
  delay(100);
  for i:=1 to 4 do
  begin
    RestoreWindow(1,0,0);
    delay(50);
    RestoreWindow(2,0,0);
    delay(50);
  end;
  for i:=1 to 8 do
  begin
    RestoreWindow(1,trunc(3*i*XMaxGlb/79),trunc(9*i*YMaxGlb/199));
    delay(100);
    RestoreWindow(2,trunc(3*i*XMaxGlb/79),trunc(9*i*YMaxGlb/199));
    delay(100);
  end;
  delay(500);
  InvertScreen;
  SelectScreen(2);
  ClearScreen;
```

```
  InvertScreen;
  SetColorBlack;
  SelectWorld(2);
  SelectWindow(10);
  DrawTextW(240,75,8,'UCT');
  DrawTextW(60,140,4,' P A R A  C Y C L E');
  CopyScreen;
  SelectScreen(1);
  delay(500);
  DefineWindowIBM(1,1,125,35,168);
  StoreWindow(1);
  DefineWindowIBM(2,36,125,78,168);
  StoreWindow(2);
  for i:=1 to trunc(34*XMaxGlb/79) do
  begin
    SelectWindow(1);
    MoveHor(1,false);
    ReDefineWindow(1,X1RefGlb,Y1RefGlb,X2RefGlb-1,Y2RefGlb);
    SelectWindow(2);
    MoveHor(-1,false);
    ReDefineWindow(2,X1RefGlb+1,Y1RefGlb,X2RefGlb,Y2RefGlb);
  end;
  delay(300);
  RestoreWindow(1,0,0);
  RestoreWindow(2,0,0);
  GotoXY(1,25);
  ClrEol;
  Centre(25,'Copyright BME UCT (C) 1986,  created by Matthias H. Popp');
  delay(2000);
  SetColorWhite;
  SetHeaderOn;
end;     {overlay procedure intro}

{*****************************************************************}


OVERLAY PROCEDURE  updateWindow_patientfile (selection:INTEGER);

{according to selection the following parameters in the patientfile
 window 5 are updated : selection = 1 --> waveform, constant amplitude
 or pulsewidth, monophasic/biphasic mode, max_rpm;  selection = 2 -->
 variable pulsewidth;  selection = 3 --> variable amplitudes;
 selection = 4 --> motorvoltage;

 This procedure is called by procedures edit_oldfile and
 inputStimulationParameters.}

VAR
          i,j,k,l,m,xplot,yplot,channel :INTEGER;
 temp_string                            : string_20;
 Correction                             : REAL;

BEGIN
  DefineWorld(3,0,0,224,35);
  SelectWindow(6);  {select window 6; subsequent drawing
                     will take place in the window 6, shifted
                     by the Restorewindow offset to its new
                     location}
  WITH patient DO
    BEGIN
      IF  selection=1 THEN
          BEGIN
            RestoreWindow(5,0,0);
            puts (6,6,' PATIENT-FILE');
            puts (4,8,'NAME : '); Write (name);
            puts (4,9,'       '); Write(initials);
```

```
        puts (4,10,'BRTH-D : '); Write (birth_day:8,' ');
        puts (4,11,'INRY-D : '); Write (injury_date:8,' ');
        puts (4,12,'LESION : '); Write (lesion_level,' ');
        puts (4,14,'-/+ RATIO : '); Write (neg_pos_ratio:3:2,' ');
        puts (4,15,'PULSE-F : ');Write (pulse_frequency:3,' Hz ');
      (* puts (4,16,'max-rpm : ');Write (rpm_limit:3,'   '); *)
        IF flag_biphasic THEN
          BEGIN
            puts (4,17,'Biphasic Mode  ');
            puts (4,13,'-/+ TIME : ');
            Write (interpulse_interval:3,' us ');
          END (if)
         ELSE
          BEGIN
            puts (4,17,'Monophasic Mode');
            puts (4,13,'                ');
          END; (else)

        IF flag_pwm_control THEN
        BEGIN
          puts(6,19,' AMPLITUDES');
          FOR i := 1 TO chHalfGlb DO
            BEGIN
              puts (4,20+i,'L');Write(i,'=');
              Write (ROUND(neg_amplitude[i,0]/2.55):3,'mA');
              puts (13,20+i,'R');Write (i,'=');
              Write (ROUND(neg_amplitude[i+chGlb,0]/2.55):3,'mA');
              puts (4,18,'                ');
            END;
        END; (if)

        IF flag_amp_control THEN
          BEGIN
            puts(4,19,' PULSE-WIDTH');
            puts (4,18,'max_pw = ');Write (max_pulsewidth,' us');
            FOR i := 1 TO chHalfGlb DO
              BEGIN
                puts (4,20+i,'L');Write(i,'=');
                Write (ROUND(pulsewidth[i,0]* pwGlb):3,'us ');
                puts (13,20+i,'R');Write (i,'=');
                Write (ROUND(pulsewidth[i+chGlb,0]* pwGlb):3
                                                        ,'us ');
              END;
            END;  (if)

      StoreWindow(5);
    END; (if=1)

  IF selection=2 THEN
    BEGIN
      SelectScreen(2);
      SelectWorld(3);
      RestoreWindow (6,0,0);
      Str (max_pulsewidth:3,temp_string);
      DrawTextW (0,7,1,' 0 ');
      DrawTextW (0,31,1,temp_string);
      i := max_pulsewidth DIV 2;
      Str (i:3,temp_string);
      DrawTextW (0,19,1,temp_string);
      DrawTextW (0,25,1,'us');
      StoreWindow (6);                  (store xy axis & labels)
      SelectScreen(1);
      ClearScreen;
      RestoreWindow(5,0,0);

    (* pulsewidth[channel,crank_position] x pwGlb is pulsewidth
```

```
        in us; in window 6  0 = 0 us  pulsewidth and 24 =
        max_pulsewidth ; so yplot is offset in window 6 (-7)
        + pulsewidth[channel,crankposition] x pwGlb x 24 /
        max_pulsewidth; the necessary correction factor is
        therefore : correction := pwGlb x 24/max_pulsewidth *)

      correction := pwGlb * 24/max_pulseWidth;
      FOR i := 0 TO 1 DO
        BEGIN
          FOR j := 1 TO chHalfGlb DO
            BEGIN
              SelectWorld(3);
              RestoreWindow (6,21+i*29,39+(j-1)*36);
              Str(j:1,temp_string);
              IF i=0 THEN
                 temp_string:='PWM Ch L '+temp_string
                ELSE temp_string := 'PWM Ch R '+temp_string;
              DrawTextW (154,31,1,temp_string);
              l := i*chGlb + j;
              FOR k := 0 TO 192 DO  {xplot max = 192}
                BEGIN
                  yplot := ROUND(7+correction*pulsewidth
                           [l,ROUND(k*1.3281)]);
                  xplot := 24 + K;
                  DrawPoint (xplot,yplot);
                END;  {for k}
            END; {for j}
        END;  {for i}
       StoreWindow (5);

     SelectWorld(3);
     SelectScreen(2);
     RestoreWindow (6,0,0);
     SetColorBlack;
     Str (max_pulsewidth:3,temp_string);
     DrawTextW (0,7,1,' 0 ');
     DrawTextW (0,31,1,temp_string);
     i := max_pulsewidth DIV 2;
     Str (i:3,temp_string);
     DrawTextW (0,19,1,temp_string);
     DrawTextW (0,25,1,'us');
     SetColorWhite;
     StoreWindow (6);                {store xy axis without labels}
     SelectScreen(1);
   END;  {if 2}

IF selection=3 THEN
   BEGIN
     SelectScreen(2);
     SelectWorld(3);
     RestoreWindow (6,0,0);
     DrawTextW (0,7,1,' 0 ');
     DrawTextW (0,19,1,'50 ');
     DrawTextW (0,31,1,'100');
     DrawTextW (0,25,1,'mA');
     StoreWindow(6);                   {store xy axis & labels}
     SelectScreen(1);
     ClearScreen;
     RestoreWindow(5,0,0);
     FOR i := 0 TO 1 DO              {neg_amplitude IN [0..254] ie.}
       BEGIN                         {multiply by 24/254 to fit in }
         FOR j := 1 TO chHalfGlb DO     {plotarea 24 high}
           BEGIN
             SelectWorld(3);
             RestoreWindow (6,21+i*29,39+(j-1)*36);
             Str(j:1,temp_string);
```

```
                IF i = 0 THEN
                   temp_string := 'AMP Ch L '+temp_string
                  ELSE temp_string := 'AMP Ch R '+temp_string;
                DrawTextW (154,31,1,temp_string);
                l := i * chGlb + j;
                FOR k := 0 TO 192 DO     {max xplot = 192}
                  BEGIN
                    yplot:=ROUND(7+neg_amplitude[l,
                                 ROUND(255-k*1.3281)] * 0.09448);
                    xplot := 24 + K;
                    DrawPoint (xplot,yplot);
                  END;  {for k}
              END; {for j}
          END;  {for i}
        StoreWindow(5);

        SelectWorld(3);
        SetColorBlack;
        SelectScreen(2);
        RestoreWindow (6,0,0);
        DrawTextW (0,7,1,' 0 ');
        DrawTextW (0,19,1,'50 ');
        DrawTextW (0,31,1,'100');
        DrawTextW (0,25,1,'mA');
        StoreWindow(6);              {store xy axis without labels}
        SetColorWhite;
        SelectScreen(1);
      END; {if 3}

    IF selection=4 THEN
        BEGIN
          RestoreWindow(5,0,0);
          SelectWorld(3);
          RestoreWindow(6,21,148);
          Str(24:2,temp_string);
          SetLineStyle(1);
          DrawLine (24,19,216,19);
          SetLineStyle(0);
          DrawTextW (0,19,1,'0 V');
          DrawTextW (0,7,1,'-'+temp_string);
          DrawTextW (0,31,1,'+'+temp_string);
          DrawTextW (150,31,1,'Motor Voltage');
          FOR k := 0 TO 192 DO
            BEGIN
              yplot := 7 + ROUND(set_motor_power[ROUND(k*1.3281)]
                                                          *0.09448);
              xplot := 24 + k;
              DrawPoint (xplot,yplot);
            END;
          StoreWindow(5);
        END; {if 4}

  END; {with}

END;   {overlay updateWindow_patientfile}

{*****************************************************************}

OVERLAY PROCEDURE Getdir(mask:Char12arr);

   { This is a procedure to list out the directory of the
     current logged_driveGlb.  The directory mask is transferred
     to this procedure with the variable mask. The found files are
     written into the directory window 16. The size of window 16
     is adjusted to the number of files found. For details of the
     used MS-Dos calls see TURBO PASCAL TUTOR manual page 20-4 to
```

```
    20-5  ! }

type
  String20              = string[ 20 ];

var
  DTA                   : array [ 1..43 ] of Byte;
  NamR                  : String20;
  fault, I,yplot        : Integer;
  files_counter,temp,J  : INTEGER;
  oldwindow             : INTEGER;
  directory_files       : ARRAY[0..63] OF string20;

begin { main body of program DirList }

  {to get the directory of the data disk the logged drive must be
   changed to the data drive; after the directory is read the system
   disk is logged again, to read subsequent overlay files !}

  log_drive (COPY(logged_driveGlb,1,1));

  oldwindow := GetWindow;      {save number of active window}
  FillChar(directory_files,SizeOf(directory_files),0);
  FillChar(DTA,SizeOf(DTA),0);          { Initialize the DTA buffer }
  FillChar(NamR,SizeOf(NamR),0);        { Initialize the file name }

  RegsGlb.AX := $1A00;              { Function used to set the DTA }
  RegsGlb.DS := Seg(DTA);           { store the parameter segment in DS }
  RegsGlb.DX := Ofs(DTA);           {    "     "       "      offset in DX }
  MSDos(RegsGlb);                   { Set DTA location }
  fault := 0;
  RegsGlb.AX := $4E00;               { Get first directory entry }
  RegsGlb.DS := Seg(mask);           { Point to the file Mask }
  RegsGlb.DX := Ofs(mask);
  RegsGlb.CX := 22;                  { Store the option }
  MSDos(RegsGlb);                    { Execute MSDos call }
  fault := RegsGlb.AX and $FF;       { Get fault return }
  I := 1;                          { initialize 'I' to the first element }
  if (fault = 0) then
    repeat
      NamR[I] := Chr(Mem[Seg(DTA):Ofs(DTA)+29+I]);
      I := I + 1;
    until not (NamR[I-1] in [' '..'~']) or (I>20);

  NamR[0] := Chr(I-1);               { set string length because assigning }
                                     { by element does not set length }
  directory_files[0] := NamR;
  files_counter := 1;

  while (fault = 0) do begin
    fault := 0;
    RegsGlb.AX := $4F00;               { Function used to get the next }
                                       { directory entry }
    RegsGlb.CX := 22;                  { Set the file option }
    MSDos( RegsGlb );                  { Call MSDos }
    fault := RegsGlb.AX and $FF;       { get the fault return }
    I := 1;
    repeat
      NamR[I] := Chr(Mem[Seg(DTA):Ofs(DTA)+29+I]);
      I := I + 1;
    until not (NamR[I-1] in [' '..'~'] ) or (I > 20);
    NamR[0] := Chr(I-1);
    if (fault = 0)   THEN
      BEGIN
      directory_files [files_counter] := NamR;
      files_counter := files_counter + 1;
```

```
      END;
  END;
  FillChar(mask,SizeOf(mask),0);          {clear mask array}
  temp := 3 + files_counter DIV 5;
  ClearWindowStack(16);
  DefineTextWindow (16,4,2,77,temp,6);
  DefineHeader(16,'DIRECTORY DRIVE  '+logged_driveGlb);
  SelectScreen(2);
  SetHeaderOn;
  SelectWindow(16);
  SetBackGround(0);
  DrawBorder;
  temp := 0;
  yplot := 3;
  REPEAT
    FOR i := 0 TO 4 DO
      BEGIN
      GotoXY(7+14*i,yplot);
      j := Pos ('.',directory_files [temp]);
      Write (Copy (directory_files [temp],1,j-1));
      temp := temp + 1;
    END; {FOR}
    yplot := yplot + 1;
  UNTIL temp > files_counter+1;
  StoreWindow(16);        {store window to stack}
  SelectScreen(1);
  SelectWindow(oldwindow);

                                  { logging the system-disk back on}
  log_drive (system_driveGlb); {system drive has been determined at
                                 the beginning of main program}
end; {GetDir}

{*****************************************************************}

OVERLAY PROCEDURE store_patient;     {stores the last processed patient
                                      file }
  BEGIN
    WITH patient DO
      BEGIN
        ClearScreen;
        centre (10,'Storing Patient file');
        centre (13,file_nameGlb);
        {$I-}
        Assign (patientfile,file_nameGlb);
        Rewrite (patientfile);
        Write (patientfile,patient);

        Close (patientfile);
        {$I+}
      END;
  END;

{*****************************************************************}

OVERLAY PROCEDURE defWindows;     {this procedure sets up all the
                                   global used windows. Axis are
                                   drawn, and the array for the
                                   crank circle is calculated. This
                                   procedure is called by FNS-2.PAS}

CONST   y = 3;            {y is the y location of the x-axis labelling}

VAR   i : INTEGER;
      alpha,incr : REAL;
```

```
BEGIN

  DefineWindowIBM (1,0,0,79,185);
  DefineWorld     (1,0,0,639,185);
  DefineHeader    (1,'   IBM  Full Screen Window');

  DefineWindowIBM (2,1,10,79,185);
  DefineWorld     (2,0,0,631,165); {each pixel=one world coordinate}

 {Window 3, World 3   is local  usable eg. proc.  xy_axis_369 }

 {Window 4, World 4   is used as moving window (little arrow) in
                      procedure inputY_versus_crankposition}

 {Window 7 is used to store the opening menu of the main progra}

 {Window 8 is used to store the editfile main menu}

  DefineWindowIBM  (5,2,25,79,185);  {patient-file}

  DefineWindowIBM  (6,0,0,28,36);    {xy-axis +labelling of window 6 to
                                      plot amplitudes, pulsewidths or
                                      motorvoltage. window 6 is pasted
                                      into patientfile window 5}
  SelectScreen(2);
  DefineWorld (3,0,0,224,35);   {world for window 6}
  SelectWorld (3);
  SelectWindow(6);
  SetBackGround(0);
  DrawLine (24,7,216,7);       {xaxis}
  FOR i:=1 TO 4 DO
    BEGIN
      DrawLine (48*i,6,48*i,8);
      DrawLine (24+i*48,5,24+i*48,8);   {draw segmentation xaxis}
    END; {for i}
  DrawLine (24,7,24,31);       {yaxais}
  DrawLine (22,19,28,19);     {segmentation yaxis}
  DrawLine (22,31,28,31);

  DrawTextW (44,y,1,'45');
  DrawTextW (68,y,1,'90');
  DrawTextW (90,y,1,'135');
  DrawTextW (114,y,1,'180');   {label xaxis from 45 to 360 degree}
  DrawTextW (138,y,1,'225');
  DrawTextW (162,y,1,'270');
  DrawTextW (186,y,1,'315');
  DrawTextW (210,y,1,'360');
  StoreWindow (6);           {store xy-axis for plots in patient-file}
  SelectScreen(1);

  DefineTextWindow (14,1,1,40,12,6);
  DefineHeader (14,'H E L P    W I N D O W');

  DefineTextWindow (15,4,25,77,25,1);  {Commandline Window};

  {window 16 is the global Text-window to display directory}

  {the following is presetting the global circle array xCirclePointGlb,
   yCirclePointGlb with the screen coordinates of a circle; The global
   constants radiusGlb, xCircleGlb, yCircleGlb define the size of the
   circle; the circle array is used to plot the crankposition cir-
   cular during stimulation on the screen; see procedure start
   stimulation. It would have been possible to store this array on disk
   and to read it when the program is started. However such a file could
   go missing easily in subsequent copying of this program. Therfore I
   preferred calculating this circle array at startup of the program }
```

```
    SetWindowModeOff;
    incr := 2*Pi/256;
    DrawPoint (255,150);     {draw endpoint of busy plot}
    FOR i := 0 TO 255 DO
      BEGIN
        alpha := i * incr;
        xCirclePointGlb[i] := ROUND (xCircleGlb - radiusGlb * COS(alpha));
        yCirclePointGlb[i] := ROUND (yCircleGlb + AspectFactor * radiusGlb
                                                 * SIN(alpha));
        DrawPoint (i,150);  {signals that busy}
      END;
    SetWindowModeOn;
END; {overlay DefWind.pas}


{*****************************************************************}

OVERLAY PROCEDURE  waveform;

VAR  done : BOOLEAN;

{ This procedure is called by procedures inputStimulatioParameters
  and edit_oldfile.
  The procedures monophasic and biphasic are procedures of waveform

  Waveforms can be defined as being monophasic or biphasic. Monophasic
  waveforms are created by setting the positive pulse amplitude and
  pulse width to zero. However it is necessary to define the interpulse_
  interval for proper function of timer 2 of the 68B40 timer chip. Thus
  interpulse_interval is set to its smallest value ie. 214 us to allow
  most accurate definition of the pulse repetition frequency (for
  details see chapter 4.1.  of the thesis.
      The biphasic waveform is defined by entering the neg_pos_ratio
  and the interpulse_interval.
      The neg_pos_ratio is the ratio between the pulse widthes of the
  negative and positive going pulse. To ensure charge balance of the
  waveform the amplitudes of the positive pulses are gained from the
  amplitudes of the negative pulses by multiplication with the
  neg_pos_ratio. The positive pulse widths are gained from the
  negative pulse widths by devision with the neg_pos_ratio ! This
  calculation is performed in procedure calculate_newfile.
      The interpulse_interval defines the time interval between the
  onset of the negative pulse and the onset of the positive pulse.
  Thus at varying negative and positive pulse widths the exact time
  interval between the end of the negative pulse and the onset of the
  positive pulse is dependent on these pulse widths. The shape of the
  defined biphasic pulse waveform is drawn on the screen !
      The defined waveform is used for all stimulating channels !
  It would be possible to define different waveforms for each channel
  or waveforms varying with crank position. The hardware can do it all.
  All it needs is some additional software modules in this overlay. }

{---------------------------------------------------------------}

PROCEDURE  monophasic_pulse;

BEGIN
  WITH patient DO
    BEGIN
      pulse_frequency := 30;
      DefineHeader(2,'Monophasic pulse');
      SelectWorld(2);
      SelectWindow(2);
      SetBackGround(0);
      SetHeaderOn;
```

```
      DrawBorder;
      puts (20,10,'Monophasic  pulse waveform is  created by');
      puts (20,11,'setting the positive pulse-width to zero');
      puts (20,13,'The maximal negative pulse-width will be');
      puts (20,14,'set  when   defining   the   pulse-width');
      puts (20,15,'Envelope');
      puts (20,17,'interpulse_interval is set to  214 us !!');
      interpulse_interval := 214;  {us}
      neg_pos_ratio := 1;
      flag_biphasic := FALSE;
      delay(3000);      {delay to read the message};
    END;  {with}

END;    {monophasic_pulse}


{-----------------------------------------------------------------}

PROCEDURE biphasic_pulse;

VAR  startpos,endpos,amppos,endneg,y,temp :INTEGER;
     temp_string : string_12;

BEGIN
  WITH patient DO
    BEGIN
      IF flag_newfileGlb THEN
        BEGIN
          pulse_frequency := 50;
          max_pulsewidth  := 400;
        END;
      DefineHeader (2,'Definition Biphasic Pulse');
      SelectWorld (2);
      REPEAT
        SelectWindow(2);
        SetHeaderOn;
        SetBackGround(0);
        DrawBorder;
        REPEAT
          neg_pos_ratio := 0;
          puts (20,9,'(min.  ');
          Write (max_pulsewidth/1710:4:3,'    Default = 1)');
          puts (20,8,'Ratio negative/positive pulse-width =          ');
          GotoXY(58,8);
          neg_pos_ratio := readreal;
        UNTIL (neg_pos_ratio=0) OR ((neg_pos_ratio>max_pulseWidth/1710)
                                  AND (neg_pos_ratio<-1));
        {1710 us is the max pulsewidth which can be produced without
         setting the msb of the timer 1 of the 6840, ie. 255*pwGlb =
         = 255 * 6.7085 = 1710 us  ! }

        IF neg_pos_ratio = 0 THEN neg_pos_ratio := 1;
        REPEAT
          interpulse_interval := 0;
          puts (25,12,'Default = 600 us');
          puts (20,11,'inter pulse interval =              ');
          GotoXY(43,11);
          interpulse_interval := readint(4);
          IF interpulse_interval = 0 THEN interpulse_interval := 600;
        UNTIL ((interpulse_interval>199)AND
              (interpulse_interval<5001)) AND (pulse_frequency <=
               10*ROUND(25000/interpulse_interval));
        y := 45;
        IF max_pulsewidth = 0 THEN max_pulsewidth := 400;
        endneg := 150 + max_pulsewidth DIV 14;   {one x pixel = 14 us}
        DrawLine (100,y,530,y);
```

```
        DrawLine (150,y,150,y-30);
        DrawLine (150,y-30,endneg,y-30);
        DrawLine (endneg,y-30,endneg,y);
        startpos := 150 + interpulse_interval DIV 14;
        endpos   := startpos + ROUND (max_pulsewidth/(14*neg_pos_ratio));
        amppos   := ROUND (y + 30*neg_pos_ratio);
        DrawLine (startpos,y,startpos,amppos);
        DrawLine (startpos,amppos,endpos,amppos);
        DrawLine (endpos,amppos,endpos,y);
        Str(max_pulsewidth:3,temp_string);
        temp_string := temp_string + ' us';
        DrawTextW (150,y-35,1,temp_string);
        temp := ROUND (max_pulsewidth/neg_pos_ratio);
        Str(temp:4,temp_string);
        temp_string := temp_string + ' us';
        temp := ROUND (startpos + (endpos-startpos)/2 - 14);
        DrawTextW (temp,amppos+6,1,temp_string);
        commandLine(11);
        REPEAT
          READ(kbd,ch);
        UNTIL upcase(ch) IN [' ','Q'];

      UNTIL upcase(ch)='Q';
      ch := ' ';
      flag_biphasic := TRUE;
    END;  (with)
END;   (biphasic_pulse)

{-------------------------------------------------------------------}

BEGIN                    (begin of overlay waveform)
WITH patient DO
BEGIN
  IF flag_newfileGlb = TRUE THEN pulse_frequency := 30;
                    (setting pulse_frequency to default)
  ClearScreen;
  Done := FALSE;

  REPEAT
    SelectWorld(1);
    SelectWindow(1);
    DefineHeader(1,'Stimulation Waveform Definition');
    DrawBorder;
    centre (6,'Define Stimulation Waveform');
    puts (25,10,'1  -  Biphasic   Waveform');
    puts (25,12,'2  -  Monophasic Waveform');
    puts (25,16,'Enter your Choice --->');
    puts (47,16,'_');
    commandLine(3);
    REPEAT
      READ(kbd,ch);
    UNTIL ORD(ch) IN [49,50,59,60,81,113];
    CASE ORD(ch) OF
       49 : BEGIN
              commandLine(8);
              biphasic_pulse;
              commandLine(3);
              done:=TRUE;
              flag_biphasic := TRUE;
            END;
       50 : BEGIN
              commandLine (13);
              monophasic_pulse;
              commandLine(3);
              done := TRUE;
              flag_biphasic := FALSE;
```

```
            END;
         59 : displayHelp(3);
         60 : display_patientfile;
         81 : IF NOT flag_newfileGlb THEN done := TRUE;
        113 : IF NOT flag_newfileGlb THEN done := TRUE;
      END; {case}
    UNTIL done;

    ch := ' ';
    END; {with}
  END;    {waveform}

{*****************************************************************}

OVERLAY PROCEDURE  frequency;

 {This procedure is called by the procedures edit_oldfile and
  InputStimulationParameters.

  This procedure lets the user define the pulse repetition frequency
  of the previous defined pulsatile waveform. The defined repetition
  frequency  is valid for all channels. It is possible to extend this
  procedure to define frequencies independent in each channel or even
  as a function of the crank position (in conjunction with an extended
  version of procedure inputY_versus_crankposition) }

  BEGIN
    WITH patient DO
      BEGIN
        DefineHeader(2,'Define Pulse Frequency');
        SelectWorld(2);
        SelectWindow(2);
        SetBackGround(0);
        SetHeaderOn;
        DrawBorder;
        commandLine(8);
        puts (20,10,'Preliminary the pulse repetition frequency');
        puts (20,11,'can only be constant, and in all  channels');
        puts (20,12,'the same  !!!!');
        REPEAT
          puts (20,16,'Pulse repetition Frequency =              ');
          GotoXY(49,16);
          pulse_frequency := readint(2);
        UNTIL (pulse_frequency <= 10*ROUND(25000/interpulse_interval))
              AND (pulse_frequency>10);
      END;  {with}
      commandLine(3);
  END;       {frequency}


{*****************************************************************}

OVERLAY PROCEDURE  pulse_width;

VAR          done : BOOLEAN;
                i : INTEGER;
     temp_string : string_5;


{This procedure is called by procedures inputStimulationParameters
 and edit_oldfile.
 constant_pulsewidth and editPulsewidth_OneChannel are procedures of
 overlay procedure pulsewidth.
 The pulse width can be defined as being constant in respect to the
 crank position or to be a function of the crank position. If the
 pulse width is constant the stimualtion level must be controlled by
```

varying the amplitude referring to the crank position and vice versa.
Constant pulse widths can be defined independently in each channel. Practical values range from 200 us to 500 us.
Before a pulse width envelope is defined in a new patient file ie. flag_newfileGlb = TRUE the program asks for the maximum pulse width to be used in all the channels. This pulse width is then used to define the maximum value for the y axis in the procedure inputY_versus_crankposition)

```
{-------------------------------------------------------------------}

PROCEDURE  constant_pulsewidth;

VAR
  i,j,k,l              : INTEGER;
  channel_pulsewidth : ARRAY[1..max_chGlb] of INTEGER;

BEGIN
  WITH patient DO
    BEGIN
      DefineHeader(2,'Define Constant Pulse-width');
      SelectWorld (2);
      SelectWindow(2);
      SetBackGround(0);
      SetHeaderOn;
      DrawBorder;
      puts (30,8,'Define  Constant   Pulse-width');
      puts (30,9,'independently for each channel');
      FOR  i := 0 TO 1 DO
        BEGIN
          FOR j := 1 TO chHalfGlb DO
            BEGIN
              k := 13 + i*(chHalfGlb+1) + j;
              REPEAT
                puts (6,k,'    Pulse-width in Ch ');
                Write (j,' ');
                IF i = 0 THEN Write ('Left ')
                  ELSE Write ('Right');
                Write (' Leg = ',ROUND(pulsewidth[i*chGlb+j,0]*pwGlb),
                       ' us, new pulsewidth = ');
                channel_pulsewidth[i*chGlb +j] := readint(3);
              UNTIL channel_pulsewidth[i*chGlb+j] < 1000;
              IF (channel_pulsewidth[i*chGlb+j]=0) AND NOT flag_newfileGlb
                 THEN channel_pulsewidth[i*chGlb+j]:=ROUND
                 (pulsewidth[i*chGlb+j,0]*pwGlb);
            END; (for j)
        END; (for i)

      max_pulsewidth := 0;  (the maximal pulse width will be found now)

      FOR i := 0 TO 1 DO
        BEGIN
          FOR j := 1 TO chHalfGlb DO
            BEGIN
              l := i*chGlb + j;
              IF channel_pulsewidth[l]>max_pulsewidth THEN
                                max_pulsewidth := channel_pulsewidth[l];
              FOR k := 0 TO 255 DO
                BEGIN
                  pulsewidth[l,k] := ROUND(channel_pulsewidth[l]/pwGlb);
                END;
            END; (for j)
        END; (for i)
    flag_pwm_control := FALSE;
```

```
    END;    {with}
END; {constant_pulsewidth}

{-------------------------------------------------------------------}

PROCEDURE  editPulseWidth_Onechannel;

VAR i,code,chan : INTEGER;

BEGIN
WITH patient DO
BEGIN
  ch := ' ';
  DefineHeader(1,'CHANGE PULSE-WIDTH OF ONE CHANNEL');
  SetHeaderOn;
  REPEAT
    SelectWorld(1);
    SelectWindow(1);
    SetBackGround(0);
    DrawBorder;
    commandLine(6);
    puts(25,6,'Change Pulse-width of one Channel');
    puts(25,12,'Left or Right Leg  (L/R) ?');
    REPEAT
      READ (kbd,ch);
    UNTIL upcase(ch) IN ['L','R','Q'];
    IF upcase(ch)='R' THEN chan:=chGlb ELSE chan:=0;
    puts(50,12,upcase(ch));
    IF upcase(ch)<> 'Q' THEN
      BEGIN
        puts(25,14,'Which channel  ');
        Write ('[1...',chHalfGlb,']   ?    ');
        GotoXY(50,14);
        REPEAT
          READ(kbd,ch);
          Val(ch,i,code);
        UNTIL (i IN [1..chHalfGlb]) OR (upcase(ch)='Q');
        IF i IN [1..chHalfGlb] THEN
          BEGIN
            i := i + chan;
            inputY_versus_crankposition (i,0);
            ch := ' ';
          END;
      END; {if not q}
  UNTIL (ch='q') OR (ch='Q');
  END; {with}
END; { editPulseWidth_OneChannel}

{-------------------------------------------------------------------}

BEGIN                  {begin of overlay procedure pulsewidth}
  WITH patient DO
    BEGIN
      done := FALSE;
      IF (flag_amp_control=FALSE) AND (flag_newfileGlb=FALSE) THEN
        BEGIN
          editPulseWidth_OneChannel;
          done := TRUE;
        END; {IF}
      IF  flag_amp_control AND (flag_newfileGlb=FALSE) THEN
        BEGIN
        commandLine(8);
        constant_pulsewidth;
        commandLine(3);
        done := TRUE;
      END;
```

```
WHILE done = FALSE DO
  BEGIN
    DefineHeader(1,'Definition of Pulse-Width');
    SelectWorld (1);
    SelectWindow(1);
    SetBackGround(0);
    SetHeaderOn;
    DrawBorder;
    centre (6,'Define Pulse-Width');
    puts (27,10,'1  =  Constant Pulse-width');
    puts (27,12,'2  =  Variable Pulse-width');
    puts (27,16,'Enter your choice ---> _');
    commandLine(3);
    REPEAT
      READ(kbd,ch);
    UNTIL ORD(ch) IN [49,50,59,60,91,113];
    CASE ORD(ch) OF
      49  : BEGIN
              IF flag_amp_control THEN
                BEGIN
                  commandLine(8);
                  constant_pulsewidth;
                  commandLine(3);
                  done := TRUE;
                END  {if}
                ELSE
                  BEGIN
                    puts(20,20,'if no amplitude modulation then');
                    puts(20,21,'then pulse-width must be variable');
                    beep;
                    delay (2000);
                  END; {IF and ELSE}
            END; {49}
      50  : BEGIN
              IF flag_newfileGlb THEN
                BEGIN
                  SelectWorld(1);
                  DefineHeader (1,'Define Maximal negative Puse-width
                  SetHeaderOn;
                  SelectWindow(1);
                  SetBackGround(0);
                  DrawBorder;
                  puts (30,8, 'Define Maximal Pulse-width  of');
                  puts (30,10,'   the negative going Pulse');
                  REPEAT
                    puts (25,16,'max. Pulse-width [100..999 us] =
                    GotoXY (58,16);
                    max_pulseWidth := readint(3);
                  UNTIL max_pulseWidth > 99;

                  FOR i := 1 TO chHalfGlb DO
                    BEGIN
                      inputY_versus_crankposition(i,0);
                    END;
                  done := TRUE;
                  flag_pwm_control := TRUE;
                END
                ELSE editPulseWidth_OneChannel;
            END; {50}

      59  : displayHelp(6);
      60  : display_patientfile;
      81  : done := TRUE;
      113 : done := TRUE;
    END; {case}
  END; {while}
```

```
    END;  {with}
END;  {pulsewidth}


{*****************************************************************}

OVERLAY PROCEDURE  amplitude;

VAR       done : BOOLEAN;
             i : INTEGER;
  temp_string : string_5;

{This procedure is called by procedures inputStimulationParameters
 and edit_oldfile. Constant_amplitude and editAmplitude_Onechannel
 are procedures of overlay procedure amplitude.
     The pulse amplitudes can be defined as being constant in respect
 to the crank position or to be a function of the crank position.
 If the pulse amplitude is constant the stimualtion level must be
 controlled by varying the  pulse width referring to the crank
 position and vice versa.
     Constant pulse amplitudes can be defined independently in each
 channel. Practical values range from 50 mA  to  maximum 100 mA.
     Variable Pulse amplitudes are defined with procedure inputY_
 versus_crankposition. Note that the amplitude of the negative
 going ie. stimulating pulse is defined and that the amplitude of
 the positive going pulse is calculated from the negative one via
 the neg_pos_ratio in procedure calculate_ newfile (see also the
 comments at procedure waveform for details !). }

{---------------------------------------------------------------}

PROCEDURE  constant_amplitude;

VAR
  i,j,k,l             : INTEGER;
  channel_amplitude : ARRAY[1..max_chGlb] of INTEGER;

BEGIN
  WITH patient DO
    BEGIN
      DefineHeader(2,'Define Constant Amplitude');
      SelectWorld (2);
      SelectWindow(2);
      SetBackGround(0);
      SetHeaderOn;
      DrawBorder;
      puts (25,8,'Define  Constant   Amplitude');
      puts (25,9,'independently for each channel');

      FOR  i := 0 TO 1 DO
        BEGIN
          FOR j := 1 TO chHalfGlb DO
            BEGIN
              k := 13 + i*(chHalfGlb+1) + j;
              REPEAT
                puts (6,k,'max. Amplitude in chan. ');
                Write (j,' ');
                IF i = 0 THEN Write ('Left ')
                  ELSE Write ('Right');
                Write (' Leg = ',ROUND (neg_amplitude[i*chGlb+j,0]/2.55),
                       ' mA, new amplitude = ');
                channel_amplitude[i*chGlb+j] := readint(3);
              UNTIL channel_amplitude[i*chGlb+j] < 100.001;
              IF channel_amplitude[i*chGlb+j] = 0 THEN channel_amplitude
                [i*chGlb+j]:=ROUND (neg_amplitude[i*chGlb+j,0]/2.55);
            END; {for j}
```

```
        END; {for i}

      FOR i := 0 TO 1 DO                    {0 mA = 0  as byte value}
        BEGIN                               {100 mA = 254 as byte value}
          FOR j := 1 TO chHalfGlb DO
            BEGIN
              FOR k := 0 TO 255 DO
                BEGIN
                  l := i*chGlb + j;
                  neg_amplitude[l,k] := ROUND(2.54*channel_amplitude[l]);
                  pos_amplitude[l,k] := ROUND(neg_pos_ratio*
                                        neg_amplitude[l,k]);
                END; {for k}
            END; {for j}
        END; {for i}
      flag_amp_control := FALSE;
    END;   {with}
END; {constant_amplitude}

{-------------------------------------------------------------------}

PROCEDURE  editAmplitude_Onechannel;

VAR i,code,chan : INTEGER;

BEGIN
  WITH patient DO
  BEGIN
  ch := ' ';
  DefineHeader(2,'CHANGE AMPLITUDE OF ONE CHANNEL');
  SetHeaderOn;
  REPEAT
    SelectWorld(1);
    SelectWindow(1);
    SetBackGround(0);
    DrawBorder;
    commandLine(6);
    puts(25,6,'Change Amplitude of one Channel');
    puts(25,12,'Left or Right Leg  (L/R) ?');
    REPEAT
      READ (kbd,ch);
    UNTIL upcase(ch) IN ['L','R','Q'];
    puts (50,12,upcase(ch));
    IF upcase(ch)='R' THEN chan := chGlb ELSE chan:=0;
    IF upcase(ch)<> 'Q' THEN
      BEGIN
        puts(25,14,'Which channel  ');
        Write ('[1...',chHalfGlb,']   ?    ');
        GotoXY(50,14);
        REPEAT
          READ(kbd,ch);
          Val(ch,i,code);
        UNTIL (i IN [1..chHalfGlb]) OR (ch='q') OR (ch='Q');
        IF i IN [1..chHalfGlb] THEN
          BEGIN
            i := i + chan;
            inputY_versus_crankposition (i,1);
            ch := ' ';
          END;
      END; {if not q}
  UNTIL (ch='q') OR (ch='Q');
END; {with}
END; { editAmplitude_OneChannel}

{-------------------------------------------------------------------}
```

```
BEGIN                    {begin of overlay procedure amplitude}
  WITH patient DO
    BEGIN
      done := FALSE;
      IF (flag_pwm_control=FALSE) AND (flag_newfileGlb=FALSE) THEN
        BEGIN
          editAmplitude_OneChannel;
          done := TRUE;
        END; {IF}
      IF  flag_pwm_control AND (flag_newfileGlb=FALSE) THEN
        BEGIN
          commandLine(8);
          constant_amplitude;
          commandLine(3);
          done := TRUE;
        END;
      WHILE done = FALSE DO
        BEGIN
          DefineHeader(1,'Definition of Pulse Amplitudes');
          SelectWorld (1);
          SelectWindow(1);
          SetBackGround(0);
          SetHeaderOn;
          DrawBorder;
          centre (6,'Define Pulse Amplitudes');
          IF flag_pwm_control THEN
            BEGIN
              commandLine(12);
              constant_amplitude;
              commandLine(3);
              done := TRUE;
            END;  {if}
          IF (flag_newfileGlb) AND (flag_pwm_control=FALSE) THEN
            BEGIN
              FOR i := 1 TO chHalfGlb DO
                BEGIN
                  inputY_versus_crankposition(i,1);
                END;
              flag_amp_control := TRUE;
              done := TRUE;
            END;  {if}

          IF (flag_newfileGlb=FALSE) AND flag_amp_control THEN
            editAmplitude_OneChannel;

        END; {while}
    END;  {with}
END;  {amplitude}

{*****************************************************************}

OVERLAY PROCEDURE  motorvoltage;

VAR                                  done : BOOLEAN;
   i,old_motor_power, old_rpm_limit : INTEGER;
                          temp_string : string_5;


   {This procedure is called by procedures inputStimulationParameters
    and edit_oldfile. The motor voltage is defined with procedure
    inputY_versus_crankposition. See this procedure for details.}


BEGIN
  WITH patient DO
    BEGIN
```

```
      DefineHeader(1,'Definition of Motor Voltage versus crank-position');
      SelectWorld (1);
      SelectWindow(1);
      SetBackGround(0);
      SetHeaderOn;
      DrawBorder;
      inputY_versus_crankposition (7,2);
    END;  {with}
    commandLine(3);
END;  {motorvoltage}

{*************************************************************************}

OVERLAY PROCEDURE namefile (flag_copy:BOOLEAN);

  VAR  i,j : INTEGER;

  {This procedure is called by procedures inputStimulationParameters
  and edit_oldfile.
      Purpose is to enter the patient's personal data such as name
  birthday, injury date etc. into the patient file.
      If flag_copy is TRUE the stimulation data of the last
  processed patient file is labelled with a new file_nameGlb and
  the new personal data and stored under the new file_nameGlb.
  This 'copy' feature allows the use of already defined data files
  which are renamed and copied with the personal data into a new
  patient file (ie. the used predefined data file doesn't get lost !).
  The  stimulation patterns etc. can then be edited and customized for
  the  new patient. }


  BEGIN
    WITH patient DO
      BEGIN
        DefineHeader (1,'Enter Patient Information');
        SetHeaderOn;
        SelectWindow (1);     {0,0,79,185}
        SetBackGround(0);
        DrawBorder;
        commandLine(8);
        name := '';                {reset these string variables}
        initials := '';
        injury_date := '';
        lesion_level := '';
        centre (3,'OPEN  PATIENT  FILE :');
        REPEAT
          puts (25,6,'Patient Surname       : ');
          name := readstr (12);
        UNTIL length (name)<13;
        puts (25,8,'Patient Initials      : ');
        initials := readstr (8);
        puts (25,10,'Patient's birth-date  : ');
        birth_day := readstr (8);
        puts (25,12,'date of injury        : ');
        injury_date := readstr (8);
        puts (25,14,'lesion level          : ');
        lesion_level := readstr (8);
        REPEAT
          puts (25,17,'per leg');
          puts (25,16,'number of channels    : ');
          GotoXY (49,16);
          ch_max := readint(1);
        UNTIL ch_max IN [1..3];

       {The following If tests whether additional channels are to be
        defined when an existing patient file is copied into a new file
```

```
        with a new file name. Thus the stored plot array is preset with
        zero envelope values for zero degree and 180 degrees crank
        position to indicate that no data has been entered into the
        additional defined channels. The lock up tables are also preset
        with zero values !}

        IF (ch_max > chHalfGlb) AND flag_copy THEN
          BEGIN
            FOR i:= chHalfGlb+1 TO ch_max DO
              BEGIN
                FOR j:=1 TO 64 DO
                  BEGIN
                    stored_plotarray[i,j,1] := 0;
                    stored_plotarray[i,j,2] := 0;
                  END;
                stored_plotarray[i,1,1] := 60;
                stored_plotarray[i,1,2] := 21;
                stored_plotarray[i,33,1] := 316;  {place data point at}
                stored_plotarray[i,33,2] := 21;   {180 degree crank pos.}
                FOR j:=0 TO 255 DO
                  BEGIN
                    pulsewidth [i,j] := 0;
                    lsb_t1 [i,j] := 0;
                    lsb_t3 [i,j] := 0;
                  END;
              END;
            FOR i:= chGlb+chHalfGlb+1 TO chGlb+ch_max DO
              BEGIN
                FOR j:=1 TO 64 DO
                  BEGIN
                    stored_plotarray[i,j,1] := 0;
                    stored_plotarray[i,j,2] := 0;
                  END;
                stored_plotarray[i,1,1] := 60;
                stored_plotarray[i,1,2] := 21;
                stored_plotarray[i,33,1] := 316;  {place data point at}
                stored_plotarray[i,33,2] := 21;   {180 degree crank pos.}
                FOR j:=0 TO 255 DO
                  BEGIN
                    pulsewidth [i,j] := 0;
                    lsb_t1 [i,j] := 0;
                    lsb_t3 [i,j] := 0;
                  END;
              END;
          END; {if}

        chHalfGlb := ch_max;
        ch_max := ch_max * 2;
        DefineHeader (1,' ');   {reset header window 1}
        IF flag_copy THEN
          BEGIN
            name := Copy (name,1,8);
            file_nameGlb := Concat (logged_driveGlb,name,'.FNS');
            Assign (patientfile,file_nameGlb);
            Rewrite (patientfile);
          END;
      END;  {with}
  END;   {procedure namefile}

{*****************************************************************}

OVERLAY PROCEDURE calculate_newfile;

VAR
  i,j,k,l,m,temp_ipi,temp_pwI1,msb2,lsb2,br_drive : INTEGER;
                              drive : BYTE;
```

```
                inv_neg_pos_ratio : REAL;


{This procedure is called by procedure edid_oldfile and calculates the
 look up tables for real time stimulation. Look up tables are the
 following arrays :
                  msb_t1 [channel,crankposition]  OF BYTE
                  msb_t2 [channel,crankposition]  OF BYTE
                  msb_t3 [channel,crankposition]  OF BYTE
                  lsb_t1 [channel,crankposition]  OF BYTE
                  lsb_t2 [channel,crankposition]  OF BYTE
                  lsb_t3 [channel,crankposition]  OF BYTE

            neg_amplitude [channel,crankposition]  OF BYTE
            pos_amplitude [channel,crankposition]  OF BYTE

        set_motor_voltage [crankposition]   OF BYTE
              brake_drive [crankposition]   OF BOOLEAN



   NOTE that because of the mechanical arrangement (gears) the
 readings of the HP shaft encoder are reversed as follows :
 The zero degree position of the crank is equal to a shaft encoder
 reading of 255. The 358.4 degree position of the crank gives a shaft
 encoder reading of 0. Thus the envelopes defined in procedure inputY_
 versus_crankposition have to be reversed :

        new array position = 255 - old array position ;}

BEGIN
  WITH patient DO
    BEGIN
      FillChar(lsb_t1,SizeOf(lsb_t1),0);  {reset of array lsb_t1}
      FillChar(lsb_t3,SizeOf(lsb_t3),0);  {reset of array lsb_t3}
      ClearScreen;
      centre (8,'Calculating The New Data File');
      centre (10,file_nameG1b);
      FOR i := 1 TO 2*chHalfG1b DO puts (32+i*2,15,'.');
      GotoXY (34,15);
      inv_neg_pos_ratio := 1/neg_pos_ratio;
      FOR i := 0 TO 1 DO
       BEGIN
         FOR j := 1 TO chHalfG1b DO
           BEGIN
             lsb2 := ROUND (interpulse_interval/Tclk2G1b);
             msb2 := (ROUND (1E6/(pulse_frequency*(lsb2+1)*Tclk2G1b)))-1;
             FOR l := 0 TO 255 DO
              BEGIN
                m := i * chG1b + j;
                k := 255-l;     {as 255 is 0 degree and 0 is 358.4 degree}
                msb_t2[m,k] := msb2;          {using the HP shaft encoder}
                lsb_t2[m,k] := lsb2;
                lsb_t3[m,k] := pulseWidth[m,1];    {T3 = neg pulse}
                msb_t3[m,k] := 0;
                IF flag_biphasic THEN     {T1 = pos pulse/neg_pos_ratio}
                  BEGIN
                    temp_pwT1 := ROUND (pulseWidth[m,1]*inv_neg_pos_ratio);
                    msb_t1[m,k] := HI (temp_pwT1);
                    lsb_t1[m,k] := LO (temp_pwT1);
                  END
                ELSE
                  BEGIN
                    msb_t1[m,k] := 0;
                    lsb_t1[m,k] := 0;
                  END;
```

```
                pos_amplitude[m,1] := ROUND(neg_amplitude[m,1]*
                                      neg_pos_ratio);
              END; {l}
              Write ('  ');
            END;  {j}
          END;  {i}


        FOR i := 0 TO 255 DO
          BEGIN
            k := 255-i;
            br_drive:=(set_motor_power[i] AND $80);
            IF br_drive=$80 THEN brake_drive[k]:=TRUE
              ELSE brake_drive[k]:=FALSE;
            IF brake_drive[k] THEN
              set_motor_voltage[k] := 2*(set_motor_power[i] AND $7F)
             ELSE
              set_motor_voltage[k] := 2 * (127-(set_motor_power[i] AND $7F));
          END; {for i}

      END; {with}
END;  {calculate_newfile}

{*************************************************************************}

OVERLAY PROCEDURE calibrate_impedance_check;

CONST   pw = 100;     {test pulsewidth in us}

VAR
  i,j,p,lsbT13,pulsewidth,amplitude,max_ch01b : INTEGER;
  trigger : BYTE;
  voltage,impedance : REAL;
  q : string_20;

  {This procedure is called by procedure Stimulate. Its purpose is
   the calibration of the opto coupler circuitry  which allows the
   measurement of the output voltage of the isolation amplifier and
   thus the calculation of the electrode impedance (this is possible
   because the output current is stable within 5% for electrode im-
   pedances varying between 200 Ohms and 1500 Ohms).
      For the measurement of the output voltage a biphasic symmetric
   waveform is generated. The neg_pos_ratio is 1, ie. the positive
   pulse width is equal to the negative pulse width = 100 us. The
   positive and the negative pulse are triggered by the software and
   the A/D board samples and digitizes the output of the opto coupler
   about 40 us after a negative pulse is triggered. Only the negative
   pulse voltage is measured.
      To trigger the pulse generation by the software (ie. by a write
   to timer latches command) the controlword A0 Hex has to be stored
   in the control register CR1 and CR3 of timer 1 and 3 of the Motorola
   68B40 timer chip. Timer 2 is disabled by writing the control word
   35 Hex into the control register of timer 2 (ie. CR2). The timer
   3 is triggered by writing first the msb Byte to the msb latch of
   timer 3 followed by the writing of the lsb Byte to the lsb latch
   of timer 3. The pulse is triggered by writing the lsb to the lsb
   latch of timer 3. The positive pulse is generated in the same way,
   but its voltage amplitude is not measures.
      See chapter 9.1.2.  for  details of programming the Motorola
   68B40 timer. }

{-------------------------------------------------------------------------}

FUNCTION stim_output_voltage(channel:INTEGER):REAL;
```

```
VAR trigger,j : BYTE;              {this Function triggers an 90 us long
                                    pulse and returns its voltage in V}
BEGIN
  IF channel IN [1..6] THEN
    BEGIN
      Port [BaADG1b+2] := channel * $10 +2;  {channel selection A/D}
      trigger := channel * $10 + 3;     {preparing the trigger for the
                                         A/D conversion. The A/D
                              conversion is triggered by setting the lsb
                              of port BaADG1b high (ie.one)}

      Port [BaG1b-2+channel*8] := 0;    {msb of T3 is set 0}
      Port [BaG1b-1+channel*8] := ROUND(pw/pwG1b);
                                        {trigger test pulsewidth = 100 us}

      FOR j:= 1 TO 3 DO     {wait ca. 40 us}
        BEGIN
        END;

      Port [BaADG1b+2] := trigger;     {set trigger bit C0 high, thus
                                        initiating the A/D}
      FOR j:= 1 TO 4 DO              {wait 40 us for end of conversion}
        BEGIN
        END;

      {The output voltage of the stimulation isolation amplifier is
       calculated directly from the 12-bit result (ie. 0....4095) of
       the A/D conversion. For calculation of the output voltage the
       linear regression method is used. The values for the linear
       regression were calculated from experimental data ie. the
       comparison of the measured output voltage (with oscilloscope)
       of the isolation amplifier with the result of the A/D
       conversion. }

      stim_output_voltage :=0.101080902*(((Port[BaADG1b+1]AND$0F) SHL 8)
                              + Port[BaADG1b]) + 16.53695415;
    END {if}
   ELSE stim_output_voltage := 0;
END;

{-------------------------------------------------------------------}

BEGIN
  ClearScreen;
  centre (3,'U N P L U G    A L L    E L E C T R O D E S   F R O M');
  centre (5,'  T H E    S T I M U L A T O R');
  centre (10,'Do You really want to calibrate the impedance Check ?');
  centre (15,' Y / N ? ');
  REPEAT
    q := readstr(1);
    q := Copy (q,1,1);
  UNTIL upcase (q) IN ['Y','N'];
  IF upcase(q) = 'Y' THEN
BEGIN
  ClearScreen;
  lsbT13 := ROUND(pw/pwG1b);
  p := BaG1b;
  REPEAT
    Port [p+1] := 0;     {select control register CR 3}
    Port [p]   := $A0;   {T3 = single shot triggered by write to latches}
    Port [p+1] := $35;   {select CR 1 and disable timer 2}
    Port [p]   := $A0;   {T1 see T3 programming}
    p := p + 8;
  UNTIL p > BaG1b + $28;
```

```
  centre (2,'CALIBRATION OF THE IMPEDANCE-CHECK HARDWARE  ');
  puts (10,8,'-  connect a 1000 Ohms resistor to the stimulator output;');
  puts (10,9,'-  view the output signal with an oscilloscope;');
  puts (10,10,'-  calibrate the neg. pulse amplitude with the potentiometer
  puts (10,11,'   below the flashing LED to -100 Volts;');
  puts (10,12,'-  calibrate the pos. pulse amplitude with the potentiometer
  puts (10,13,'   above the flashing LED to +100 Volts;');
  puts (10,15,'-  PRESS  ANY  KEY  TO  CONTINUE');

  FOR i := 1 TO 6 DO    {i = number channel to be tested}
    BEGIN
      centre (6,'C H A N N E L   ');
      Write (i);
      Port [BaGlb+$2F+i*2] := 254;  {set neg. amplitude to - 100mA}
      Port [BaGlb+$2E+i*2] := 254;  {set pos. amplitude to + 100mA}
      REPEAT
        Port [BaGlb-2+i*8] := 0;    {msb_T3 = 0}
        Port [BaGlb-1+i*8] := lsbT13;
        FOR j := 1 TO 10 DO         {delay of approx. 200 us between neg.}
          BEGIN                     {pos. pulse}
          END;
        Port [BaGlb-6+i*8] := 0;    {msb_T1 = 0}
        Port [BaGlb-5+i*8] := lsbT13;
        delay (20);      {approx 20 ms  delay}
      UNTIL KeyPressed;
      clrKbd;

  puts (10,17,'-  calibrate the potentiometer  (right upper hole)  with a')
  puts (10,18,'   long isolated screwdriver to  give an impedance reading')
  puts (10,19,'   of  1000  Ohms  !!!!');
  puts (10,21,'   PRESS  ANY   KEY  TO   CONTINUE !');

      REPEAT
        voltage := stim_output_voltage(i);
        impedance := voltage/0.1;
        puts (10,23,'impedance = ');
        Write (impedance:5:1,'  Ohms  ');
        GotoXY (40,23);
        Write ('   Voltage = ',voltage:4:1,'  Volts  ');
        delay(100);
      UNTIL KeyPressed;
      clrKbd;

      FOR j := 17 TO 23 DO   {clears lines 17 to 23}
        BEGIN
          GotoXY(1,j);
          ClrEol;
        END;

    END;  {for i}
END; {if q = 'Y'}
  ClearScreen;
END;  {overlay calibrate_impedance_check}

{*****************************************************************}

OVERLAY PROCEDURE impedance_check;

CONST  pw = 100;

VAR
  i,j,p,T3_lsb,pulsewidth,amplitude,max_chGlb : INTEGER;
  trigger : BYTE;
  voltage,impedance : REAL;

{This procedure is called by procedure Stimulate and checks the
```

```
 electrode impedance of the channels which are to be used for
 stimulation. The measurement of the output voltage is performed as
 already described in the comments of procedure calibrate_impedance_
 check. The electrode impedance is calculated with Ohms law from the
 measured output voltage and the defined pulse current (ie. 80 mA) }

{-------------------------------------------------------------------}

FUNCTION stim_output_voltage(channel:INTEGER):REAL;

VAR trigger,j : BYTE;            {this Function triggers an 90 us long
                                  pulse and returns its voltage in V}
BEGIN
  IF channel IN [1..6] THEN
    BEGIN
      Port [BaADG1b+2] := channel * $10 +2;
      trigger := channel * $10 + 3;
      Port [BaG1b-2+channel*8] := 0;    {msb of T3 is set 0}
      Port [BaG1b-1+channel*8] := ROUND (pw/pwG1b);
                                        {trigger test pulsewidth = 100 us}

      FOR j:= 1 TO 3 DO     {wait ca. 40 us}
        BEGIN
        END;

      Port [BaADG1b+2] := trigger;     {set trigger bit C0 high, thus
                                       initiating the A/D}
      FOR j:= 1 TO 4 DO     {wait 40 us}
        BEGIN
        END;

      stim_output_voltage :=0.101080902*(((Port[BaADG1b+1]AND$0F) SHL 8)
                             + Port[BaADG1b]) + 16.53695415;
    END {if}
   ELSE stim_output_voltage := 0;
END;

{-------------------------------------------------------------------}

PROCEDURE TimerInitialisation;

VAR
   i,p : INTEGER;

BEGIN
   p := BaG1b;
   WITH patient DO
     BEGIN
       REPEAT
         Port [P+1] := 1;    {select CR1, LSB CR2=1}
         Port [P]   := 1;    {all timers preset}
         Port [P+2] := 0;    {M1}
         Port [P+3] := 0;    {L1}
         Port [P+4] := msb_t2[1,0];    {set frequency}
         Port [P+5] := lsb_t2[1,0];    {set interpulse interval}
         Port [P+6] := 0;    {M3}
         Port [P+7] := 0;    {L3}
         Port [P+1] := 0;    {select CR3, LSB CR2=0}
         Port [P]   :=$80;   {T3 in single shot mode, 16 bit}
         Port [P+1] :=$95;   {T2 in continuous mode,8 bit,
                              select CR1 with LSB CR2=1}
         Port [P]   :=$80;   {T1 in single shot mode,
                              all timers in operation}
         P:=P+8;
       UNTIL P > BaG1b + $28;
     END; {with}
```

```
END;  {timerinitialization}

{-------------------------------------------------------------------}

BEGIN
  ClearScreen;
  WITH  patient DO
    BEGIN
      max_chGlb := ch_max;
    END;
  p := BaGlb;
  REPEAT
    Port [p+1] := 0;     {select control register CR 3}
    Port [p]   := $A0;   {T3 - single shot triggered by wright to latches}
    Port [p+1] := $35;   {select CR 1 and disable timer 2}
    Port [p]   := $A0;   {T1 - single shot triggered by wright to latches}
    p := p + 8;
  UNTIL p > BaGlb + $28;
  centre (2,'Electrode Impedance Check');

  amplitude := 80;      {setting amplitude of the check pulse to 80 mA}

  REPEAT
    centre (23,'Press Spacebar to trigger Test Sequence , Q --> QUIT !!');
    REPEAT
      getkey(ch);
    UNTIL (upcase(ch) = 'Q') OR (ch=' ');
    IF upcase(ch) <> 'Q' THEN
     BEGIN
      FOR i:= 1 TO chHalfGlb DO
        BEGIN
          Port [BaGlb+$2F+i*2] := ROUND (amplitude*2.54);  {set amplitude}
          voltage := stim_output_voltage (i);
          puts (1,5+2*i,'CHANNEL ');
          Write (i);
          puts (20,5+2*i,'Impedance = ');
          impedance := 1000 * voltage/amplitude;
          Write (impedance:6:1,' Ohms');
          puts (55,5+2*i,'Vout = ');
          Write (voltage:4:1,' Volts    ');
      END;
      FOR i:=chGLB+1 TO chGlb+max_chGlb DIV 2 DO
        BEGIN
          Port [BaGlb+$2F+i*2] := ROUND (amplitude*2.54);  {set amplitude}
          voltage := stim_output_voltage (i);
          puts (1,7+2*i,'CHANNEL ');
          Write (i);
          puts (20,7+2*i,'Impedance = ');
          impedance := 1000 * voltage/amplitude;
          Write (impedance:6:1,' Ohms');
          puts (55,7+2*i,'Vout = ');
          Write (voltage:4:1,' Volts    ');
        END;
     END; {if}

  UNTIL upcase(ch) = 'Q';

  ch := ' ';
  TimerInitialisation;
  resetD_A;
  ClearScreen;
END;  {overlay impedance_check}

{*******************************************************************}
```

```
{$I stimulat.ovl }    {overlay procedures for real time stimulation}

     {*****************************************************
      *                                                   *
      *       Real Time Stimulation Module for the        *
      *                                                   *
      *                  UCT  PARACYCLE                   *
      *                                                   *
      *                  H.M. Popp  7/86                  *
      *                                                   *
      *****************************************************}

OVERLAY PROCEDURE start_stimulation;

{this procedure is called by the procedure stimualate}

VAR   i,j,k,l,m,                  {loop count variables}
      Pcount,                     {count variable to count the 8 samples
                                   for calculating the motor power}
      VoltageSum,                 {the sum of 8 motor voltage samples}
      CurrentSum,                 {the sum of 8 motor current samples}
      f,                          {pulse repetition frequency}
      temp,                       {temporary variable}
      radius,                     {radius of the crank circle on the screen}
      power,                      {the calculated motor power in Watts}
      oldpower,                   {the last displayed motor power}
      patient_speed_control       {level of the patient speed control}
      : INTEGER;

      voltage,                    {the sampled motor voltage}
      current,                    {the sampled motor current}
      powerfactor                 {factor for adjusting range of the}
      : REAL;                     {screen power display}

      crank_position,              {the current crank position}
      old_crank_position,          {the previous crank position}
      operator_FNS_control,        {the level of operator FNS control}
      speed_control,               {the combined speed control ie. the
                                    product of operator FNS control and
                                    subject speed control}
      motor_control,              {level of operator motor control}
      old_motor_control,          {the old level of operator motor control}
      speed,                      {the current crank speed in rpm}
      old_speed_control           {the old crank speed in rpm}
      : BYTE;

      temp_string                 {temporary string variable}
      : String_20;

      display,          {flag that indicates which set of current pulse
                         widths is displayed on the screen in procedure
                         settimers. If TRUE the channels of the left leg
                         are displayed}

      flag_stop         {this flag is FALSE as long as the subject presses
                         the button in the speed control lever at the left
                         arm rest of the paracycle. If he releases the
                         button the stimulation and motor assist stops.
                         Stimulation stars as soon as the button is
                         pressed again !}
      : BOOLEAN;

{-------------------------------------------------------------------}
```

```
{the following are the Procedures of the
 overlay procedure start_Stimulation}

{-------------------------------------------------------------------}

FUNCTION ADSAMPLE (channel:INTEGER): INTEGER;  {returns the result  }
                                               {of 12-bit A/D conversion}
                                               {of the selected channel}
  VAR I:INTEGER;                               {assuming monopolar mode}
                                               { 0 to + 10 V range     }
  BEGIN
    PORT [BaADG1b+2] := 02;                    {clear channel selection and  }
                                               {software-clock bit     }
    PORT [BaADG1b+2] := channel * $10 +3;      {channel selection and }
                                              {setting of software-clock bit}
    FOR I:=1 TO 5 DO
      BEGIN                          {loop until end of conversion }
      END;

    ADSAMPLE :=((PORT[BaADG1b+1] AND $0F) SHL 8) + PORT[BaADG1b];
  END;

{-------------------------------------------------------------------}

PROCEDURE TimerInitialisation;    {presets the timers of all six
                                   68840 chips and selects the
                                continuous operastion mode for timer 2
                                and the single shot mode for timer 1
                                and 3  }
VAR
   i,p : INTEGER;

BEGIN
   p := BaG1b;
   WITH patient DO
     BEGIN
       REPEAT
         Port [P+1] := 1;    {select CR1, LSB CR2=1}
         Port [P]   := 1;    {all timers preset}
         Port [P+2] := 0;    {M1}
         Port [P+3] := 0;    {L1}
         Port [P+4] := msb_t2[1,0];    {set frequency}
         Port [P+5] := lsb_t2[1,0];    {set interpulse interval}
         Port [P+6] := 0;    {M3}
         Port [P+7] := 0;    {L3}
         Port [P+1] := 0;    {select CR3, LSB CR2=0}
         Port [P]   :=$B0;   {T3 in single shot mode, 16 bit}
         Port [P+1] :=$95;   {T2 in continuous mode,8 bit,
                              select CR1 with LSB CR2=1}
         Port [P]   :=$B0;   {T1 in single shot mode,
                              all timers in operation}
         P:=P+8;
       UNTIL P > BaG1b + $28;
     END; {with}

END;  {timerinitialization}

{-------------------------------------------------------------------}

PROCEDURE settimers (crank_position,speed_control : BYTE;
                     motor_control : INTEGER;
                     pwm_flag,display : BOOLEAN);

  {This procedure is called by procedure start_stimulation. Its
   purpose is the to access the values stored in the look up tables
```

```
referring to the crank_position and to multiply them with the
control variables speed_control and motor_control. If the flag
pwm_flag is TRUE the pulse width values of the look up tables are
multiplied with the speed_control, otherwise the amplitude values
are multiplied with the speed_control !

NOTE that to gain execution speed the data ports are addressed
immediate. Thus all these addresses have to be changed if the
base address of the stimulation waveform generation board is
changed !!!

This Procedure takes on the IBM (ie. 4.77 MHz clock) 6.42 ms and
on the SPERRY  (7.16 MHz clock) 4.59 ms }

VAR
   i,j,
   amp_speed_control,
   pwm_speed_control    :BYTE;
   tempCH1,tempCH2,tempCH3,tempCH4,tempCH5,tempCH6 : BYTE;

BEGIN

   Port [$8FD] := 255;  {setting D/A 2 for timing measurement}

   IF pwm_flag  THEN
      BEGIN
         amp_speed_control := 255;
         pwm_speed_control := speed_control;
      END  {end of IF}
    ELSE
      BEGIN
         amp_speed_control := speed_control;
         pwm_speed_control := 255;
      END; {enf of ELSE}

    WITH patient DO
      BEGIN
        Port [$8F0] := Hi(pos_amplitude [1,crank_position]
                                        *amp_speed_control);
        Port [$8F1] := Hi(neg_amplitude [1,crank_position]
                                        *amp_speed_control);
        Port [$8F2] := Hi(pos_amplitude [2,crank_position]
                                        *amp_speed_control);
        Port [$8F3] := Hi(neg_amplitude [2,crank_position]
                                        *amp_speed_control);
        Port [$8F4] := Hi(pos_amplitude [3,crank_position]
                                        *amp_speed_control);
        Port [$8F5] := Hi(neg_amplitude [3,crank_position]
                                        *amp_speed_control);
        Port [$8F6] := Hi(pos_amplitude [4,crank_position]
                                        *amp_speed_control);
        Port [$8F7] := Hi(neg_amplitude [4,crank_position]
                                        *amp_speed_control);
        Port [$8F8] := Hi(pos_amplitude [5,crank_position]
                                        *amp_speed_control);
        Port [$8F9] := Hi(neg_amplitude [5,crank_position]
                                        *amp_speed_control);
        Port [$8FA] := Hi(pos_amplitude [6,crank_position]
                                        *amp_speed_control);
        Port [$8FB] := Hi(neg_amplitude [6,crank_position]
                                        *amp_speed_control);

     { amp_speed_control       = 0    ---> min. amplitude,
       amp_speed_control       = 255  ---> max. amplitude }
```

Listing of FNS-2.PAS-include file STIMULAT.OVL

```
 { S E T   T I M E R - C H I P   C H A N N E L  1 }

Port [$8C2] :=    msb_t1 [1,crank_position];
Port [$8C3] := Hi(lsb_t1 [1,crank_position]*pwm_speed_control);
Port [$8C6] :=    msb_t3 [1,crank_position];
tempCH1     := Hi(lsb_t3 [1,crank_position]*pwm_speed_control);
Port [$8C7] := tempCH1;

 { S E T   T I M E R - C H I P   C H A N N E L  2 }

Port [$8CA] :=    msb_t1 [2,crank_position];
Port [$8CB] := Hi(lsb_t1 [2,crank_position]*pwm_speed_control);
Port [$8CE] :=    msb_t3 [2,crank_position];
tempCH2     := Hi(lsb_t3 [2,crank_position] * pwm_speed_control);
Port [$8CF] := tempCH2;

 { S E T   T I M E R - C H I P   C H A N N E L  3 }

Port [$8D2] :=    msb_t1 [3,crank_position];
Port [$8D3] := Hi(lsb_t1 [3,crank_position] * pwm_speed_control);
Port [$8D6] :=    msb_t3 [3,crank_position];
tempCH3     := Hi(lsb_t3 [3,crank_position] * pwm_speed_control);
Port [$8D7] := tempCH3;

 { S E T   T I M E R - C H I P   C H A N N E L  4 }

Port [$8DA] :=    msb_t1 [4,crank_position];
Port [$8DB] := Hi(lsb_t1 [4,crank_position] * pwm_speed_control);
Port [$8DE] :=    msb_t3 [4,crank_position];
tempCH4     := Hi(lsb_t3 [4,crank_position] * pwm_speed_control);
Port [$8DF] := tempCH4;

 { S E T   T I M E R - C H I P   C H A N N E L  5 }

Port [$8E2] :=    msb_t1 [5,crank_position];
Port [$8E3] := Hi(lsb_t1 [5,crank_position] * pwm_speed_control);
Port [$8E6] :=    msb_t3 [5,crank_position];
tempCH5     := Hi(lsb_t3 [5,crank_position] * pwm_speed_control);
Port [$8E7] := tempCH5;

 { S E T   T I M E R - C H I P   C H A N N E L  6 }

Port [$8EA] :=    msb_t1 [6,crank_position];
Port [$8EB] := Hi(lsb_t1 [6,crank_position] * pwm_speed_control);
Port [$8EE] :=    msb_t3 [6,crank_position];
tempCH6     := Hi(lsb_t3 [6,crank_position] * pwm_speed_control);
Port [$8EF] := tempCH6;

Port [$8FC] := 255 - HI (motor_control
                         * set_motor_voltage[crank_position]);
IF brake_drive[crank_position] THEN Port [$77E] := $10
                 {set bit C4 of Port C shaftencoder PPI}
   ELSE Port [$77E] := 0;

IF display THEN
  BEGIN
    SetColorBlack;
    DrawPoint (500,90-oldCH1_Glb);      {plotting the new values}
    DrawPoint (524,90-oldCH2_Glb);      {of neg_pulsewidth to the
    DrawPoint (548,90-oldCH3_Glb);      {screen}
    SetColorWhite;
    DrawPoint (500,90-tempCH1);     {this IF block, or        }
    DrawPoint (524,90-tempCH2);     {the following ELSE block }
    DrawPoint (548,90-tempCH3);     {takes about 3 ms on the  }
    oldCH1_Glb := tempCH1;              {the SPERRY PC }
    oldCH2_Glb := tempCH2;
```

```
            oldCH3_Glb := tempCH3;
          END
        ELSE
          BEGIN
            SetColorBlack;
            DrawPoint (572,90-oldCH4_Glb);
            DrawPoint (596,90-oldCH5_Glb);
            DrawPoint (620,90-oldCH6_Glb);
            SetColorWhite;
            DrawPoint (572,90-tempCH4);
            DrawPoint (596,90-tempCH5);
            DrawPoint (620,90-tempCH6);
            oldCH4_Glb := tempCH4;
            oldCH5_Glb := tempCH5;
            oldCH6_Glb := tempCH6;
          END;    {if and else}

        Port [$8FD] := 0;  {resetting D/A 2 for timing measurement}

      END; {with}

  END; {procedure settimers}

{-------------------------------------------------------------------}

    {   main  body of the overlay start_stimulation }

BEGIN
  SetAspect(1);
  Timerinitialisation;
  ClearScreen;
  SetWindowModeOff;
  radius := radiusGlb + 10;

  {The following lines set up the screen for real time stimulation.
   Note that the window mode is switched off to increase display
   speed (ie. drawing takes place in screen coordinates !) }


 {draw circle and labelling for indication of crank position}

  DrawLine (xCircleGlb-radius-3,yCircleGlb,xCircleGlb-radius+3,
                                                    yCircleGlb);
  DrawLine (xCircleGlb+radius-3,yCircleGlb,xCircleGlb+radius+3,
                                                    yCircleGlb);
  DrawCircle (xCircleGlb,yCircleGlb,radius);
  DrawText (xCircleGlb-radius-20,yCircleGlb,1,'0');
  DrawText (xCircleGlb+radius+12,yCircleGlb,1,'180');
  radius := ROUND (radius*AspectFactor);
  DrawLine (xCircleGlb,yCircleGlb-radius+4,xCircleGlb,
                                    yCircleGlb-radius-2);
  DrawLine (xCircleGlb,yCircleGlb+radius+2,xCircleGlb,
                                    yCircleGlb+radius-4);
  DrawText (xCircleGlb-10,yCircleGlb+radius+5,1,'270');
  DrawText (xCircleGlb-25,yCircleGlb-radius-7,1,'90 degree');

  WITH patient DO
    BEGIN
      DrawLine (320,90,320,40);  {motorpower Bargraph-display}
      DrawLine (320,90,324,90);
      DrawLine (320,65,324,65);
      DrawLine (320,40,324,40);
      DrawText (285,30,1,'Motorpower');
      Str (max_motor_power/4:3:0,temp_string);
```

```
DrawText (278,40,1,temp_string + ' W');
Str (-max_motor_power/4:4:0,temp_string);
DrawText (278,90,1,temp_string +' W');
DrawText (290,65,1,' 0 W');
DrawText (280,55,1,'Drive');
DrawText (280,75,1,'Brake');

DrawLine (320,168,320,104);  {patient_speed_control Bargraph}
DrawLine (320,168,325,168);  {0 tick}
DrawLine (320,152,323,152);  {1/4 tich}
DrawLine (320,136,325,136);  {1/2 tick}
DrawLine (320,120,323,120);  {3/4 tick}
DrawLine (320,104,325,104);  {full tick}
DrawText (283,104,1,'Full');
DrawText (290,120,1,'3/4');
DrawText (290,136,1,'1/2');
DrawText (290,152,1,'1/4');
DrawText (290,168,1,' 0');
DrawText (275,174,1,'Speed Control');
DrawText (275,181,1,'   Patient');

DrawLine (440,160,440,110);  {operator_FNS_control Bargraph}
DrawLine (440,160,445,160);
DrawLine (440,148,443,148);
DrawLine (440,135,445,135);  {scale ticks}
DrawLine (440,123,443,123);
DrawLine (440,110,445,110);
DrawText (403,110,1,'Full');
DrawText (410,123,1,'3/4');
DrawText (410,135,1,'1/2');
DrawText (410,148,1,'1/4');
DrawText (410,160,1,' 0');
DrawText (395,168,1,' FNS  Control');
DrawText (395,175,1,'   Operator');
Drawtext (400,182,1,'+ = 8 ! - = 2');
DrawPoint (455,160);

DrawLine (490,90,490,30);        {pulsewidth Bargraph}
DrawLine (490,90,495,90);
DrawLine (490,45,495,45);
DrawLine (490,56,494,56);
DrawLine (490,68,495,68);
DrawLine (490,79,494,79);
DrawText (497,95,1,'L1  L2  L3  R1  R2  R3');
Drawtext (520,32,1,'Pulse-width');
DrawText (467,90,1,' 0 ');
DrawText (467,68,1,'150');
DrawText (467,45,1,'300');
DrawText (467,35,1,'us');

DrawLine (400,90,400,40);    {Speed Bargraph}
DrawLine (400,90,405,90);
DrawLine (400,78,403,78);
DrawLine (400,65,405,65);
DrawLine (400,53,403,53);
DrawLine (400,40,405,40);
DrawText (370,30,1,'Speed [rpm]');
DrawText (380,40,1,'60');
DrawText (380,53,1,'45');
DrawText (380,65,1,'30');
DrawText (380,78,1,'15');
DrawText (380,90,1,' 0');

DrawLine (520,160,520,110); {Motor_control Bargraph for Operator}
DrawLine (520,160,525,160);
DrawLine (520,148,523,148);
```

```
DrawLine (520,135,525,135);
DrawLine (520,123,523,123);
DrawLine (520,110,525,110);
DrawText (490,160,1,' 0');
DrawText (490,148,1,'1/4');
DrawText (490,135,1,'1/2');
DrawText (490,123,1,'3/4');
DrawText (490,110,1,'Full');
DrawText (500,168,1,'  MOTOR');
DrawText (500,175,1,'  CONTROL');
Drawtext (500,182,1,'+ = 9 | - = 3');

IF flag_pwm_control THEN
  BEGIN
    DrawText (580,110,1,'AMPLITUDES');
    FOR i := 1 TO chHalfGlb DO
      BEGIN
        str(i:1,temp_string);
        DrawText (580,120+(i-1)*8,1,'L'+temp_string+' = ');
        DrawText (580,150+(i-1)*8,1,'R'+temp_string+' = ');
        str(ROUND(neg_amplitude[i,0]/2.55):3,temp_string);
        DrawText (605,120+(i-1)*8,1,temp_string+' mA');
        str(ROUND(neg_amplitude[i+chGlb,0]/2.55):3,temp_string);
        DrawText (605,150+(i-1)*8,1,temp_string+' mA');
      END;
  END;

IF flag_amp_control THEN
  BEGIN
    DrawTex (560,110,1,'PULSE WIDTHS');
    FOR i := 1 TO chHalfGlb DO
      BEGIN
        str(i:1,temp_string);
        DrawText (580,120+(i-1)*8,1,'L'+temp_string+' = ');
        DrawText (580,150+(i-1)*8,1,'R'+temp_string+' = ');
        str(ROUND(pulsewidth[i,0]*pwGlb):3,temp_string);
        DrawText (605,120+(i-1)*8,1,temp_string+' us');
        str(ROUND(pulsewidth[i+chGlb,0]*pwGlb):3,temp_string);
        DrawText (605,150+(i-1)*8,1,temp_string+' us');
      END;
  END;

puts (30,1,'Stimulate  Patient  ');
Write (name);

f := ROUND(1E6/((msb_t2[1,0]+1) * (lsb_t2[1,0]+1) * Tclk2Glb));
str (f:3,temp_string);
DrawText (20,140,1,'Frequency = '+ temp_string +' Hz');

IF flag_biphasic THEN DrawText (20,148,1,'Biphasic')
   ELSE DrawText (20,148,1,'Monophasic');
str (ROUND(lsb_t2[1,0]*Tclk2Glb):4,temp_string);

DrawText (20,156,1,'Inter pulse interval = '+temp_string+' us');
str (neg_pos_ratio:3:2,temp_string);
DrawText (20,164,1,'neg_pos_ratio = '+temp_string);

puts (5,25,'Press  SPACEBAR  to QUIT');
puts (10,10,'Crank position');


{presetting all the variables}

ch := 'x';
Pcount := 0;
voltageSum := 0;
```

```
currentSum := 0;
voltage := 0;
current := 0;
power := 0;
oldpower := 0;
patient_speed_control := 255;
operator_FNS_control := 0;

{reset old channel pulsewidths}
oldCH1_Glb := 0; oldCH2_Glb := 0; oldCH3_Glb := 0;
oldCH4_Glb := 0; oldCH5_Glb := 0; oldCH6_Glb := 0;

speed_control  := 0;
crank_position := 0;
motor_control  := 0;
old_crank_position := 0;
powerfactor := 0.5;
display := FALSE;

{ the execution times mentioned in the following reffer to the
  SPERRY PC (7.16 MHz), execution times on the standard IBM PC
  are 1.4 times longer ! the following REPEAT - UNTIL block takes
  with-out the procedure settimers (which executes in 4.59 ms)
  4.2 ms; The ELSE block of the motor power calculation adds
  1.6 ms every 8 loop cycles. At crankpositions between 240 and
  255 the calculation and plotting of the crankspeed takes
  another 1.2 ms. Thus the worst case execution time (ie. the
  time between updating of the stimulation parameters) is
  4.59 + 4.2 + 1.6 + 1.2 = 11.59 ms, ie. 86 Hz update rate !
     Every IF ch = 'x' Test takes 100 us, so additional control,
  eg. amplitude control of each channel via the keyboard, is
  easily possible and does not slow down the execution speed
  significantly !
     The timing of the REPEAT - UNTIL block can be checked by
  viewing the signal at pin 9 of the 15-D connector of the Stimu-
  lation board; This signal is high for the time the procedure
  settimers is executed and it is low for the time the rest of
  the  loop is busy. }


REPEAT

  getkey(ch);       {returns a character if a key is pressed}


  IF (Port [$77E] AND $01) = 0 THEN  {Check whether the patient}
                  flag_stop := TRUE  {presses his button}
    ELSE flag_stop := FALSE;



  IF ch = '8' THEN                  { increase FNS control}
    BEGIN
      SetColorBlack;
      DrawPoint (455,160-0.196*operator_FNS_control);
      IF operator_FNS_control<246 THEN
         operator_FNS_control:=operator_FNS_control+10
        ELSE operator_FNS_control := 255;
      SetColorWhite;
      DrawPoint (455,160-0.196*operator_FNS_control);
    END;

  IF ch = '2' THEN                  {decrease FNS control}
    BEGIN
      SetColorBlack;
      DrawPoint (455,160-0.196*operator_FNS_control);
```

```
        IF operator_FNS_control >9 THEN
          operator_FNS_control := operator_FNS_control-10
          ELSE operator_FNS_control := 0;
        SetColorWhite;
        DrawPoint (455,160-0.196*operator_FNS_control);
      END;

    IF ch ='9' THEN                    {increase motor control}
      BEGIN
        SetColorBlack;
        DrawPoint (535,160-0.196*motor_control);
        IF motor_control<245 THEN motor_control:=motor_control+10
          ELSE motor_control := 254;
        SetColorWhite;
        DrawPoint (535,160-0.196*motor_control);
      END;

    IF ch='3' THEN                     {decrease motor control}
      BEGIN
        SetColorBlack;
        DrawPoint (535,160-0.196*motor_control);
        IF motor_control>9 THEN motor_control:=motor_control-10
          ELSE motor_control := 0;
        SetColorWhite;
        DrawPoint (535,160-0.196*motor_control);
      END;

    IF (ch=#27) AND Keypressed  THEN        {display patient file}
      BEGIN                                 {and switch stimulation off}
        READ (kbd,ch);
        IF Ord(ch)=60 THEN
          BEGIN
            old_speed_control := speed_control;
            old_motor_control := motor_control;
            FOR i := 1 TO 8 DO
              BEGIN
                speed_control := speed_control SHR 1;
                motor_control := motor_control SHR 1;
                settimers(crank_position,speed_control,
                          motor_control,flag_pwm_control,display);
                delay (100);
              END;
            display_patientfile;
            TimerInitialisation;
            FOR i := 1 TO 8 DO
              BEGIN
                speed_control := old_speed_control SHR (8-i);
                motor_control := old_motor_control SHR (8-i);
                settimers(crank_position,speed_control,
                          motor_control,flag_pwm_control,display);
                delay(100);
              END;
          END;
        ch := 'x';  {clear ch}
      END;

    SetColorBlack;
    DrawPoint (xCirclePointGlb[crank_position],
                          yCirclePointGlb[crank_position]);
    DrawPoint (335,168-(patient_speed_control SHR 2));

    crank_position := Port[BaShftEncGlb];

{-------------------------------------------------------------------}

    {patient_speed_control is measured at channel 9 of the
```

```
        A/D board;    0 V --> + 1.7 V  =  brake;  (0V = full brake)
                    + 2.1 V --> + 4.6 V  =  drive;  (4.6V = full drive)
        + 2.1 V  =  860; + 4.6 V = 1883; difference = 1023;
        i.e. subtract 860 from adsample(9) and devide by 4, in order
        to get a result between 0 and 255 for patient_speed_control )

{-------------------------------------------------------------------}

        patient_speed_control := adsample(9);
        IF patient_speed_control<860 THEN patient_speed_control:=860;
        patient_speed_control := (patient_speed_control-860) SHR 2;
        IF patient_speed_control>255 THEN patient_speed_control:=255;

        speed_control := HI ((patient_speed_control)*
                             operator_FNS_control);

        SetColorWhite;
        DrawPoint (xCirclePointGlb[crank_position],
                                   yCirclePointGlb [crank_position]);
        DrawPoint (335,168-(patient_speed_control SHR 2));

{-------------------------------------------------------------------}

      { The following IF-THEN-ELSE block calculates the power which
        drives the patient or the power with which the patient gets
        loaded .

        Measurement of the actual motor voltage takes place
        via the channel 8 of the A/D board :

        Because of hardware reasons the voltage coming from the PWM
        motor controller depends on whether the motor drives, or
        brakes the patient :

        if the motor drives the patient (ie.brake_drive[crank_position]
        is TRUE ) :
        Voltage at channel 8 = (Vbattery - Vmotor)/5;
        thus if Vbattery = 24 V,

        Vmotor =  24 - (Voltage at channel 8)*5;    (I)

        if the patient drives the motor (ie. brake_drive[crank_position]
        is FALSE) :
        Voltage at channel 8 = Vmotor / 5;
        thus Vmotor = (Voltage at channel 8) * 5;

        Samples of the motor voltage are averaged over 8 cycles of the
        loop. individual samples are added to the variable VoltageSum
        if the motor drives the patient, or subtracted if the patient
        drives the motor (ie. negative power means the patient creates
        energy which is dissipated in a resistive load !)

        The A/D board is in monopolar mode, thus 1 Volt gives 409.5
        as the result of the A/D conversion.

        Thus the equation (I) can be rewritten:

        Vmotor := 1966 - adsample(8);

        where 1966 = 409 * 24V/5; and Vmotor is the motor voltage in
        A/D format (ie. 0 = 0 V and 1966 = 24 V as motor voltage;

        Motor current is measured via channel 7 of the A/D converter;
        A voltage of 0.2 V at this input is equal to a current of 1 A;
        Thus, assuming a maximal current of 5A the maximal result of
        the A/D conversion is 409
```

```
max. VoltageSum = 1966  * 8 = 15728 (24 V max motor voltage)
max. CurrentSum = 409 * 8 = 3276  (assuming 5A as max. current)
Power is the product of VoltageSum and CurrentSum, ie.
maximal 15728 * 3276 = 51 524 928, this is the equivalent of
5 A * 24 V = 120 Watts; therefore we only have to devide the
product of VoltageSum and CurrentSum by 51 524 928/120 =
429 374 to get the power in watts !
To avoid overflow during multiplication of the integer
variables VoltageSum and CurrentSum, VoltageSum is devided by
128, and CurrentSum by 16 (ie.SHR 4); The result of the
multiplication of VoltageSum and CurrentSum must be devided
by 429374/(128*16) = 209.655; It is faster, however to multiply
the result by 1/209.655 = 0.00476974 instead !

THUS
power := 0.00476974 * (VoltageSum DIV 128)*(CurrentSum SHR 4); }


IF Pcount < 8 THEN
  BEGIN
    IF brake_drive[crank_position] THEN
      BEGIN
        temp := 1966 - adsample(8);
        IF temp>0 THEN voltageSum := voltageSum+temp;
        currentSum := currentSum+adsample(7);
      END
    ELSE
      BEGIN
        voltageSum := voltageSum - adsample(8);
        currentSum := currentSum + adsample(7);
      END;
    Pcount := Pcount + 1;
  END
 ELSE                  { the ELSE block takes 1.6 ms longer }
                       { than the IF block }
   BEGIN
     power:=ROUND(0.00476974*((voltageSum DIV 128)*
                  (currentSum SHR 4)) -(0.326*speed-1.81));

            {the term 0.326*speed-1.81 represents the power
             which is neccessary to overcome the friction of
             the chain and gearbox ! The values were obtained
             experimentally.
             +-25 W give +-25 vertical plotpoints }

     SetColorBlack;
     DrawPoint (335,65-(oldPower*powerfactor));
     SetColorWhite;
     DrawPoint (335,65-(power*powerfactor));
     oldpower := power;
     IF brake_drive[crank_position] THEN
       BEGIN
         temp := 1966 - adsample(8);
         IF temp>0 THEN voltageSum := temp;
         currentSum := adsample(7);
       END
      ELSE
       BEGIN
         voltageSum := - adsample(8);
         currentSum := adsample(7);
       END;
     Pcount := 0;
   END;  {if Pcount < 8  & else}
```

```
          IF crank_position IN [240..255] THEN   {ie. after passing the }
            BEGIN                                { zero degree mark     }
              IF Port [BaShftEncGlb+1] > 0 THEN  {3000/port[speed] gives}
                BEGIN                            {the speed in rpm      }
                  SetColorBlack;
                  DrawPoint (415,90-speed*0.833);
                  speed := ROUND (3000 DIV Port [BaShftEncGlb+1]);
                  {ie. speed in rpm}
                  SetColorWhite;
                  DrawPoint (415,90-speed*0.833);
                  {i.e. 50 y-pixels are 60 rpm}
                END
               ELSE                                { this IF - ELSE  block takes }
                BEGIN                              { 1.2 ms to execute }
                  SetColorBlack;
                  DrawPoint (415,90-speed);
                  SetColorWhite;
                  DrawPoint (415,90);
                  speed := 0;
                END;
            END;

            IF display THEN display := FALSE
              ELSE display := TRUE;

            m := HI (motor_control * patient_speed_control);

            IF flag_stop THEN
                settimers (crank_position,0,0,flag_pwm_control,display)
              ELSE
                settimers(crank_position,speed_control,m,
                          flag_pwm_control,display);

        UNTIL (ch = ' ') OR (ch = 'q');

    END;  {with}

    resetD_A;          {switching the stimulation off}

    ClearScreen;
    ch := ' ';
    SetWindowModeOn;
    SetClippingOn;

END;  {overlay procedure start_stimulation}

{ ************** End of Real Time Stimulation Module ************}


{ ***************  End Of Overlay Procedures Section  ***************}

                { The overlay section started with
                  overlay procedure intro. ie. all
                  the procedures in between are
                  stored  in  the  overlay - file
                          FNS-2.000 }
```

```
{$I editfile.pas}     {file handling for input stimulation}

{include file  editfile.pas}

          {***********************************************
           *                                             *
           *    Module for handling of patient files     *
           *         for editing or stimulation          *
           *                                             *
           *          Matthias H. Popp   7/86            *
           *                                             *
           ***********************************************}


PROCEDURE Stimulate;

{This procedure is called by the main program body of FNS-2.pas .
 It promts the already selected file and asks the operator to
 specify a new file or to press ENTER for using the already selected
 patient file for stimulation. If a new file is specified the already
 selected file is stored on disk.}


VAR   done,setupWindow : BOOLEAN;
        temp_file_name : string_20;
                length : INTEGER;
BEGIN
  Port [BaShftEncGlb+3] := $93; {initialisation  shaftencoder interface}
                                {port A,B,C0-C3 input, port C4-C7 output}
  Port [BaADGlb+3] := $92;        {initialisation  PC-26 A/D  board}

  ClearScreen;
  DefineHeader (1,'Stimulation of the patient');
  SelectWindow(1);
  SetBackGround (0);
  DrawBorder;
  centre (4,'Stimulate Patient');
  RestoreWindow(16,0,30);                {display Directory window}
  done        := FALSE;
  setupWindow := FALSE;
  commandLine(6);
  puts(20,14,'Selected  patient   is : ');
  length := pos ('.',file_nameGlb) - 3;
  Write (Copy(file_nameGlb,3,length));
  Write ('      (drive ',logged_driveGlb,')');
  puts(20,16,'Enter new patient name : ');
  temp_file_name := readstr (8);
  temp_file_name := Copy (temp_file_name,1,8);
  ch := Copy (temp_file_name,1,1);
  IF (ch='q') OR (ch='Q') THEN done:=TRUE;
  IF (ch <> ' ') AND (ch<>'q') THEN
    BEGIN
      Assign (patientfile,logged_driveGlb+temp_file_name+'.fns');
      {$I-}
      Reset(patientfile);
      {$I+}
      IOCheck;
      IF IOValGlB = 0 THEN            {ie. the selected file exists}
        BEGIN
          IF NOT flag_first_fileGlb THEN store_patient;
          setupWindow := TRUE;
          file_nameGlb := temp_file_name;
          File_nameGlb := Concat (logged_driveGlb,File_nameGlb,'.FNS');
        END;
    END;
```

```
  IF (IOValGlB = 0) OR ((ch=' ') AND (NOT flag_first_fileGlb))  THEN
    BEGIN
      WHILE  done = FALSE  DO
        BEGIN
          IF ch<>' ' THEN                 {ie. only a new file is read}
            BEGIN
              Assign (patientfile,file_nameGlb);
              Reset(patientfile);
              READ(patientfile,patient);
              Close(patientfile);
            END;
          WITH patient DO
            BEGIN
              IF setupWindow THEN              {setting up a patientfile}
                BEGIN                         {only if a new file is read}
                  chHalfGlb := ch_max DIV 2;
                  initialize_window_patientfile;
                  updateWindow_patientfile(1);
                  IF flag_pwm_control THEN updateWindow_patientfile(2);
                  IF flag_amp_control THEN updateWindow_patientfile(3);
                  updateWindow_patientfile(4);
                END;
              DefineHeader (2,'Stimulate patient '+name);
              SelectWindow (2);
              REPEAT
                SetBackGround (0);
                DrawBorder;
                commandLine(3);
                centre (6,'Functional  Neuromuscular  Stimulation');
                puts (25,10,'1  -  Electrode Impedance  Check');
                puts (25,12,'2  -  Start Stimuation');
                puts (25,14,'3  -  Calibrate Impedance Check');
                puts (25,16,'Q  -  QUIT');
                REPEAT
                  READ(kbd,ch);
                UNTIL ORD(ch) IN [49..51,59,60,81,113];
                CASE ORD(ch) OF
                  49 : impedance_check;
                  50 : Start_stimulation;
                  51 : calibrate_impedance_check;
                  60 : display_patientfile;
                  59 : displayhelp (7);
                END; {case}
              UNTIL upcase(ch)='Q';
              SetBackGround (0);
            END; {with}
          done := TRUE;
        END; {while}
      END;  {if IOValGlB=0}
    ch := ' ';

END; {procedure Stimulate}

{*****************************************************************}

PROCEDURE edit_oldfile;

{This procedure is called by procedure inputStimulationParameters.
 It allows changes to be made on the data in a selected patient
 file. The patient file display window is updated after changes have
 been made. The look up tables are recalculated before edit_oldfile
 exits to procedure inputStimulationParameters }


VAR  done : BOOLEAN;
       i : INTEGER;
```

```
BEGIN
  done := FALSE;
  DefineHeader(1,'Change   Stimulation   Parameters');
  SelectWindow(1);
  SetHeaderOn;
  SetBackGround (0);
  DrawBorder;
  commandLine(3);
  centre (4,'Change  Stimulation Parameters');
  puts (26,8,'1  -   Stimulation Waveform ');
  puts (26,10,'2  -   Stimulation Frequency');
  puts (26,12,'3  -   Pulse-Width Envelope ');
  puts (26,14,'4  -   Channel Amplitudes   ');
  puts (26,16,'5  -   Motor-voltage Envelope ');
  puts (26,18,'6  -   Copy and rename opened file');
  puts (26,22,'ENTER YOUR CHOICE     !      ');
  StoreWindow(8);   {stores this menu}
  WHILE done = FALSE DO
    BEGIN
    RestoreWindow(8,0,0);
    REPEAT
      READ(kbd,ch);
    UNTIL ORD(ch) IN [49..54,59,60,81,113];
    WITH patient DO
     BEGIN
      CASE ORD(ch) OF
        49  : BEGIN
                waveform;
                updateWindow_patientfile(1);
              END;
        50  : BEGIN
                frequency;
                updateWindow_patientfile(1);
              END;
        51  : BEGIN
                pulse_width;
                IF flag_pwm_control THEN updateWindow_patientfile(2)
                  ELSE updateWindow_patientfile(1);
              END;
        52  : BEGIN
                amplitude;
                IF flag_amp_control THEN updateWindow_patientfile(3)
                  ELSE updateWindow_patientfile(1);
              END;
        53  : BEGIN
                motorvoltage;
                updateWindow_patientfile(1);
                updateWindow_patientfile(4);
              END;
        54  : BEGIN
                namefile(TRUE);     {True --> copy and rename}
                updateWindow_patientfile(1);
              END;
        59  : displayHelp(2);
        60  : display_patientfile;
        81  : done := TRUE;
        113 : done := TRUE;
      END; {case}
     END; {with}
  END;   {while}
  calculate_newfile;
END;     {Procedure edit_oldfile}

{*****************************************************************}
```

```
PROCEDURE inputStimulationParameters;

{This procedure is called by the main body of the program FNS-2.PAS.
 It asks the operator to specify a file he wants to edit or create.
 If the file does not exist the operator is asked whether he wants
 to create a new file. If so the operator is automatically guided
 through all the stimulation parameter defining procedures. These
 are : 1. namefile,  2. waveform,  3. frequency,  4. pulse_width,
       5. amplitude,  6. motorvoltage  ;
 After going through all these procedures the procedure edit_oldfile
 is called to allow for changes to be made on the entered data. }

VAR
  done,flag_return,setupWindow     : BOOLEAN;
  temp_file_name : string_20;
          length : INTEGER;

BEGIN
  ClearScreen;
  DefineHeader(1,'Input Stimulation Parameters');
  SetHeaderOn;
  SelectWindow(1);
  puts(25,4,'Input Stimulation Parameters');
  DrawBorder;
  commandLine(12);
  RestoreWindow(16,0,50);                  {display Directory window}
  done         := FALSE;
  flag_return := FALSE;
  setupWindow := FALSE;

  REPEAT
    puts(20,14,'Selected  patient   is : ');
    length := Pos ('.',file_nameGlb)-3;
    Write (Copy(file_nameGlb,3,length));
    Write ('       (drive ',logged_driveGlb,')');
    puts(20,16,'Enter new patient name : ');
    temp_file_name := readstr (8);
    temp_file_name := Copy (temp_file_name,1,8);
    IF temp_file_name = 'q' THEN done := TRUE;
    IF (temp_file_name <>' ') AND (temp_file_name<>'q') THEN
      BEGIN
        Assign (patientfile,logged_driveGlb+temp_file_name+'.fns');
        {$I-}
        Reset(patientfile);
        {$I+}
        IOCheck;
      END;

    WHILE done = FALSE DO
      BEGIN
      IF IOValGlb = 1 THEN              {ie. the file does not exist}
        BEGIN
          flag_newfileGlb:=TRUE;
          puts(20,18,'Create a new file   y/n  ?');
          READ(kbd,ch);
          IF upcase(ch) = 'Y' THEN
            BEGIN
              IF NOT flag_first_fileGlb THEN  {the file in memory is }
                                 store_patient;  {stored because a new  }
                                                 {file is created}
              file_nameGlb := temp_file_name;
              file_nameGlb:=Concat(logged_driveGlb,file_nameGlb,'.FNS');
              initialize_window_patientfile;
              namefile(FALSE);
              WITH patient DO
```

```
                BEGIN
                  FillChar(stored_plotarray,SizeOf(stored_plotarray),0);
                                {resetting plotarrays}
                  waveform;
                  frequency;
                  flag_amp_control := TRUE;
                  updateWindow_patientfile (1);
                  pulse_width;
                  IF flag_pwm_control THEN updateWindow_patientfile (2)
                    ELSE updateWindow_patientfile (1);
                  amplitude;
                  IF flag_amp_control THEN updateWindow_patientfile (3)
                    ELSE updateWindow_patientfile (1);
                  motorvoltage;
                  updateWindow_patientfile (1);
                  updateWindow_patientfile (4);
                  flag_newfileGlb := FALSE;
                END; {with}
              edit_oldfile;        {the entered data can be edited}
              done := TRUE;
              flag_first_fileGlb := FALSE;
            END  {IF}
          ELSE
            BEGIN
            done        := TRUE;
            flag_return := TRUE;
            END;
        END       {if IOVal=1}

      ELSE        {ie.  the selected file already exists}
        BEGIN
        IF temp_file_name<>' ' THEN
          BEGIN
            IF NOT flag_first_fileGlb THEN   {the file in memory is }
                              store_patient;   {stored because a  file}
                                               {will be loaded from disk}
            file_nameGlb := temp_file_name;
            file_nameGlb := Concat (logged_driveGlb,file_nameGlb,'.FNS');
            Assign (patientfile,file_nameGlb);
            Reset(patientfile);
            READ(patientfile,patient);
            Close(patientfile);
            setupWindow := TRUE;
          END;
        IF setupWindow THEN
          BEGIN
            WITH patient DO        {setting up a patientfile window}
              BEGIN
                chHalfGlb := ch_max DIV 2;
                initialize_window_patientfile;
                updateWindow_patientfile(1);
                IF flag_pwm_control THEN updateWindow_patientfile(2);
                IF flag_amp_control THEN updateWindow_patientfile(3);
                updateWindow_patientfile(4);
              END; {with}
          END;
        flag_newfileGlb := FALSE;
        edit_oldfile;   {procedure call edit_oldfile}
        done        := TRUE;
        flag_first_fileGlb := FALSE;
        END;  {IF}

    END;  {while}

  UNTIL done = TRUE;
  ch := ' ';
```

```
    {file handling for input stimulation}
                    {parameters and envelopes}

{*****************************************************************}

{      ----------> M A I N      P R O G R A M <---------      }


BEGIN
  InitGraphic;
  resetD_A;
  file_nameGlb := '';
  beep;
  SetBreakOn;
  SetMessageOn;
  SetHeaderOn;
  DefineWorld (2,0,199,639,0);              {defined for intro}
  DefineWindowIBM  (10,0,0,XMaxGlb,YMaxGlb);  {defined for intro}
  intro;            {modified intro from Borlands' tgdemo.pas}
  ResetWindows;
  ResetWorlds;
  ClearScreen;
  puts(20,12,'Busy  initializing   the  System  !!');

  WITH patient DO
    BEGIN
      FillChar (patient,SizeOf(patient),0);
    END; {with}
  system_driveGlb := DefaultDrive;
  logged_driveGlb := 'b:';          {get current logged drive for
                                     FNS data storage}
  dir_maskGlb := '????????.fns';    {definition of directory mask}
  defWindows;                       {Procedure to define all used windows}
  getDir(dir_maskGlb);
  ClearScreen;
  DefineHeader (1,'M A I N     M E N U');
  SelectWorld(1);
  SelectWindow (1);
  SetBackGround(0);
  DrawBorder;
  commandLine(0);          {procedure to display commandline0 in line 25}
  puts(10,4,' U C T     P A R A - C Y C L E   F N S    S T I M U L A T O R');
  puts(25,10,'1  -  Input Stimulation Parameters');
  puts(25,12,'2  -  Stimulate Patient');
  puts(25,14,'3  -  Change Logged Drive (currently ');
  StoreWindow(7);
  REPEAT
    RestoreWindow (7,0,0);
    GotoXY(62,14);
    Write (logged_driveGlb:2,')');
    puts (30,18,'ENTER  YOUR  CHOICE');
    REPEAT
      READ (kbd,ch);
    UNTIL ORD(ch) IN  [49..51,59,60,81,113];
    CASE ORD(ch) OF
      49 : BEGIN
             inputStimulationParameters;
             GetDir(dir_maskGlb);
           END;
      50 : BEGIN
             stimulate;
             getDir (dir_maskGlb);
           END;
```

```
        51 : BEGIN
               change_drive;
               getDir(dir_maskGlb);
             END;
        59 : displayhelp(1);
        60 : displayDirectory(0,25);
      END;
      IF upcase(ch) = 'Q' THEN
        BEGIN
          beep;
          centre (21,'Do you really want to quit this program  Y/N ?');
          REPEAT
            READ (kbd,ch);
          UNTIL upcase(ch) IN ['Y','N'];
          IF upcase(ch) = 'Y' THEN ch := 'Q' ELSE ch := ' ';
        END;
    UNTIL (upcase(ch) = 'Q');
    IF NOT flag_first_fileGlb THEN store_patient;
    LeaveGraphic;
END.
```

REFERENCES

BAJD T, KRALJ A
1982a
Gait synthesis in paraplegic patients
World Conference on Medical Physics and Biomedical Engineering, Hamburg

BAJD T, KRALJ A, KRAJNIK J, TURK R, BENKO H, SEGA J
1984
Standing by FES in paraplegic patients.
Proceedings 8th. International Symposium on External Control of Human Extremities, Dubrovnik 51-61

BAJD T, KRALJ A, TURK R, BENKO H, SEGA J
1982b
A four-channel electrical stimulator enabling the walking of paraplegic patients
Journal Physical Therapy

BAJD T, KRALJ A, TURK R, BENKO H, SEGA J.
1983
The use of a four-channel electrical stimulator as an ambulatory aid for paraplegic patients.
Physical Therapy 63 7 1116-1120

BAKER LL
1981
Neuromuscular electrical stimulation in the rehabilitation of purposeful limb functions
Electrotherapy (Wolf SL (ed)) New York Churchill Livingstone

BAKER LL, MCNEAL DR, WONG T, SMITH D
1986
The effect of duty cycle on muscle fatigue using a stimulation frequency of 100 pulses per second
In Proceedings of RESNA 9th Annual conference Minneapolis, Minnesota 1986 : 276-278

BARTO P, GRONLEY J, PERRY J, YOSHIDA H
1985
Correlation of clinical profile with gait parameters in the SCI population : a basis for FES patient selection
Proceedings 8th Annual Conference on Rehabilitation Engineering Memphis 111-113

BENTON LA, BAKER LL, BOWMAN BR, WATERS RL
1981
Functional Electrical Stimulation - A practical Clinical Guide - Second edition, Rancho Los Amigos Rehabilitation Center, Rancho Los Amigos Hospital, Downey, California

BOGARTAJ U, KLJAJIC M, STANIC U, ACIMOVIC R, GROS N
1984
Gait pattern behaviour of hemiplegic patients under the influence of a 6 channel microprocessor stimulator in a real environment
Proceedings 2nd International Conference on Rehabilitation Engineering Ottawa 529-530

BORLAND INTERNATIONAL Inc.
1985a
Turbo Pascal Reference Manual, version 3.0
Borland International Inc. 4585 Scotts Valley Drive, Scotts Vally, CA 95066

BORLAND INTERNATIONAL Inc.
1985b
Turbo Graphix Toolbox, Owner's Handbook, version 1
Borland International Inc. 4585 Scotts Valley Drive, Scotts Vally, CA 95066

BOWMAN BR, BAKER LL
1985
Effects of waveform parameters on comfort during transcutaneous neuromuscular electrical stimulation
Annals of Biomedical Engineering 13 59-74

BOWMAN BR, MEADOWS P
1983
Effect of stimulation pulse duration on comfort in controlled motor contractions.
Rehabilitation Research & development progress reports 51

BOXTEL A VAN
1977
Skin resistance during square-wave electrical pulses of 1 to 10 mA
Medical & Biological Engineering & Computing 15 679-687

BRANDELL BR
1982
Development of a universal control unit for functional electrical stimulation (FES).
American Journal Physical Medicine 61 6 279-301

BRENNEN KR
1976
The characterization of transcutaneous stimulating electrodes
IEEE Transactions on Biomedical Engineering 23 4 337-340

CHIZECK HJ, MARSOLAIS EB, KOBETIC R
1984
Closed-loop controller design of neuro-prosthetics for functional walking in paralyzed patients
9th World Congress of IFAC

DAY GA, ANDREWS B, PAUL JP
1983
The development of a hybrid programmable electro-stimulator for research applications
Proceedings 1st Vienna International Workshop on FES Vienna

DESIPRES M.
1974.
An electromyographic study of competitive cycling conditions simulated on a treadmill.
pp. 349-355 In: Nelson RC, Morehouse CA, eds, Biomechanics IV.
Baltimore: University Park Press.

DIMITRIJEVIC MR, LARSSON LE.
1981
Neural control of gait: clinical neurophysiological aspects.
Applied Neurophysiology 44 152-159

EDWARDS RHT, YOUNG A, HOSKING GP, JONES DA.
1977
Human skeletal muscle function: description of tests and normal values.
Clinical Science and Molecular Medicine 52 283-290

EICHHORN KF, SCHUBERT W, DAVID E
1984
Maintenance, training, and functional use of denervated muscles
Journal of Biomedical Engineering 6 : 205-211

ERICSON MO, NISELL R, ARBORELIUS UP, EKHOLM J.
1985.
Muscular activity during ergometer cycling.
Scandinavian Journal of Rehabilitation Medicine, 17(2):53-61.

FARIA I, CAVANAGH P.
1978.
The physiology and biomechanics of cycling.
New York: John Wiley and sons.

GANONG WF
1977
Review of medical physiology
LANGE Medical publications, Los Altos, California 94022

GLASER RM, GRUNER JA, FEINBERGER SD, COLLINS SR.
1983
Locomotion via paralysed leg muscles : feasibility study for a leg-propelled vehicle.
Journal of Rehabilitation Research & Development, 20 1 10-38

GORMAN PH, MORTIMER JT,
1983
The effect of Stimulus Parameters on the Recruitment Characteristics of Direct Nerve Stimulation
Transactions on Biomedical Engineering, BME-30 (7)

GRACANIN F, TRNKOCZY A.
1975
Optimal stimulus parameters for minimum pain in the chronic stimulation of innervated muscle.
Archives Physical Medicine & Rehabilitation 56 June 243-249

GRANDJEAN PA, MORTIMER JT
1982
Muscle force modulation with subfascial monopolar and bipolar electrodes
IFAC: Control aspects of Prosthetics & Orthotics, Ohio 17-23

GUYTON AC
1982
Human Physiology and mechanisms of disease
W.B. Saunders Company : Philadelphia,PA USA

HEINEMANN AW, MAGIERA-PLANEY R, GIMENES MG
1985
Evaluating the special needs of functional neuromuscular stimulation research candidates
Journal Medical Engineering & Technology 9 4 167-173

HENDERSHOT DM, PETROFSKY JS, PHILLIPS CA
1985
Blood pressure and heart responses in paralyzed and nonparalyzed man during isokinetic leg training
Journal Neurological & Orthopaedic Medicine & Surgery 6 3 259-264

HOUTZ SJ, FISCHER F.
1959
Analysis of muscle action and joint excursion during exercise on a stationary bicycle.
Journal of Bone and Joint Surgery, 41A(1):123-131.

HULL ML, JORGE M.
1985
A method for biomechanical analysis of bicycle pedalling.
Journal of Biomechanics, 18(9):631-644.

IEC
1977
Safety of medical electrical equipment, Part 1 : General requirements
Bureau Centrl de la Commission Electrotechnique Internationale, 1, rue de Varembe' , Geneve, Suisse

IBM
1983
Personal Computer Hardware Reference Library
First edition (revised 1983) International Business Machines Corporation

KHOSLA SV, PERKASH I
1983
Functional electrical stimulation : Review and recent applications.
Journal American Paraplegia Society 6 4 89-93

KOBETIC R, VANCE F, MARSOLAIS EB
1984
32-channel portable programmable stimulator for walking of paraplegics
Proceedings 2nd International Conference Rehabilitation Engineering, Ottawa 533-534

KOLB P, HEFFTNER G
1986
Shaft encoder interface for the IBM Personal Computer
Internal report Department Biomedical Engineering, Univerisity of Cape Town

KRALJ A, BAJD T, TURK R
1980
The influence of electrical stimulation on muscle strength and fatigue in paraplegia
Proceedings International Conference on Rehabilitaion Engineering, Toronto 223-226

KRALJ A, JAEGER R.
1983
Standing by FES in paraplegia.
American Spinal Injury Association, Annual Meetin Chicago

KRALJ A, VODOVNIK L
1977a
Functional electrical stimulation of the extremities : Part 1
Journal Medical Engineering and Technology January 12-15

KRALJ A, VODOVNIK L
1977b
Functional electrical stimulation of the extremities : Part 2
Journal Medical Engineering and Technology March 75-80

KRALJ A, BAJD T, TURK R, KRAJNIK J, BENKO H
1983
Gait Restoration in Paraplegic Patients : A Feasibility Demonstration using Multichannel Surface Electrode FES
Journal of Rehabilitation Research and Development Vol. 20. No 1, 1983, 3-20

MARSOLAIS EB, KOBETIC R
1983
Functional walking in paralysed patients by means of electrical stimulation.
Clinical Orthopaedics 175 30-36

MARSOLAIS EB, KOBETIC R, CHIZECK H
1984a
Standing of paraplegics using closed-loop controlled stimulation
Proceedings 8th International Synposium on External Control of Human Extremeties, Dubrovnik 63-66

MARSOLAIS EB, KOBETIC R, VANCE F,CHIZECK H
1984b
Paraplegic walking with electrical stimulation
10th American Scientific meeting, Abstracts Houston 49-52

MCMIKEN DF, TODD-SMITH M, THOMPSON C.
1983
Strengthening of human quadriceps muscles by cutaneous electrical stimulation.
Scandinavien Journal Rehabilitation Medicine 15 25-28

MCNEAL DR, BAKER LL
1985
Stimulating the quadriceps and hamstrings with surface electrodes
RESNA 8th Annual Conference Memphis 237-239

MCNEAL DR, MEADOWS MS
1984
Selection of stimuli for functional control of extremities
IEEE Frontiers of Engineering and Computing in Health Care 1984, 841-846

MEADOWS PM, MCNEAL DR
1984
A laboratory FES system for modulated control of the lower extremities
Proceedings 8th International Symposium on External Control of Human Extremities, Dubrovnik 101-111

MILLMAN J, TAUB H
1965
Pulse, Digital and Switching waveforms
McGraw-Hill book company, Inc. New York, p. 64-82

MORSE PS
1982
The 8086/8088 Primer
(second edition)
Hayden Book Company,Inc; USA

MOTOROLA
1979
MC6840 Programmable timer, fundamentals and applications
NMOS Microcomputer Systems Applications Austin, Texas

NATHAN RH
1983
Fatigue of electrically stimulated muscle
Proceedings 1st Vienna International Workshop on FES Vienna 1983

NAUMAN S, WIFSUD M, CAIRNS BJ, MILNER M
1985
Dual channel electrical stimulators for use by children with diplegic spastic cerebral palsy
Medical & Biological Engineering & Computing, September 1985: 435-444

NEUMAN MR
1978
Biopotential electrodes
p. 215-272, In: Medical Instrumentation, Application and design; Editor Webster JG, Houghton Mifflin Company, Boston USA

PETROFSKY JS, GLASER RM, PHILLIPS CA, Gruner JA
1982
The effect of electrically induced bicycle ergometer exercise on blood pressure and heart rate
The Physiologist 25 253

PETROFSKY JS, HEATON HH, PHILLIPS CA
1983
Outdoor bicycle for exercise in paraplegics and quadriplegics.
Journal of Biomedical Engineering 5 4 292-296

PETROFSKY JS, HEATON HH(III), PHILLIPS CA.
1984b.
Leg exerciser for training of paralysed muscle by closed-loop control.
Medical and Biological Engineering and Computing, 22:298-303.

PETROFSKY JS, PHILLIPS CA, ALMEYDA J, BRIGGS R.
1985a
Aerobic trainer with physiological monitoring for exercise in paraplegic and quadriplegic patients.
Journal of Clinical Engineering 10 4 307-31

PETROFSKY JS, PHILLIPS CA, HEATON HH
1984a
Feedback control system for walking in man.
Computers in Biology and Medicine 14 2 135-149

PETROFSKY JS, PHILLIPS CA, HEATON HH, GLASER RM
1984c
Bicycle ergometer for paralysed muscles
Journal of Clinical Engineering 9 1 13-19

PETROFSKY JS, PHILLIPS CA, HENDERSHOT D
1985b
The cardiorespiratory stresses which occur during dynamic exercise in paraplegics and quadriplegics
Journal Neurological & Orthopaedic Medicine & Surgery 6 3 252-258

PETROFSKY JS, PHILLIPS CA, PETROFSKY SH
1985c
Electronic physicians' prescription system for functional electrical stimulation (FES) patients
Journal Neurological & Orthopaedic Medicine & Surgery 6 3 239-246

PHILLIPS CA, PETROFSKY JS
1985
A review of fracture cases in spinal cord injured individuals participating in closed-loop functional electrical stimulation (FES) experiments
Journal Neurological & Orthopaedic Medicine & Surgery 6 3 247-251

POON CW, Ko WH, PECKHAM PH, MCNEAL DR
1981 An implantable RF-powered dual channel stimulator
Biotelemetry Patient Monitoring 8 180-188

PONS D
1986
The Paracycle : Rehabilitation of paralyzed patients using Functional Neuromuscular Stimulation.
Cape Town : Department of Biomedical Engineering, Cape Town, MSc. Dissertation

POPP HM
1986
The PC-26 12-bit A/D converter card for the IBM Personal Computer Data sheet
Eagle Electric Co. (PTY.) LTD. Electronic Devision, P.O. Box 3106, Cape Town, South Africa

SMITH B, BRANWELL SD, BUCKETT JR, JACKSON KC
1984
A portable microprocessor controlled functional neuromuscular stimulation system
Proceedings 2nd International Conference on Rehabilitation Engineering, Ottawa 531-532

STEFANOVSKA A, VODOVNIK L, LIBEJ J, KOJOVSEK M
1983
Comparison of effects of sinusoidal and rectangular electrical stimulation currents in strengthening the quadriceps muscles
Proceedings 3rd Meditterranian Conferencec Medical & Biological Engineering

STREETER LA, CHIZECK HJ, KOBETIC MS
1985
Input-output response of quadriceps muscle in paraplegic patients
RESNA 8th Annual Conference on Rehabilitation Techniques Memphis 240-242

STROJNIK P, KRALJ A, URSIC I.
1979
Programmed six-channel electrical stimulator for complex stimulation of leg muscles during walking.
IEEE Transactions on Biomedical Engineering BME 26, No 2, 112-116

THOMA H, KERN H, BOCHDANSKY T, FREY M, HOLLE J
1983
Muscle training of paraplegic patients by FES device
Proceedings 1st Vienna International Workshop on FES Vienna

THROPE GB, PECKHAM PH, CRAGO PE
1985
A computer-controlled multichannel stimulation system for laboratory use in functional neuromuscular stimulation
IEEE Transactions on Biomedical Engineering 32 6 363-370

TURBO POWER SOFTWARE
1985
Turbo Extender Software package
Turbo Power Software Inc. California, USA

TURK R, KRALJ A, BAJD T, STEFANCIC M, BENKO H.
1980
The alteration of paraplegic patients muscle properties due to electrical stimulation exercising.
Paraplegia 18 386-391

VANCE F, KOBETIC R, MARSOLAIS EB, CHIZECK HJ
1983
Portable microprocessor-controlled stimulator for activation of paralyzed muscles
23rd International Symposiun on Mini & Micro-computers San Antonio 101-105

VODOVNIK L, BOWMAN BR, HUFFORD P
1984
Effects of electrical stimulation on spinal spasticity
Scandinavian Journal Rehabilitation Medical 16 1 29-34

WALMSLEY RP, LETTS G, VOOYS J
1984
A comparison of torque generated by knee extension with a maximal voluntary contraction vis-a-vis electrical stimulation.
Journal of Orthopaedic & Sports Physio Therapy 6 1 10-17

WATERS RL, MCNEAL DR, PERRY J
1975
Experimental correction of footdrop by electrical stimulation of the peroneal nerve.
Journal Bone Joint Surgery (Am) 57 8 1047-1054

YOUNG JL, MAYER RF
1981
Physiological properties and classification of single motor unit activity by intramuscular microstimulation in the first dorsal interosseous muscle in man
Progres in clinical Neurophysiology Vol 9, Editor Desmond JE. p. 17-25