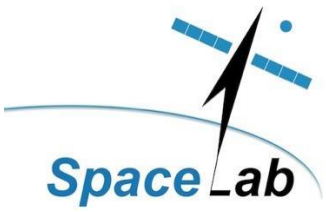

# FIRMWARE DESIGN AND TESTING FOR DATA HANDLING AND CONTROL OF TIME DELAY INTEGRATION IMAGE SENSORS IN AN EARTH OBSERVATION PAYLOAD

**Muhammad Ebtisam Ahmed**




This dissertation is submitted in partial fulfilment of the requirements for

the degree of Master of Philosophy in Space Studies

June 2022

SL20-10M

# DECLARATION

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. It has not been previously submitted, in part or whole, to any university of institution for any degree, diploma, or other qualification. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed Signed by candidate Date15 June 2022

Muhammad Ebtisam Ahmed, MPhilSpace Studies

University of Cape Town

# ABSTRACT

Earth observation is becoming more and more important in this age of information. It provides critical information in every field of work be it military, scientific research, disaster management, urban planning and many others. Better planning in all these fields is possible with more detailed information which translates to a requirement of high-resolution Earth observation data. These high-resolution images tend to generate a lot of data which needs to be processed in a very short time. This high data rate problem is the focus of this work. This work tries to tackle this problem in two ways. Firstly, a Time Delay Integration (TDI) image Sensor is used which drastically improves image Signal to Noise Ratio (SNR) and decreases the amount of data generated. Secondly, an FPGA is used for data handling and processing of this generated data. The large number and high bandwidth of FPGA inputs/outputs enables it to handle huge data generated. Functional analysis and requirements of firmware design will be analyzed based on previous works addressing a similar problem. Based on that, requirements will be proposed. The firmware design of the FPGA is proposed, which drives firmware architecture. This architecture serves as a guideline for detailed firmware design which will be done in VHDL. This firmware design will be tested as per requirements and presented in this work.

# ACKNOWLEDGEMENTS

It was memorable day of my life when I arrived at university of Cape Town, Menzies building for registration in the M.Phil Space Studies Program, where I was warmly welcomed during registration process in department of electrical engineering by Professor Peter Martinez.

I would like to pay my special thanks Professor Peter Martinez for all his efforts during my M.Phil studies course work as well as in this thesis. It was not possible to complete my thesis without his extra ordinary support during this time

Finally, I would to thank to all my family membersespecially my parents, wife and kids as well as my friends,who supported me during my studies and stay in Cape Town.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

| | |
|---|---|
| ADC | Analogue to Digital Convertor |
| AOCS | Attitude and Orbital Control System |
| BIC | Bus Interface Card |
| BRAM | Block Random Access Memory |
| BUFR | Buffer |
| CCD | Charge Coupled Device |
| CDS | Correlated Double Sampling |
| CLKDIV | Clock Divider |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPHA | Phase |
| CPOL | Polarity |
| CRC | Cyclic Redundancy Check |
| CS | Chip Select |
| DAM | Data Manipulation |
| DCLK | Divided Clock |
| DDLY | Delayed Input |
| DDR | Dual Data Rate |
| DN | Digital Number |
| DRDY | Data Ready |
| EMI | Electro-Magnetic Interferance |
| EO | Electro-Optical |
| FIFO | First In First Out |
| FPGA | Field Programmable Gate Arrays |
| FW | Firmware |
| GPS | Global Positioning System |
| GSD | Ground Sampling Distance |
| HDL | Hardware Description Language |
| IBUFDS | Differential Input Buffer |
| IC | Integrated Circuit |
| IDELAYCTRL | Delay Control Block |

| | |
|---|---|
| IO | Input Output |
| IOBDELAY | Input Output Delay Buffer |
| IP | Intellectual Property |
| LUT | Look Up Table |
| LVDS | Low Voltage Differential Signal |
| MISO | Master In Slave Out |
| MMCM | Mixed Mode Clock Manager |
| MOSI | Master Out Slave In |
| MRCC | Multi Region Clock Capable |
| MS | Multispectral |
| NA | Not Applicable |
| NUM_CE | Control Enable Numeral |
| OBC | On Board Computer |
| OBDH | On Board Data Handling |
| PAN | Panchromatic |
| PCB | Printed Circuit Board |
| PGA | Programmable Gain Amplifier |
| PLL | Phase Locked Loop |
| PPS | Pulse Per Second |
| PWR | Power |
| RAM | Random Access Memory |
| REFCLK | Reference Clock |
| RF | Radio Frequency |
| RST | Reset |
| SCL | SPI Clock |
| SDA | SPI Data |
| SDR | Single Data Rate |
| SEN | Sensor Handling |
| SERDES | Serializer/Deserializer |
| SIC | Sensor Interface Card |
| SME | Space Mission Engineering |
| SNR | Signal to Noise Ratio |

| | |
|---|---|
| SPF | Single Point Failure |
| SPI | Serial Peripheral Interface |
| SRCC | Single Region Clock Capable |
| TBC | To Be Confirmed |
| TBD | To Be Decided |
| TDI | Time Delay Integration |
| TMT | Represents TMTC |
| TMTC | Telemetry/ Telecommand |
| VAR | Variable |
| VHDL | Very High-Speed Integrated Circuit Hardware Descriptive Language |
| VIO | Virtual Input Output |
| XXX | Represents Top Level Function |
| YYY | Represents Requirement Number |

# 1 INTRODUCTION

Earth observation is one of the major applications of satellites aiding the growth of civilization. Users of Earth observation often work only with satellite images but a number of systems are used in this process. When a request from auser comes, the first step is to plan the optimum operations procedure to image the specific area of interest. This includes a step by step procedure for the satellite to get ready for imaging including battery charging with solar arrays, planning the time suitable for imaging with respect to satellite position and lighting of the intended target area, checking health telemetry of the satellite and payload, to name but a few. This is done by the Mission Planning Center as depicted in Figure 1-1. This procedure is transferred to the satellite by Command and Control Center, which communicates it to satellite via RF link. In the satellite, these commands are communicated to the Onboard Data Handling (OBDH) Unit which disseminates the commands to different units of the satellite. Every satellite has two basic systems, spacecraft and payload. This work focusses on a small part of the payload.

With enough power and correct pointing/position, the OBDH commands the payload to start imaging. An Electro-Optical (EO) payload consists of four distinct units based on functionalities which are Structural, Optics, Opto-mechanical and Electronics units. This work focuses on a part of the Electronics unit. The Electronics unit also has different entities with varying functions including structure, PCB cards and firmware. This work focusses completely on firmware of payload responsible for data handling and control of image sensors.

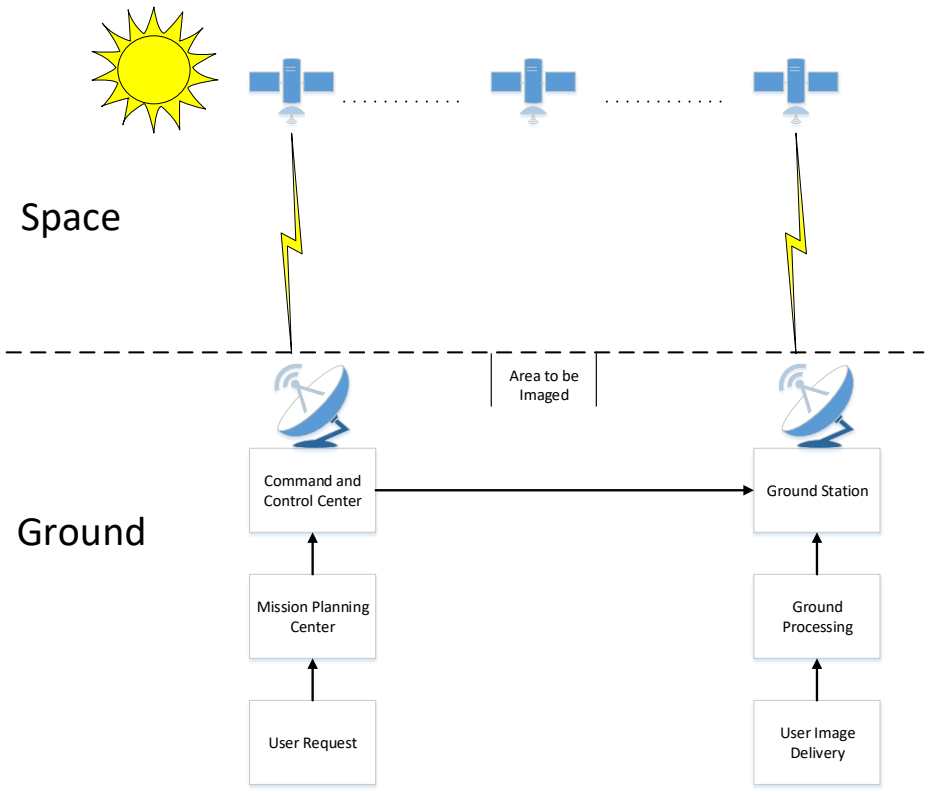



**Figure 1-1** Satellite Control and the Earth Observation Process.

Upon reception of the command to start imaging, the firmware controls the image sensor to start image acquisition. This data is processed and sent to the spacecraft for storage. When the satellite reaches within range of a ground station, it start transmission of the stored image as programmed by the Command and Control Center. Ground processing is later done after acquisition to correct the image with respect to position information. Then this image is delivered to the user.

It is difficult to increase all these resolutions at the same time but the definition of low, medium and high resolution is being elevated each year. What was considered high resolution a decade ago can now barely qualify for medium resolution. Just like processing power, a significant improvement in image sensor technology is also seen. With the large number of bands in small image sensors and incorporation of Time Delay Integration (TDI), it is possible to achieve spatial and radiometric resolution with mediocre spectral resolution in a single payload. This empowers the payload to acquire maximum information of the image scene generating a lot of data. It is critical to pass on this data before the arrival of next GSD data to ensure no data loss. Thus, it poses a real time, high data rate engineering problem which will be tackled in this work.

A Field Programmable Gate Array (FPGA) provides an ideal solution for this problem. It provides parallel capabilities often coupled with a large number of inputs and outputs with a lot of bandwidth. This is often required for high-resolution payload data handling. Furthermore, it also provides high performance at low power and size with easier reconfiguration[1], which makes it ideal for satellite usage.

Another important aspect of FPGA is scalability[2].Unlike normal processors which are limited by the number of cores or thread handling, FPGAs are scalable creating as many cores as possible, which can be of different size based on the complexity required. It is only limited by the available FPGA resources which is often but not limited to number of Look-Up Tables (LUTs). Thus the FPGA provides a unique and powerful ability to adapt as per requirements rather than requirement compromise based on hardware. This also ensures easier future upgradability.

FPGAs have a strong space heritage from interplanetary space probes to Earth remote sensing missions like those that we focus on this work. It has been used in variety of missions ranging from Cubesats[3], smallsatellites[4] and high-resolution missions[5]. It has also been used extensively for geostationary communication satellites[6]. A successful moon mission[7]has also been done with FPGA processing power.

## 1.1 Aims and Objectives

The primary objective of this work is to provide a high data rate link between image sensors and the bus interface with minimal data loss. Satellite imagers use line sensors and the frame rate of these line sensors is very fast, especially for high-resolution satellites. In a matter of microseconds, the data handling system should not only transmit the data to the bus, but must be ready to receive next swath of data.

Once a satellite is launched, it is virtually impossible to repair. Thus high reliability is required in the design. Redundancy is an important consideration for reliability improvement. However, it’s impossible to incorporate redundancy in every component and subsystem, especially due to cost considerations. The high cost component varies from mission to mission and for an Earth remote sensing satellite, it’s often the payload.

A system is said to be Single Point Failure (SPF) free if it’s fully redundant. Some missions make a cost-reliability compromise and thus tailor the definition of SPF. The

Earth remote sensing payload considered in this work is SPF free, with SPF defined as any single part or unit failure which would cause more than $1/6^{th}$ swath width reduction. To achieve this, a modular design approach is adopted. As per this philosophy, the design is broken down into separate smaller components that can be developed separately and work independent of each other. The overall design of the system is given in Figure 1-2.

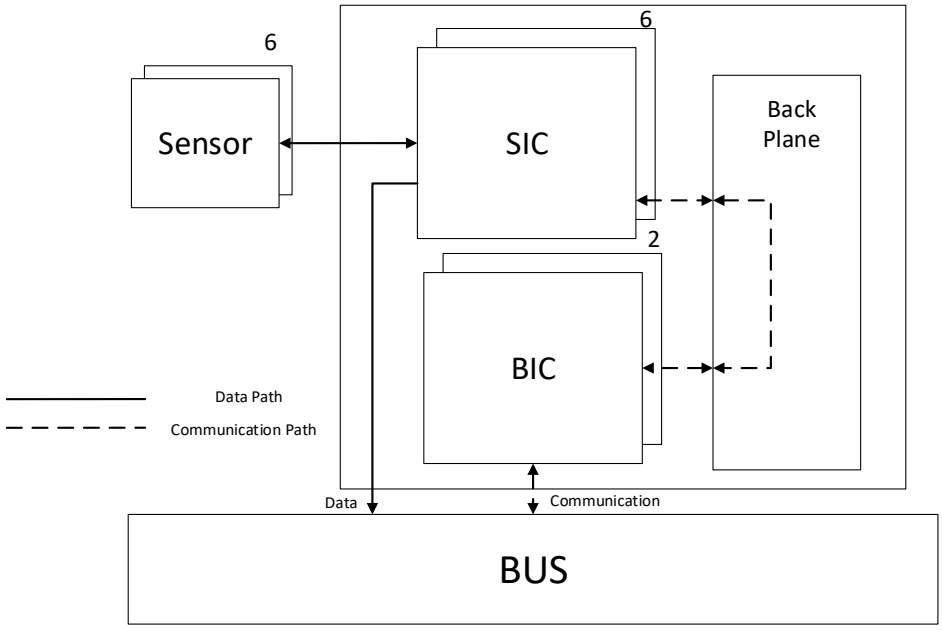



**Figure 1-2** Payload Design Overview.

It can be seen fromFigure 1-2 that even if one sensor chain is facing problems, the operation of other sensors will not be affected. This is in accordance with the SPF requirement. A modular approach not only provides an easier development path, but any card is also easier to replace in case of a problem. One such case can be malfunction of a particular hardware component near the launch date. As launch dates are very strict and often depend on more than one variables, that hardware can be replaced with a module instead of complete set replacement. This is particularly important in shared launches, which is a growing trend these days.

However, a satellite or satellite component failure is not an easy replacement depending on where the failure is encountered. For instance, a card cannot be easily replaced if it is a component of a payload that has been subjected to thermal vacuum and EMI/EMC testing. At a minimum the card replacement would require some subset of payload hardware testing that in all cases would require removal from the satellite and in all

likelihood, removal from the launch vehicle. An approach would be an in-line replacement payload. This is often driven by system reliability to cost curve, a full consideration of which is beyond the scope of this work.

## 1.2 Scope of Work

This work focuses on firmware development of the image Sensor Interface Card (SIC). The modular design philosophy is not limited to a top level cards. SIC firmware is subdivided into number of components until every component has a unique functionality set. This is also important from a testing point of view. Testing of one function which is independent of other functions is easier to manage and verify. This will also start the testing process early in the development phase providing useful inputs for development. Test cases for every single component are designed to simulate every possible input to that component. Testing of payload firmware design is a core part of this design process as it ensures design reliability in every possible scenario during the life time of a payload. The testing referred here is strictly functional simulation testing and does not involve any hardware testing.

Once functionality of all components is verified, then complete firmware integration is done. As detailed testing of every component is done already, only integration issues are highlighted in this process. Hence, the focus of this integration is not to debug functionality problems in a specific component, but to verify core functionalities of the complete SIC firmware.

## 1.3 Outline of Subsequent Chapters

Chapter 2 looks deeper into the what and how of firmware design based on previous published work. Multiple designs are reviewed and based on detailed analysis problem statements are extracted. Along with identification of problems, solution approaches are also discussed. These solutions will drive the baseline design which will be carried on in further chapters. It also discusses the firmware design process which is provided by Xilinx in a step by step fashion. The importance of following these guidelines is highlighted and the repercussions of haphazard development are highlighted. Then this chapter delves into coding style guidelines for providing reliable, configurable and manageable firmware development.

Chapter 3 starts with requirements imposed on firmware due to sensor architecture. This will clarify the data and control interfaces specific to the sensor which is used to communicate and control it. It then emphasizes functional analysis based on following a systematic iterative process. Requirements are highlighted for firmware design as an outcome of this process. These requirements are analyzed and any design considerations required will be fed back to requirements. Based on this iterative process, the final specifications of the firmware are documented in this chapter which will be followed in design. With a clear definition of requirements, output data rate budgets for the sensor and SIC are discussed in this chapter.

Coupled with sensor architecture and requirements, firmware architecture is proposed in Chapter 4. This firmware architecture serves as a foundation upon which the FPGA selection decision is made and discussed in detail. After finalization of the architecture, the designer is able to get a good grasp of the project. This will enable the designer to have an estimate of FPGA resources required. This chapter will also discuss number of FPGA parameters useful in analyzing the best suitable FPGA for this work. The FPGA selection process will also be highlighted in this chapter.

Chapter 5 discuss the development of subsequent components in detail. It discuss the inner working of components and the reasoning for selection of a particular approach. State diagrams of every component are discussed here, providing a useful insight into every component. It also highlights the specific implementation parameters used for configuration of built-in resources and IP cores provided by the manufacturer. Use of such rigorously tested resources simplifies the development process and increases the reliability of the final product.

Chapter 6 discusses the simulations required for detailed testing of the firmware design. It includes test case definition along with functional and timing simulation. It discusses the simulation results and required models used for simulation. The iterative simulation feedbacks into the design process and its benefits are highlighted. Finally, simulation of complete integrated firmware will be discussed and verified.

Chapter 7 presents our conclusions and makes recommendations for future work.

# 2 LITERATURE REVIEW

Every sensor pixel gets energy from sunlight reflected from a Ground Sampling Distance (GSD). For a high-resolution payload with correspondingly smaller GSD, reflected sunlight decreases. As a result the energy received for every pixel decreases. However, noise generated in each pixel is independent of resolution. Thus as resolution increases, the Signal to Noise Ratio (SNR) decreases. This presents one of the fundamental challenges to high resolution payloads.

To increase SNR, the accumulated energy needs to be increased without reduction in GSD. This is achieved by taking multiple samples of the same GSD area. These multiple samples are added to increase signal energy. The beauty of this process is in the randomness of noise. Addition of these pixels increase signal energy by N times while noise in the pixel is only increased by the square root of N where N is number of pixels used for addition. This process is called Time Delay Integration (TDI) because multiple samples are taken with delayed time. The number of stages used is limited by satellite stability and pixel readout time. Satellite stability is beyond scope of this work but readout time will be discussed here.

There are two methods by which this increase in SNR is realized; software TDI and hardware TDI. The former is done by reading multiple pixels and then adding them in software/firmware, while the latter is done within the sensor. Software TDI adds pixel digital values while hardware TDI adds the charge in subsequent pixels, reading out the resultant sum. Overall the effect is same with minor increased SNR with hardware TDI due to reduced readout and quantization noise. In addition to that, the large number of lines read for software TDI makes it slow. This limits the flight time (ratio of GSD and satellite ground speed) which in turn limits the resolution of a satellite.

Hardware TDI is done in theimage sensor, which is often called a TDI sensor. This type of sensor proves to be very beneficial as it doesn’t limit readout time. The accumulated charge from multiple stages is read only once at the end of the TDI stages. As TDI is being done for the upcoming GSD, the previous GSD pixel value is read out.

## 2.1 Algorithm Overview

To efficiently develop any system, algorithms entailing the system should be researched first. This work will make use of many such algorithms. As it was established in Chapter 1 that this work is an engineering problem, this problem has been tried before. The purpose of this section is to look into such solutions proposed by multiple engineers before this work. The following design chapters will tailor these findings to our desired specifications which will be implemented and analyzed. Based on the design approach followed here, major algorithms are discussed below.

### 2.1.1 Data Synchronization

Data synchronization refers to correct data reception between two systems which don't have a common sampling clock. Synchronization is done in two stages; bit alignment and word alignment. Bit alignment is achieved by taking data samples at the correct times forming a perfect an eye diagram[8]. It is formed by observing the repetition of a digital signal in an oscilloscope, also called eye pattern. The output of bit alignment is to ensure that every data sample is taken at the middle ofthe eye. The eye diagram can define a lot about a received signal:

- Too long/short
- Too high/low
- Synchronized/unsynchronized with clock
- Too noisy
- Overshoot/undershoot level

Data synchronization is done with transmission of a known pattern. For correct bit alignment, a data clock is required which is often provided by many digital systems like ADCs[9] and digital image sensors[10]. However, the data channels provided by these devices lag[10]with regard to the sensor clock or other data channels. In addition to different delays in data channels inherent by source, PCB design can also be a source of timing mismatch. As a result, the data received is not correct and requires separate time delay adjustment for different channels w.r.t. the data clock. Provision of an incorrect delay may result in data sample recovery which is actually intended for an adjacent clock cycle. Bit alignment is done when data received is constant for a particular time.

Another important aspect of synchronization is word alignment. Data is often provided by digital devices serially which is later converted to a parallel stream to process this data in real time. This parallel sequence of bits thus generated in a single time is referred as a word. The sequence of bits in the word may vary depending on the position of the first sample taken. As transmitted pattern is known, each received word is compared to that pattern. If the pattern is not matched, a circular shift is done until it is matched. This process is called word alignment. Source[10] often provides a digital sequence to enable efficient bit and word alignment.

### 2.1.2 Data Handling in Multiple Clock Domains

To switch data handling from one clock domain to another is a very sophisticated task and requires special attention to timing analysis. It can be done in logic, but this itself will be a project. It is easier to use IPs already provided by FPGA vendors to handle this. Payload firmware has to handle image data in at least three clock domains.

High-speed sensors provide data with multiple channels. The first clock domain for data handling is the frequency at which the sensor provides this data. Data is received on the data clock provided by the sensor. Firmware often needs to perform processing on it like binning and auxiliary data tagging. This requires a separate clock domain for data handling. A separate domain is required because this preprocessing often changes the amount of data. To efficiently handle this, a separate clock domain tailored to this processed data is used.

Finally this processed data is to be sent to the satellite bus. The reception speed of the bus can be separate from firmware internal handling and thus require another clock domain crossover for data. The data path is shown inFigure 2-1.

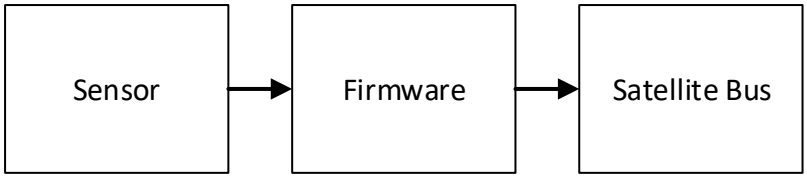



**Figure 2-1**Data handling in different clock domains

Several FPGA vendors provide an excellent First In First Out (FIFO) IP core which is ideal for this clock domain cross over[11]. The FIFO core provided by Vivado is adequate and it provides a useful mechanism to handle data in different clock domains

with independent read and write clocks. To compensate for the clock separation of Figure 2-1, at least two independent clock FIFOs will be required.

### 2.1.3 Sensor Clock Selection

The efficiency of the sensor depends critically on timing of the driving circuitry. CCD based sensors have three distinct timing requirements which are met by three separate clocks. These clocks correspondingly drive individual functionalities which define the working methodology of the sensor. A TDI sensor needs to accumulate charge from multiple acquisitions. Photons from the GSD are received and converted to electrons which are then stored. When one acquisition is done, the accumulated charge is transferred to the next pixel in the flight direction. This charge transfer is done with the help of the CCD clock. This clock is typically provided as an input to the sensor.

Earth observation satellites may see a lot of change in their altitude throughout their life. The leading reason is that most of the satellites are launched in elliptical orbits, thus their altitude change within an orbit. Apart from that, many orbital perturbations also result in slow altitude deterioration over the satellite's life time. Change in altitude, in any case, results in variation of satellite speed. This means that pixel time to acquire reflected energy is different with changing altitude. As GSD is a parameter of the end product which is a satellite image, it should remain constant and the end user should not worry about altitude variation. As the CCD clock is responsible for charge transfer and charge accumulation from incident light, it should be adjustable[12] to changing altitude.

Flight time of a pixel is defined as the time taken by a satellite to pass over one GSD. It can be calculated by dividing the satellite ground speed with GSD distance in the flight direction. This time typically ranges within a few hundred microseconds. Thus its frequency is typically a few kilohertz which makes it a slow clock. Sometimes, depending on the architecture, another CCD clock is also provided as an input to the CCD sensor. This second clock is needed to transfer accumulated charge within one line. This is typically required in sensors with a common ADC for all pixels in a line. This CCD clock rate is higher than the flight direction of the CCD, as it has to transfer charge of all pixels within one flight time. Although, this clock is faster than the first clock, it's still considered a slow clock and ranges from few tens to hundred kilohertz depending on resolution.

A Master clock is used to control other operations of a sensor. This includes sensor control operation, readout circuitry and facilitating the operation of ADCs. As these operations need to be done at a fixed rate, the master clock is fixed throughout the mission life time. Accumulated charge by CCD clock is converted to voltage by a stage normally referred to as a Charge to Voltage converter. Once charge is converted to voltage, it needs to be converted to a Digital Number (DN) which is done by ADC.

ADCs operating in CCD sensors also need a timing reference or operating clock. Mostly this timing reference can be deduced from the Master clock thus no additional clock is needed as input. Its operation is also dependent on sensor architecture and more specifically how many ADCs a sensor has. In section2.1.1, the idle packet concept is introduced which is used for data synchronization. As a CCD clock needs to be variable, thus data will not be provided to the ADC at a definite time. This makes ADC operation also variable which will make the sensor operation quite complex. The simpler solution is to realize that by changing number of idle packets transmitted, the Master clock can be kept constant. This makes the system more reliable.

### 2.1.4 Correlated Double Sampling

Before applying the accumulated charge to the Charge to Voltage converter, it is reset to generate a baseline value without any signal input. This reading acts as a reference for the actual reading and is termed as floating gate level. Then the accumulated charge for a pixel is given as an input to the converter which translates that accumulated charge to a voltage corresponding to the charge stored. This voltage in a pixel is often referred to as a video signal. It is important to note that the Charge to Voltage converter takes time to convert charge into voltage, much like a capacitor charging up. Thus the corresponding voltage applied to the ADC input depends on the time provided to settle this voltage. The time provided for floating gate signal should be equal to the video signal. This will ensure the same voltage buildup time for reset and video signal, thus ensuring correct processing.

The actual signal value resulted from pixel charge is then calculated by subtracting the video signal value from the floating gate value. This process is called Correlated Double Sampling[13] (CDS). Accumulation of charge in a floating gate and video signal also helps to smooth high frequency noise, making the resultant output more accurate.

### 2.1.5 Black Level Clamping

A sensor often provides black pixels for correct mapping of analogue output with incident light. Black pixels are carefully placed in a sensor to shield incoming incident light. With no incident light, the outcome of these pixels after the CDS process should be zero. However, this is not the case[14] as there is additional pixel noise involved. This implicates that similar noise must be present in all pixels irrespective of incident light. To correct this, black pixel value also called “dark noise” should be subtracted from every pixel output.

The analogue output of a CCD sensor is given in Figure 2-2. The black level will depend on a lot of factors and it is difficult to have a fixed black output level. Actual incident light value can be calculated by subtracting[15] black pixel CDS DN from active pixel CDS DN. This is an important part of data handling for any image sensor, CCD or CMOS. The subtraction part can be performed at ground or onboard for TDI based sensors but it must be done on board before application of software based TDI.

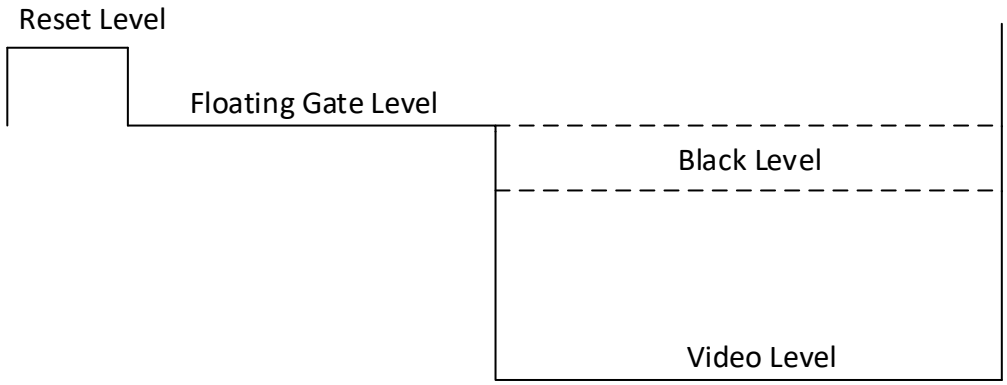



**Figure 2-2** CCD Pixel Analogue Output Waveform.

### 2.1.6 Timing Generator

CCD sensors are available with different architectures. The choice of architecture may vary from one application to another. Every architecture needs to provide a timing sequence[16] for driving a CCD sensor and processing CCD analogue output signal. Multiple architectures are described in that reference. The work referenced here is based on a hardware CCD sensor and thus it requires separate processing electronics. Timing generation is required for CCD sensor operations, including image transfer, readout, exposure, shuttering and reset, etc. Processing electronics performs operations like

digitization, CDS, black level clamping etc. For operation, it is required to control both sensor and processing electronics. Five building blocks of any CCD system are explained in the above referenced work, which is shown in Figure 2-3.

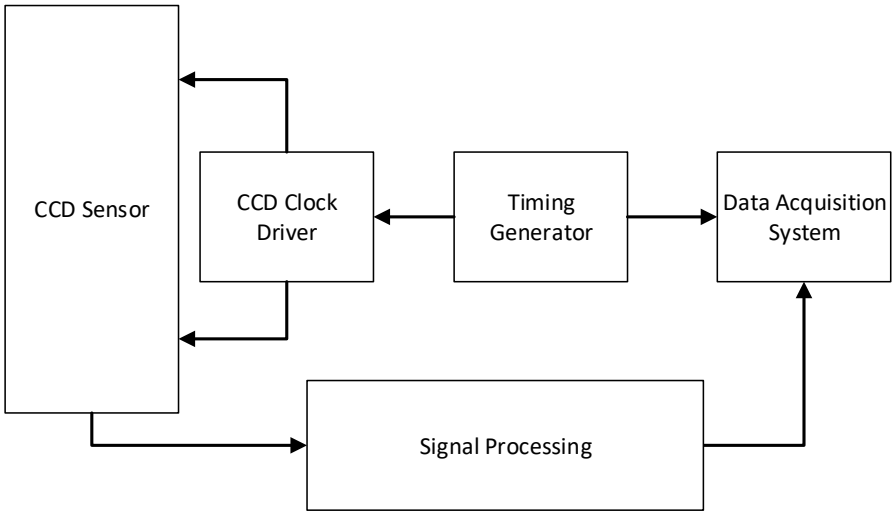



**Figure 2-3** Blocks of CCD Imaging System.

Clocking requirements of a CCD detector and processing electronics is also discussed in detail. Timing generator architecture is explained along with the timing requirement of every clock. Based on these requirements, clocks are either generated with PLL for tight tolerances or with counters for tolerances less than the flight time. The horizontal and vertical signals used for charge shifting, also called CCD clocks, are also discussed in section 2.1.3.

Special emphasis is needed for a trigger signal among timing generator signals. This sensor input controls the start of image acquisition. This is a repetitive pulse pattern and exposure time of pixels is encoded in the width of the trigger signal. This phenomenon is observed in multiple camera systems[17] and also in image sensors [18]. In CMOS[10] sensors, exposure time can be controlled by setting register values.

A camera, either with CMOS sensors or CCD sensors, generally involves multiple sensors or one long sensor with two or more separate controls. To get continuous swath, all sensors need to be triggered together. This principle is highlighted in the Master/Slave[19] Control architecture. Although, the image is taken with multiple

sensors, continuous swath is obtained as acquisition of all sensors starts at the same time. The Master will send a synchronization pulse to all slaves when triggering the sensors for data acquisition. Upon reception of this pulse, slave timing generators shall trigger their corresponding sensors. All the signals to be generated by timing generators can be seen in the references. Some pulse signals worth mentioning are Reset, Transfer Gate and phase signals.

### 2.1.7 TDI CCD Timing Generator

Most of the signals required for these sensors are thesame as for an area CCD sensor. The working of these signals, however, can be different for some signals. TDI CCD sensors are very efficient at producing much better SNR at low lighting conditions. This is done by increasing[20] integration time with help of additional TDI stages. It is also seen from this work that most of the timing signals required for TDI CCD sensors are the same as for area CCD sensor. This work uses the Dalsa IL-E2TDI-CCD sensor to generate requirements of timing generator and it is seen that timing generator requirements are highly dependent on image sensor. The architecture of this sensor is discussed in detail in this work. Signals generated for timing control includes Line Transfer Clock, Pixel shift clocks, Output reset clock, Image region clock and stage control clock.

The operation varies from the number of stages used but timing generator can be configured to all ninety six stages provided by this sensor. This sensor provides TDI in only one direction. Thus in designing the focal plane with this sensor, special care should be taken to orient the sensor in such a way that only one direction TDI is used. An important parameter that often limits the spatial resolution of a payload is the time taken to read pixel data. With a TDI CCD sensor, pixel readout time is considerably reduced. This is also a major difference from an area-CCD or matrix sensor and paves way towards a highresolution satellite.

### 2.1.8 Time Synchronization

Time tagging is important to provide information about imaging time. This imaging time is later correlated with satellite AOCS data to know satellite position and pointing. As the payload must incorporate this timing information in image data, it must be synchronized with satellite time. Moreover the payload provides data after every flight

time, which is very short. This provides the high time synchronization requirement on payload. This can be done by utilizing[21] message based or hardware based signals.

Message based time transmission involves a dedicated time synchronization network between different satellite units. It's a costly and complicated solution in terms of harnessing. However, it simplifies the unit operation. The hardware based signal approach requires every unit to sync its time with hardware tick provided at regular intervals. A simplified version of this scheme is Pulse Per Second (PPS) [22], which is easily provided by GPS. The interval can be reduced by using advanced GPS or an OBC with multiple pulses per second, thereby reducing the time required for units to synchronize with the central system.

## 2.2 Broad Level Functionalities

First level functional analysis is done and presented here. This project focusses on solving an engineering problem and efforts have been made in the past to solve similar problems. This will benefit the design in utilizing experience of the experts already working in the field. One of the critical problem faced during functional analysis is to isolate firmware functionality with a specific design approach. Functions define the purpose of firmware which can be solved by number of techniques. However, a design approach refers to one way of accomplishing that particular function.

### 2.2.1 Bus Communication

Essentially the satellite bus is a user of the payload camera. The only intelligent/controllable device in a payload is firmware. Thus, in order to use the payload camera, firmware must provide a communication link [23] to the satellite bus.

### 2.2.2 Health Telemetry

Health telemetry[24] provides meaningful information for successful operation of the payload throughout its lifetime. This provides useful control data for autonomous onboard control or manual control from ground in case of single event latchup. It can also be used for monitoring the health of particular components with operational time in space to provide a meaningful space heritage of a part. This is particularly useful in

planning future missions as space heritage is a defining criteria of success for any space mission.

### 2.2.3 Sensor Communication

A payload is the user of the image sensor as the satellite is the user of thepayload. The only way for the payload to utilize image sensor data is to communicate [25] with it. Although the sensor has the capability to communicate, it should not be attached directly to bus because of its very detailed control mechanism. Most of the parameters used to control sensor are constant and depend critically on the mission rather than operation. The payload provides a useful buffer in such conditions and help in simplification of already complex Satellite concept of operation.

### 2.2.4 Sensor Data Receiver

The sensor provides image acquisition data to the payload based on the parameters set via sensor configuration. Different sensors may provide data in different formats but the primary objective of every payload is sensor data acquisition [26].

### 2.2.5 Binning

Multispectral cameras have many narrow spectral bands due to their unique application. This reduces the amount of light passing through multispectral filters to the pixels. This decreases the SNR and to compensate for this neighboring pixels need to be merged. This process is called binning and is applied in virtually every satellite. One such satellite example is given here for reference[5].

### 2.2.6 Pixel Sequencer

Data produced by image sensors is very large. Recent payloads have more demanding needs, thus high resolution is required not only in the spatial domain but also in the radiometric and spectral domains. To sustain such high standards, a very high data rate is required. To cope with that, the easiest solution is to increase number of data channels [10]. When one row of data is divided into the number of channels, pixels are no more aligned. All pixels in a row are divided evenly into the number of channels and later

aligned in the camera. To get meaningful image out of this data, received data is buffered temporarily to form a continuous image.

### 2.2.7 Image Data Transmitter

The acquired data along with preprocessing will be sent to the satellite bus[27]. This function provides the required outcome of a payload. Every other function is there to facilitate completion of this function as per payload's user i.e. satellite bus.

## 2.3 Broad Objectives and constraints

The algorithm overview and first level functional analysis discussed in 2.1 and 2.2 respectively contribute towards formation of first set of broad objectives and constraints. These objectives are not meant to be perfect but this will provide a starting point for an iterative Space Mission Engineering [28] process. These broad objectives are provided as following:

- The FW shall be able to communicate with bus.
- The FW shall be able to provide Telemetry and handle telecommand requests.
- The FW shall be able to communicate and configure sensor.
- The FW shall be able to start imaging within TBD time.
- The FW data output shall have synchronization header, auxiliary data and image data.
- The FW shall acquire image lines at 10kHz frame rate.
- The FW shall be able to receive sensor data at 10kHz.
- The FW shall be able to train itself for the correct reception of idle data before start of sensor data reception.
- The FW shall be able to receive sensor data at 80Mbps.
- The FW shall be able to run on a system clock of 32MHz.
- The FW shall be able to manipulate sensor data in at least three clock domains.
- The FW shall be able to generate all clocks required by sensor for its operation.
- The FW shall be able to generate all signals required by sensor for its operation.
- The FW shall be able to bin 2x2Multispectral Band pixels.
- The FW shall be able to power the image sensoron/off.
- The FW shall be able to synchronize with FW running on other SICs.

## 2.4 Importance and Methodology of Simulation

Design implementation into FPGAs is a time consuming activity due to availability of actual hardware late in development process and long implementation time of firmware in FPGAs. FPGAs operate at very high speed which makes the PCB design and layout very critical. The design process of such PCBs takes time and by the time PCB hardware is reached and stuffing is done, a lot of project time has passed. Secondly, the long implementation time of firmware, particularly for complex designs, in FPGAs doesn't allow trivial developmental testing in hardware.

FPGAs are expensive devices and it is not worth the cost to incur such an expense unless a particular FPGA is finalized. Although FPGA resources are analyzed at the start of the design and a corresponding FPGA is selected, it is often the case that the resources required at the end of the project are more than estimated. Optimistic resource utilization can be a culprit for this but more often the designer faces another problem called design creep, i.e. design requirements can change in the middle of project requiring a much more powerful FPGA.

As per these two ground realities, faster and cheaper firmware verification is needed prior to implementation in an FPGA. As hardware is not available in the early phases of theproject, this verification is done in software by simulating actual hardware. Apart from design verification at the end of thedesign, simulation is an important step that needs to be performed throughout the design and continuously provides feedback to improve the design. For firmware simulation to be performed, three things are required; best approximation of hardware, input or stimulus and output monitoring.

Firmware design software often provides a simulator to simulate FPGA hardware. Simulation is specific to each particular FPGA selected as design software is often provided by FPGA manufacturers. It has the knowledge of the device architecture and can simulate signals as close to the exact details of the actual hardware as possible.

Stimulus is provided by careful selection of test cases, thus simulating every input possible during the lifetime of the payload. It is important to note here that stimulus is not just limited to the value of an input but timing and often sequence of theinput stream. Incorporating all these aspects in an elaborate set of inputs often result in very complex branching resulting large number of test cases possible. It is not always

feasible to test every stimuli but recurring and critical stimulus should be incorporated into test case identification. Test cases are designed in VHDL, with special syntax for timing which is not feasible for normal VHDL design.

Similarly apart from the signal values for output, timing and sequence of signals is important for design to work. Sequencing of data is important for communication data transmission modules which require outputs in a certain packet format. Thus, the output of every signal needs to be monitored as per its requirements.

## 2.5 Ultrafast Design Methodology

Increasing the computationalpower of FPGAs makesthem very useful and ideal for a number of applications. As their applications grow, so does their complexity to better cope with growing needs. This makes firmware design for FPGAs very challenging. To keep them working seamlessly, special attention is required for every minute detail. This special attention not only makes the design powerful but also reliable, simple and fast. Xilinx provides a detailed set of questions to ask during design process and guides the design towards efficiency. These are called Ultrafast Design Methodology [29]. This methodology will be adopted for this work.

Achieving the most out of FPGA devices is only possible by understanding the fundamentals of the particular FPGA being used. Many vendors have provided detailed documentation for every family of FPGAs. Xilinx provides number of documents which help various aspects of design. Kintex-7 series FPGA documentation is provided inTable 2-1.

**Table 2-1** Kintex-7 Xilinx FPGA Documentation Example clearly defines the extent of guidelines provided by Xilinx.

| Serial No. | Name | Description |
|---|---|---|
| 1 | FPGA Series Datasheet | Datasheet outlining main features of a series. |
| 2 | FPGA Subseries Datasheet | Datasheet outlining main features of a subseries. |
| 3 | FPGA Configuration User Guide | User Guide for FPGA configuration procedures |
| 4 | FPGA SelectIO Resources | User Guide for IO tile resources and their limitations |
| 5 | FPGA Clocking Resources UG | User Guide for Clocking resources and their limitations |
| 6 | FPGA Memory Resources UG | User Guide for Memory resources and their limitations |
| 7 | FPGA Configurable Logic Block UG | Provides CLB architecture and guides HDL and pin assignment for efficient device utilization. |
| 8 | FPGA Packaging and Pinout Product Specification | Provides guidance for selection of best device and package within family for your application. |
| 9 | FPGA Library Guide | Provides information and syntax of FPGA built in primitives and how to use them |
| 10 | Application Notes | Application notes specify the correct usage and highlight benefits and shortcomings of a particular feature |
| 11 | IP Documentation | Provides detailed information about IP parameters and its configurability |

# 3 FIRMWARE REQUIREMENTS ANALYSIS

Before proceeding to the actual firmware development process, preliminary steps need to be followed to efficiently define the firmware design. Firmware development is then a simple translation from design to VHDL firmware. This means that every aspect of the design should be properly defined and quantified. This eliminates any vague expectations. The way to achieve that is to perform functional analysis, develop a functional flow block diagram and requirement analysis.

This chapter highlights core issues required for SIC operation. Quantified requirements are then formalized and fed to the next chapter for an efficient design process. Multiple iterations of the SME[28] process has been done from the starting point of objectives and constraints highlighted in section2.3. With every iteration, new functionalities are added and existing functions are focused. According to Akin's Laws[30], number of iterations required is always one greater than current iteration. But due to limitation of time and resources, these iterations are stopped when certain level of maturity is achieved.

Functional analysis is used to identify functions and no thought is given for solution apart from practical feasibility. This defines exactly what needed to be done. The main functions of the Sensor Interface Card (SIC) is to provide Sensor Handling, Telemetry and Telecommand (TMTC) and Data Manipulation. The effective approach for design development incorporating these functions is to break them down to sub-functions. Logical decomposition[31] is a process used to define a concise and quantified functional breakdown. In addition to logical decomposition used for design, this process is also used for gaining a better understanding of requirements and their inter-relations. Logical decomposition of functions is given in Figure 3-1.The reasoning behind every function identification will be described in this section.

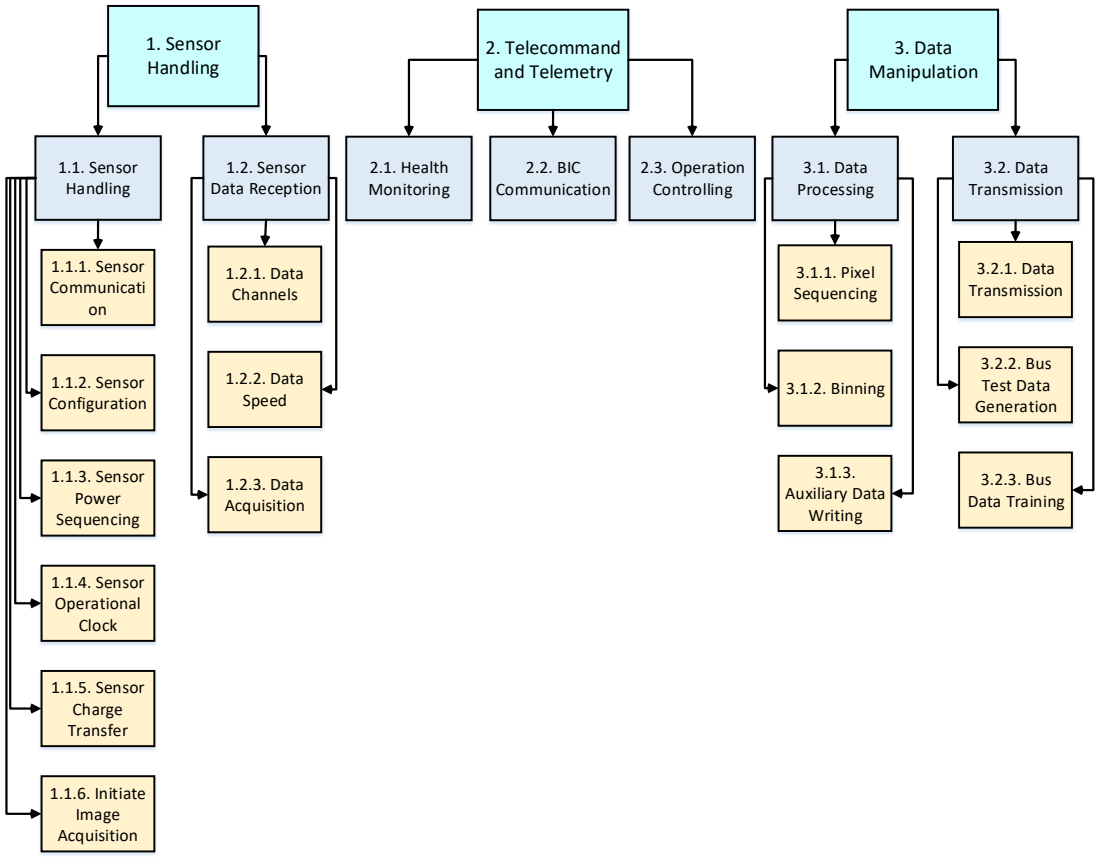



**Figure 3-1**Logical Decomposition of SIC Functions.

## 3.1 Design Drivers for Sensor Data Handling

The image sensor is the driving part of firmware design as firmware itself can't generate any useful data. It can only receive data and command the sensor to provide meaningful data. As these functions are specific to the sensor, firmware should be tailored to the specifications of sensor. Functional breakdown of these major functions is discussed in this section.

### 3.1.1 Command Sensor

This ensures that the firmware has complete control of the sensors. This function is broken down into sub-functions below for clear firmware specifications.

#### 3.1.1.1 Sensor Communication

To achieve the major function, the firmware shall be able to communicate with the sensor. Sensor provides an SPI interface link upon which commands can be relayed.

Firmware shall be operated in a Master configuration to be able to initiate communication with the sensor.

#### 3.1.1.2 Imaging and data link configuration

Every image sensor is used to capture light. This functionality can be performed for a number of applications. The way a sensor needs to capture light for one application may differ drastically from another. Each sensor provides number of configurable registers which can be accessed via SPI to provide imaging information tailored to this work. This includes exposure time, Programmable Gain Amplifier (PGA) gain, etc. It also provides a configuration register for data link including pixel bit depth, number of channels to be used, etc.

#### 3.1.1.3 Sensor Power Sequencing

The image sensor runs on number of power supplies. These power supplies need to follow a certain procedure to ensure successful power-up. If not followed correctly, the sensor is at risk of damage. This power sequencing is provided by firmware along with timing requirements between these supplies.

#### 3.1.1.4 Sensor Operational Clock

Every electronic device needs to operate at certain clock frequency. Complete operation of an imaging sensor is dependent on this. It also derives the data output rate of thesensor. Firmware needs to provide this clock and should ensure that this clock should not vary.

#### 3.1.1.5 Sensor Charge Transfer Clock

Charge transfer for multiple TDI stages is also dependent on the specific application. As seen in section2.1.3, the CCD clock needs to be dependent on altitude which may vary. Firmware shall be able to change this clock based on request.

#### 3.1.1.6 Initiate Image Acquisition

The firmware shall be able to instruct the image sensor when to image. The satellite velocity is very high and depending on resolution, the firmware shall be able to

command sensor for image acquisition at regular intervals. This rate will be based on request from BIC and not deduced in real time by the SIC Firmware.

### 3.1.2 Sensor Data Reception

A number of things need to be considered for a sensor to receive image sensor data. This main function is broken down in this section.

#### 3.1.2.1 Data Channels

A sensor can provide a variable number of channels as selectable by function described in section3.1.1.2. Sensor has sixteen selectable data channels in increments of two. Firmware shall be able to receive data in each configuration.

#### 3.1.2.2 Data Speed

Sensors can provide data at different data rates as seen in section3.1.1.4. Firmware shall be able to handle data at a maximum rate of 100Mbps.

#### 3.1.2.3 Data Acquisition

The sensor provides data synchronous to the sensor data clock. This is used communicate to communicate with the sensor when to take sample of data channels. Firmware shall be able to acquire data as per sensor data clock.

## 3.2 TMTC

SIC firmware itself is not an intelligent entity. It will operate as per commands provided by the BIC. This main function is decomposed into health telemetry, BIC communication and operational control. The first two functions are specific to BIC, thus SIC firmware will be designed as per BIC specifications. Third function is internal to the SIC.

#### 3.2.1.1 Health Telemetry

SIC firmware will acquire health telemetry of the SIC and sensor. It will update this information at the rate set by BIC. The BIC can access this information whenever it is needed.

#### 3.2.1.2 BIC Communication

SIC firmware should provide a communication interface which allows the BIC to provide telecommands to the SIC. Commands execution status will also be communicated to the BIC via this interface. Health telemetry will be provided to BIC to make intelligent decisions about SIC firmware operation.

#### 3.2.1.3 Operation Control

SIC firmware will operate in predefined states and the number of states may vary from one operation to another. For a fixed command, operations will always be performed in a specific order. However, this function will ensure that next step should only be performed if previous steps are completed. In case of error, it shall not try to correct the error but only update its status.

## 3.3 Data Manipulation

This function provides data manipulation which requires the FPGA to temporarily store image line data. This makes this function most resource hungry. It includes Data processing within FPGA and Data transmission to spacecraft.

#### 3.3.1.1 Data Processing

Data provided by the sensor needs to be processed before it can generate some meaningful information. This processing function is further decomposed into three functions.

#### 3.3.1.2 Pixel Sequencing

The sensor provides image data in a format that is most suitable to the sensor architecture. Any processing to comply with specific image data format is not done. This results in sensor data generation which is often not aligned by pixel number. This function sorts data coming from the sensor by pixel number. This is required for any processing required on data. It is also useful for the end user to only look at an image and not worry about sensor formatting.

#### 3.3.1.3 Binning

The sensor provides data for multiple rows. All rows can accept light for the same wavelength range, but some applications require filtered light. This is achieved by incorporating separate filters on different rows. This makes particular rows of the sensor into bands termed as panchromatic or multispectral. Panchromatic usually covers large range of wavelengths while a multispectral filter only allow small wavelength range filtered light to pass through to the sensor. This lowers the multispectral pixel energy level. To increase this, multiple pixels are added at the cost of decreased spatial resolution.

#### 3.3.1.4 Auxiliary Data Writing

A user often need to know exactly where and when an image was taken. It is also helpful for user to know the look angle of telescope at the time of imaging. This information is added in the image data by this function. It also appends useful information to distinguish one line/band information from the other.

### 3.3.2 Data Transmission

Image data after processing is sent to spacecraft. To achieve that, this function is broken down into following functions.

#### 3.3.2.1 Data Transmission

This function enables the data transmission to the spacecraft. This transmission is independent of data provided input to this function.

#### 3.3.2.2 Bus Test Data Generation

This function generates the test data to be transmitted to the bus using the data transmission function. It is useful for the receiver to verify its link with transmitter.

#### 3.3.2.3 Bus Data Training

Training data is sent usually before an actual data transmission to enable the data receiver to successfully receive data. Training data is a repetitive fixed pattern already

communicated to the receiver. The receiver can provide time delay and bit shift to recover training pattern. This enables reliable data reception without any loss.

## 3.4 Functional Flow Diagram

Functional flow defines the exchange of information between different functions necessary to operate a system. It defines the interconnections necessary between functions to perform bigger operations. Functional flow diagrams define the operations of the SIC firmware. Three basic functional flows are discussed here that are found in any payload. Functional analysis is cross checked as per these operations. Every operation is checked to ensure that the functional analysis provides adequate functions to perform these operations successfully. Moreover, it is also observed that there should not be any functions of the SIC firmware that are not used in defining these operations. If any such function is highlighted in functional analysis, it should be revisited.

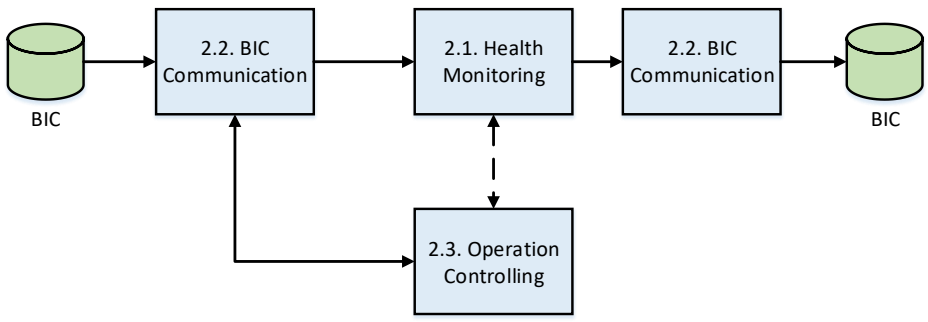



**Figure 3-2**SIC Firmware Functional Flow: Health Monitoring.

Health monitoring operationsare depicted in Figure 3-2. It involves three functions that will be running whenever SIC is powered on. This operation is initiated by the BIC and the result of this operation will also be fed into the BIC.

Another useful operation is to test the link between the spacecraft and SIC which is depicted in Figure 3-3. This is initiated by BIC when it is commanded by spacecraft to generate test data. Similarly, spacecraft must also generate same test data. One to one correspondence of this data will verify SIC to Spacecraft link.

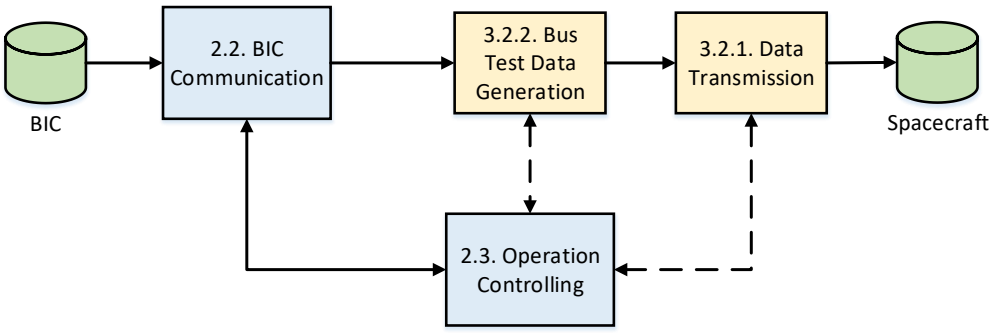



**Figure 3-3**Bus Test Data Generation Operation.

The main operation of the Sensor Interface Card is imaging. SIC firmware is commanded by the Bus Interface Card to start imaging. It triggers sensor to provide data, processes it and then transmit it to spacecraft. It involves almost all functions and this operation is depicted in Figure 3-4.

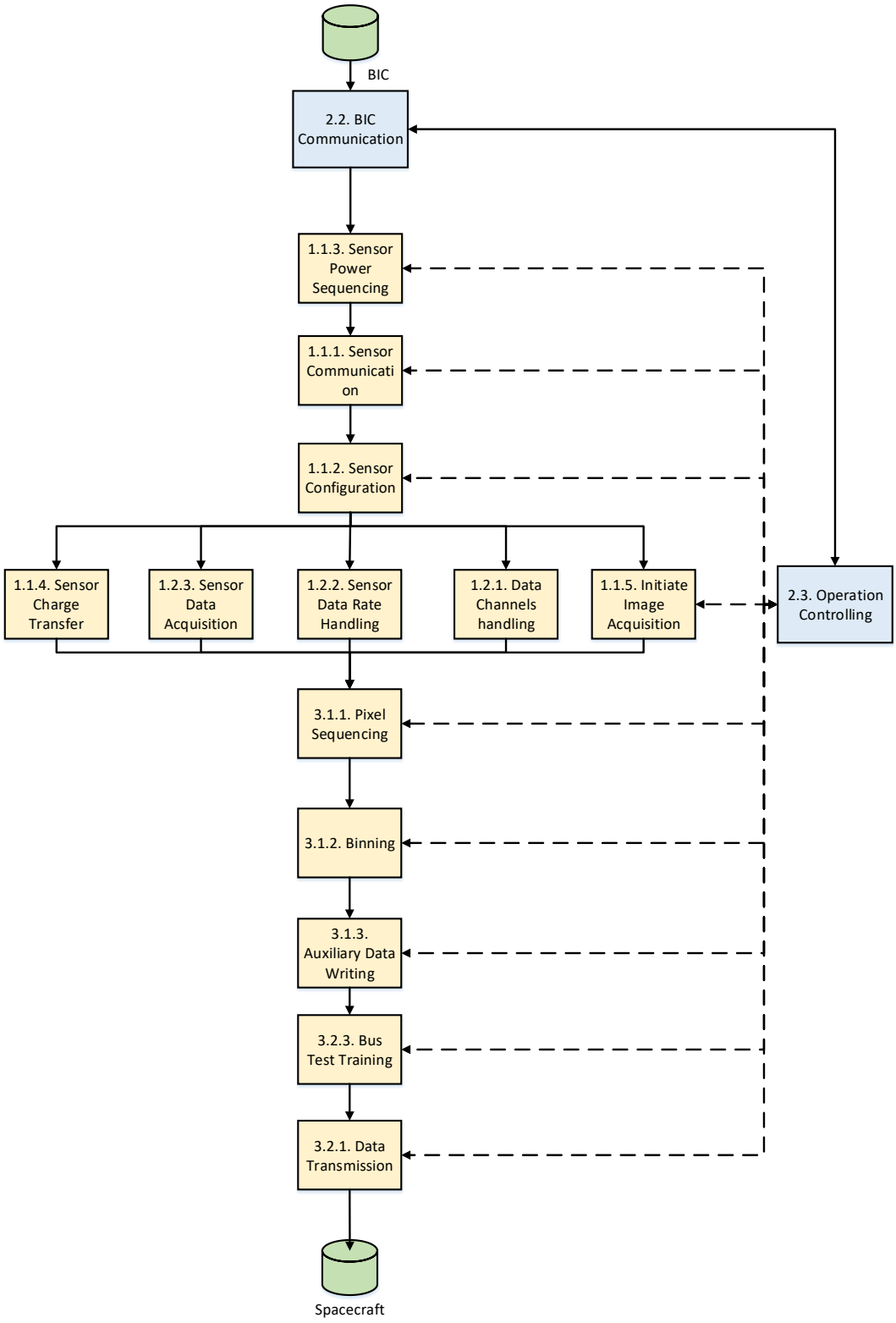



**Figure 3-4** Imaging Operation.

## 3.5 Requirements

Requirements are quantitative definitions of functions. Sections 3.1, 3.2 and 3.3 define the functions of the SIC firmware. Requirements are necessary to bind the system specifying the exact level of functionality to be done. Nomenclature used for the requirements definition in this work is given below:

FW-XXX-YYY

where YYY represents requirement number and XXX specifies top level function from which the requirement is derived. As seen before, there are three top level functions and their designators are given below:

- Sensor Handling: SEN
- TMTC: TMT
- Data Manipulation: DAM

Iterative requirements based on functional analysis are provided inTable 3-1.

**Table 3-1** Image Sensor Driven Firmware Requirements.

| ID | Title | Description |
|---|---|---|
| FW-SEN-001 | Sensor Data Channels | FW shall be able to receive data on sixteen channels. |
| FW-SEN-002 | Register Configuration | FW shall be able to configure sensor registers on SPI protocol at 500kHz. |
| FW-SEN-003 | Sensor Data channel speed | FW shall be able to receive maximum data rate of 80Mbps on every sensor data channel in DDR mode. |
| FW-SEN-004 | Sensor Data Clock | FW shall be able to receive sensor data on clock provided by sensor. |
| FW-SEN-005 | Master Clock | FW shall be able to provide fixed master clock to sensor with frequency $1/10^{th}$ of required data clock frequency. |
| FW-SEN-006 | CCD Clock | FW shall be able to provide variable CCD clock to sensor with 1kHz range centered at 10kHz. |
| FW-SEN-007 | Sensor Power Sequencing | FW shall be able to provide sensor power on and off sequencing required by sensor. |
| FW- | Image | FW shall be able to trigger image acquisition of sensor |

| ID | Title | Description |
|---|---|---|
| SEN-008 | Acquisition | by providing pulse at regular intervals as per set line rate. |
| FW-SEN-009 | Band data | The SIC firmware shall be able to handle one Panchromatic and four multispectral band data. |
| FW-TMT-001 | BIC Communication Interface | The FW shall have a SPI communication interface with the BIC at 500kHz. |
| FW-TMT-002 | Telemetry update rate | The SIC firmware shall be able to update Telemetry on SPI registers every second for BIC to fetch. |
| FW-DAM-001 | Pixel sequencing | The SIC firmware shall sort the sequence of every band pixel as per pixel number. |
| FW-DAM-002 | Binning | The SIC firmware shall be able to perform 2x2 binning of multispectral band data. |
| FW-DAM-003 | Auxiliary data writing | The SIC firmware shall be able to add at least following auxiliary data in every line.<br>Time<br>AOCS data |
| FW-DAM-004 | Spacecraft training data | The SIC firmware shall be able to generate training data for spacecraft data interface to train upon. |
| FW-DAM-005 | Spacecraft test data | The SIC firmware shall be able to generate spacecraft test data as incremental 10-bit wrap-around counter. |
| FW-DAM-006 | Data Transmission | The SIC firmware shall be able to transmit training data, test data and image data. |

# 4 BASELINE FIRMWARE DESIGN

This chapter focuses on the firmware design overview. It starts with the SIC input and output data budgets. These will serve as inputs to the baseline architectural design. In addition, the general architecture is also derived from the literature review in Chapter 2 and the functional analysis and requirement analysis as seen in Chapter 3. In literature review, multiple aspects of payload camera firmware design are highlighted. Multiple schemes to achieve those algorithms and functionalities will be provided in this chapter which will serve as alternate approaches for the SME process.

Furthermore, advantages and disadvantages of every methodology are noted. Based on that, baseline selection of every aspect is provided which combine to form SIC firmware baseline architecture, which drives the FPGA selection process. At the end of the chapter, the FPGA selection for this work will be finalized.

## 4.1 Data Budget

Data handling is an important consideration of this work. Data budget analysis puts a concrete quantitative analysis for the data handling targets required by this firmware. SIC firmware provides data interfaces with the image sensor and spacecraft.

### 4.1.1 SIC Data Budget

The image sensor is the data source to be handled by this firmware. The image sensor contains four rows which will be populated by different filters to provide Panchromatic (PAN) and Multispectral (MS) imaging in the red, blue and green bands. Each pixel has a pixel depth of 10 bits. A frame consists of four bands of data which needs to be received within the flight time. Flight time of a pixel is defined as the time required for a satellite to pass over a GSD. This will set minimum data rate requirements for theSIC firmware.

Based on the master clock of a sensor, data will be provided to firmware at a slightly higher rate. This is done to serve two purposes. Firstly, this ensures that in case of satellite altitude variations as seen in section2.1.3, the firmware will be able to handle data at slightly higher data rate. Secondly, some idle data needs to be transmitted in between and before actual data transmission for sensor training done by receiver as seen

in section2.1.1. This will increase the total data to be transmitted by the sensor which is to be handled by firmware. Input data rate budgets and sensor data rate acceptable by every channel arelisted in Table 4-1.

**Table 4-1** Sensor Data Rate Budget: SIC Firmware Input Data Budget.

| S.No. | Parameter | Value | Unit | Description |
|---|---|---|---|---|
| 1 | Bands | 4 | Bands | Total Bands of Sensor |
| 2 | Pan | 1 | Band | Panchromatic band to accept complete light |
| 3 | MS | 3 | Bands | Multispectral bands to accept filtered light |
| 4 | Sensor Width | 2,048 | Pixels | Total number of pixels for one sensor |
| 5 | Total pixels | 8,192 | Pixels | Total amount of pixels for each frame (all bands) |
| 6 | Bits per pixel | 10 | Bits | |
| 7 | Total Bits per frame | 81,920 | Bits | Total amount of bits for each frame |
| 8 | Flight timer per pixel | 100.00 | us | Time to travel one GSD |
| 9 | Sensor Bandwidth Required | 819.20 | Mbps | Minimum bandwidth to accept sensor data on all 16 channels |
| 10 | Minimum Data rate required by every channel | 51.20 | Mbps | Considering that sensor has 16 data channels |
| 11 | Sensor Master Clock | 4.00 | MHz | Sensor master clock generated by sensor |
| 12 | Clock multiplier | 10 | | Multiplication factor of sensor |
| 13 | LVDS clock | 40 | MHz | 10*Master Clock |
| 14 | Data Rate (DDR) | 80 | Mbps | 2 bits per LVDS clock cycle for rising and falling edge of clock |
| 15 | Total Output Rate | 1,280 | Mbps | Must be higher than sensor Bandwidth required |

### 4.1.2 Spacecraft Data Budget

The data received by the sensor needs to be processed as highlighted in the functional analysis and requirement analysis. This data will be then provided to thespacecraft. In processing, data may change. Thus it will not be same as data provided by sensor. The data rate required for the SIC firmware output is given in Table 4-2.

**Table 4-2** Image Data Transmission: SIC Firmware Output Data Rate Budget.

| S.No. | Parameter | Value | Unit | Description |
|---|---|---|---|---|
| 1 | Binning | 2 | | 2 by 2 pixel binning |
| 2 | Pan lines | 2 | Lines | Pan does not get binned |
| 3 | MS lines | 3 | Lines | Multispectral data provided after binning |
| 4 | Bits per pixel | 16 | Bits | Pixel depth of every pixel after binning |
| 5 | Auxiliary Bytes per line | 80 | Bytes | Auxiliary information for every band |
| 6 | Total bits | 117,888 | Bits | |
| 7 | Time to output MS lines | 200 | us | Flight Time * Binning |
| 8 | SIC Bandwidth Required | 589.44 | Mbps | Total bandwidth to output all the data in time |

## 4.2 SIC Baseline Firmware Architecture

We discuss how the SIC firmware architecture and how individual components interact with each other is provided. The goal of architecture design is to achieve a low cost, high reliability system with minimum development time. This architecture will be the backbone for further development. Effort is put to generate a mix of modularity, testability, compatibility and future upgradeability in architecture. This will result in maximizing firmware reusability, easier development and testing of complete project. The high level firmware architecture is given in Table 4-1.

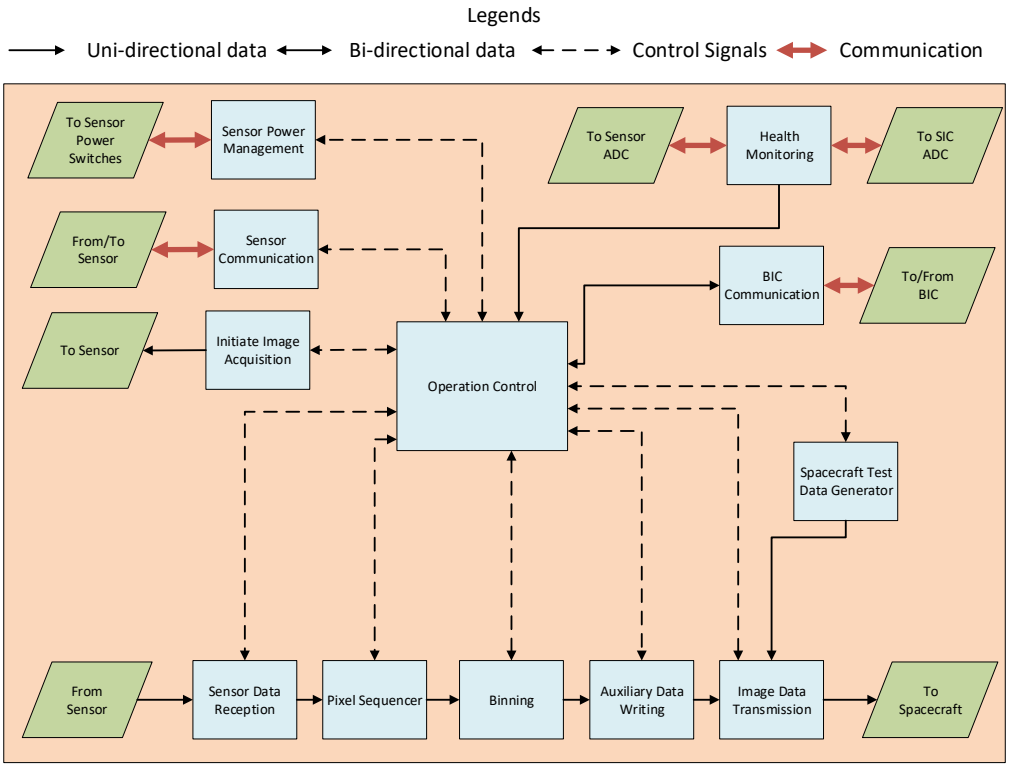



**Figure 4-1** Firmware Architecture.

## 4.2.1 Data Handling Components

Firmware architecture consists of twelve components. Among these components, five components which handle and process sensor data are very resource intensive. These components will be discussed here in detail. Every component helps in resolving a number of challenges which can be solved in several ways. The baseline procedure of solving a problem is finalized keeping in view its advantages and drawbacks.

### 4.2.1.1 Sensor Data Reception

This component is responsible for the reception of data from the sensor, which provides one LVDS clock and up to sixteen LVDS data channels. The firmware configures the sensor for 16 LVDS channels to decrease data rate required. FPGAs can receive clock and data with multiple buffers which are given inTable 4-3.

**Table 4-3** Sensor Data Reception: LVDS Input Methodologies for FPGA.

| S. No | Buffer Parameters → Buffer types ↓ | Output Type | Termination Enable Control | Input Enable Control |
|---|---|---|---|---|
| 1 | IBUFDS | Single Ended | No, Fixed Termination | No, Always Enable |
| 2 | IBUFDS_DIFF_OUT | Differential | No, Fixed Termination | No, Always Enable |
| 3 | IBUFDS_DIFF_OUT _IBUFDISABLE | Differential | No, Fixed Termination | Yes |
| 4 | IBUFDS_DIFF_OUT _INTERMDISABLE | Differential | Yes | Yes |
| 5 | BUFDS_IBUFDISABLE | Single Ended | No, Fixed Termination | Yes |
| 6 | IBUFDS_INTERMDISABLE | Single Ended | Yes | Yes |

As can be seen from Table 4-3a lot of options are provided by the FPGA to receive inputs. However, the sensor provides data with fixed termination so there is no need for termination control at input. Input control is also not necessary because the SIC only powers up the sensor when commanded. The time for which SIC is powered on without taking data input from the sensor is very short. Differential signaling is more immune to common-mode noise. Within the FPGA, chances of noise to be added are much lower. Thus, single ended output type is better in simplicity and connections required. It can be seen that every buffer can be used, but for simplicity IBUFDS will be used in baseline design.

After clock and data reception, it can be seen from Section 2.1.1that delay should be added in their path. This delay can be added from two methods highlighted inTable 4-4. A built-in delay macro is already part of multiple FPGAs so it doesn't require any resources while implementing a delay scheme in a FPGA takes a lot of resources. Implementation of delay scheme in FPGA requires a lot of resources to implement a very small delay. However, the delay value provided by built-in resources has no effect on resources. A built-in delay is very easy to use compared to FPGA resources. Detailed testing of built-in delay macro is done by the FPGA manufacturer, thus it provides high

reliability compared to a user defined logic delay. Thus option 1 in Table 4-4 is better to incorporate in baseline design.

**Table 4-4** Sensor Data Reception: Delay Mechanism.

| S.No. | Delay Parameters → Delay types ↓ | Resources | Efficiency | Ease of use | Reliability |
|---|---|---|---|---|---|
| 1 | Built-in Delay Macro | Very low | Good | Very Easy | High |
| 2 | FPGA Resources | High | Low | Difficult | Low |

The next important aspect discussed in section2.1.1is serial to parallel conversion. There are two methods by which this conversion can be done, which are provided in Table 4-5. IDDR simply converts data coming on rising and falling edges to rising edge of data clock. Sensor pixel transmitted is 10 bits wide. Thus serial to parallel conversion is done using FPGA resources. This makes it complex, difficult to use and requiring high resources. Use of ISERDES on the other hand doesn't require any resources for serial to parallel conversion as it is built into the macro design along with bit shift control. Thus it's easier to use, give better control and has less complexity. Thus the ISERDES option is more suitable.

**Table 4-5** Sensor Data Reception: Serial to Parallel Conversion.

| S.No. | Conversion Parameters → Conversion types ↓ | Resources | Ease of use | Control | Complexity | Bit shift Control |
|---|---|---|---|---|---|---|
| 1 | IDDR | High | Medium | Medium | Medium | No |
| 2 | ISERDES | Low | Easy | good | Low | Yes |

Clock division is required to handle parallel side of ISERDES and FIFO writing. Two methods are possible for this clock division. It can either be done by using clock divider buffers or it can be done by using MMCM. MMCM also provides fractional division and multipliers which makes MMCM implementation difficult. There are also limited number of MMCMs present in an FPGA. Thus it is much easier and less resource

intensive to use clock divider buffers. This scheme will be used for the baseline design architecture.

As per all these considerations, the architecture of Sensor Data Reception is given inFigure 4-2. The FIFO width required is 160 bits to write 16 pixels. Each pixel has a width of 10 bits. With same read width, FIFO design of this component requires 2.5 BRAMs.

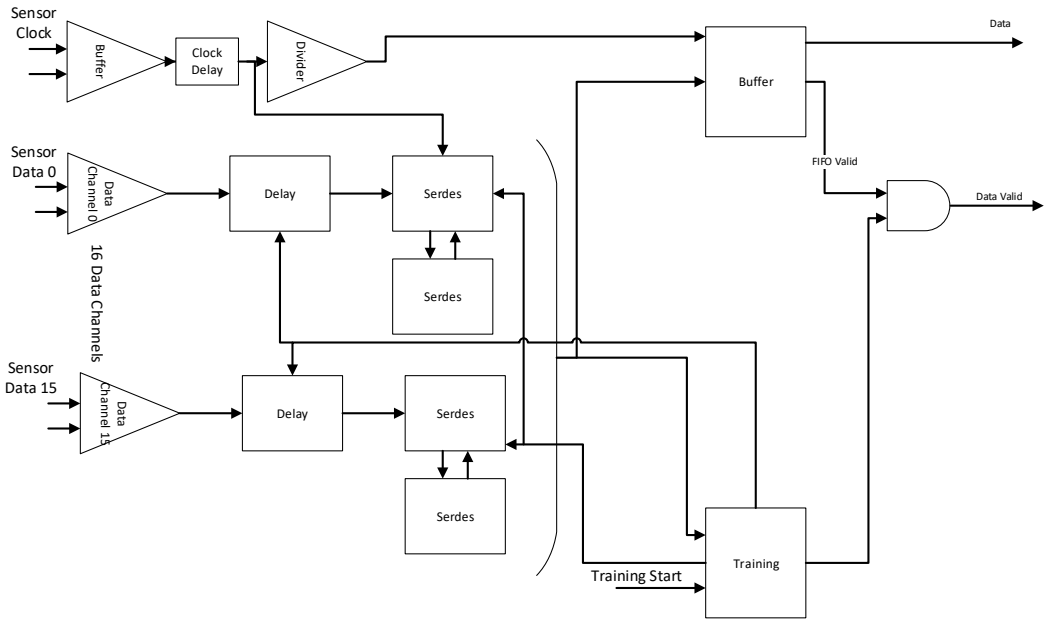



**Figure 4-2**Sensor Data Reception Architectural Design.

### 4.2.1.2 Pixel Sequencing

The pixel output format of a sensor is particularly difficult to streamline due to its unusual transmission of odd pixels followed by even pixels. Sensor pixels are further divided into number of channels. Every channel transmits 128 pixels, with sixteen channels completing 2048 pixels. Thus to sort this data with pixel number, multiple factors are involved i.e. complexity, system clock speed and pixel sorting level.

FPGA devices are particularly good in meeting timing requirements of any given system. Complexity in a design results in more area coverage of an FPGA which results actual logic in fabric to be far apart. This creates difficulty in meeting timing constraints which are otherwise possible to meet in a simpler design.

Operational clock speed is also directly related to timing constraints. Another important consideration for increased clock speed is EMC/EMI. Satellites are prone to EMC/EMI

as it houses multiple complex systems in a confined space. To ensure smooth working of a satellite, it is important that minimal noise is generated during operation. Increasing data output rate of pixel sequencing component will result in increasing operating speed of other components. This may drive the system clock of FPGA higher. As the system clock is provided to complete FPGA logic, a higher system clock will result in more electronic interference for other satellite units.

Pixel sequencing is crucial in all the modern satellites for two reasons, namely binning and compression. As seen in Chapter3, binning is part of this firmware and without some form of sequencing, binning is not possible. Although compression is not in scope of this work, it is often part of the complete data handling chain of a satellite. For compression, the perfect order of pixels is important as it works closely on neighboring pixel values.

Trade-offs can be done to achieve best desired outcome meeting the system needs. Based on these factors, three possible solutions are discussed and then one of them will be chosen. The only thing common in these trade-offs is the presence of temporary storage which is provided in FPGAs in the form of Block Random Access Memories (BRAMs). The number of BRAMs required varies with different approaches.

*FIFOs*

A FIFO is proposed to be the storage element in this scheme which simplifies the complexity angle of trade-off as these are easier to handle because of the simplicity of FIFO operation. The number of FIFOs required is twice the number of channels involved. Each FIFO in an individual channel will store either odd or even pixels. These pixels will be read once line data is complete. FIFOs used can have 10 bit write width and 80 bit read width.

Thus by using even and odd FIFOs and reading two FIFOs at a time, it is ensured that the input rate and output rate of this component are same. This will result in perfect pixel sequencing without increasing the operating clock. The downside of this approach is large storage requirement. For 16 data channels, this scheme will require 32 FIFOs each taking 1.5BRAMs. Thus 48 BRAMs are required in this solution.

*Dual Port Memory*

This approach uses random access memory to perfectly sequence pixels. Received pixels will be written into memory as per their pixel number. After that, the data will be read continuously. Although odd and even pixels of the line can be written in the same memory, a second memory is required to ensure there is no conflict in reading and writing of the next band data. The size of memories required is not just linked to data stored but also on the read out clock and effectively read width of memory used. The data read clock and its implications on BRAMs is given in Table 4-6.

**Table 4-6** Dual Port Memory Concept for reordering. Due to peculiar nature of sensor, such higher estimates of BRAMs are required and yet only option 1 is feasible in this case.

| Memory Approaches | Read Width | BRAMs per channel per one band data | Total BRAMs | Output Clock/Input Clock |
|---|---|---|---|---|
| 1 | 80 bit (8 Pixels) | 2.5 | 80 | 2 |
| 2 | 160 bit (16 Pixels) | 5 | 160 | 1 |

The best approach from an EMC/EMI point of view would be the second. However, the large number of BRAMs required would drive the design to more sophisticated FPGA requirements. This will drive cost, thus only approach 1 is practical here, with still a very large number of BRAMs.

*Partial Sequencing*

This approach lowers the amount of FIFOs used in first approach with the help of temporary registers in the FPGA. It also assumes that pixels will be sorted just enough that binning of 2x2 is possible. Thus it requires sequencing in ground for true ordered pixels. FIFOs used in this scheme will have 20 bit write width with 20 bit read width.

All odd pixels will be written in FIFOs of corresponding channels in word of two pixels each. Then with arrival of the first even pixel, the FIFO is read. The first odd pixel of the read data is combined with first even pixel received and provided for the FIFO write in the next cycle, while the other odd pixel is stored temporarily to be written with next even pixel. This will be done for every channel. Read and write width is not required to be high as every FIFO is read to provide output.

Thus sensor data will be partially sorted with pixel number. This technique will enable every FIFO to be 1.5 BRAMs large. Thus, for a total of 16 FIFOs, 24 BRAMs will be used to partially sequence complete data. This makes binning of partially sequenced data possible with minimal resources. However, data generated by using this method can't be compressed without perfect sorting w.r.t. pixel number.

*Pixel Sequencing Baseline Design*

Based on this analysis, the first approach seems most logical. This provides perfect sequencing of pixels and simplicity without any increase in operational clock. The pixel sequencing architecture is given in Figure 4-3 with 48 BRAMs required for FIFOs.

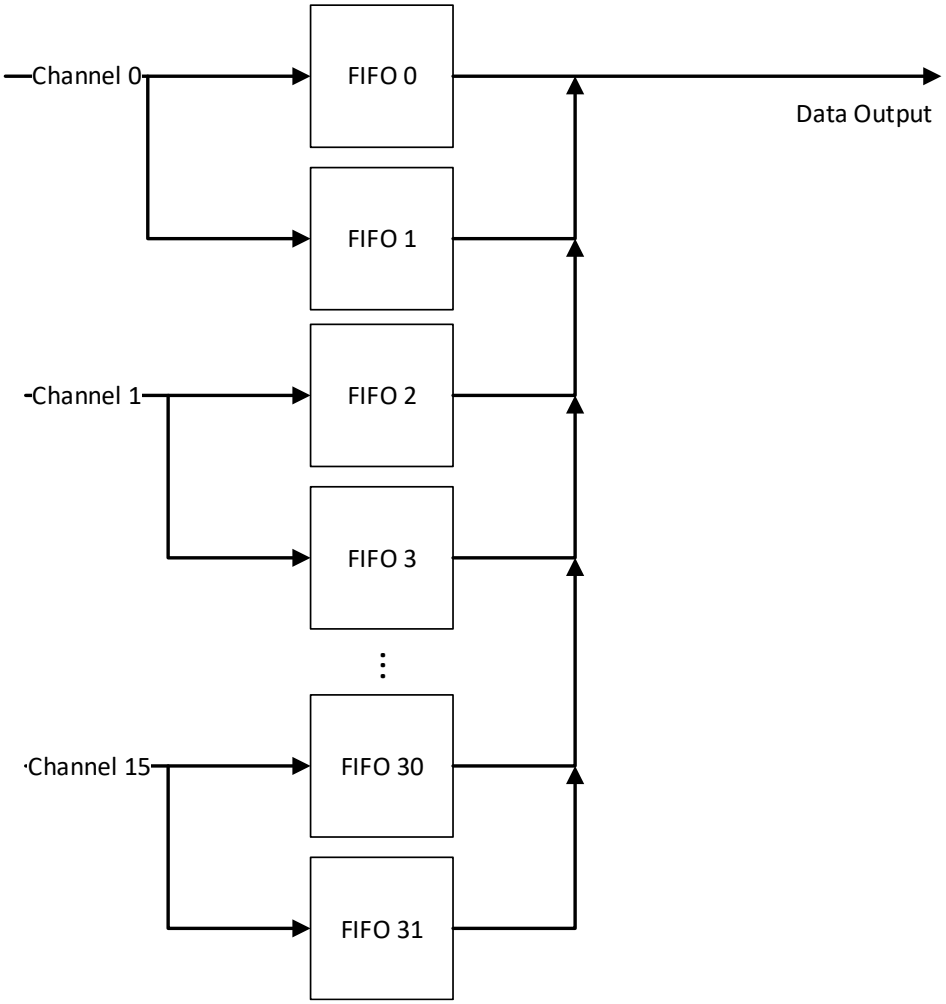



**Figure 4-3** Pixel Sequencing Architectural Design.

#### 4.2.1.3 Binning

Binning can be done to improve Signal to Noise Ratio (SNR) of an image. This can be increased by either increasing signal level or decreasing noise in the image. It is done by adding neighboring pixels. Addition works by increasing the signal level by factor of N while noise increase only by a factor of square root of N.

PAN data pixels are directly given to output with zero padding of most significant six bits. Thus making binning output of 16 bit pixel. For addition of 4 (2x2) pixels of MS band data, two pixels of the current line and two pixels of the next line are required. Thus partial addition of pixels are done for the current row and the result needs to be stored for the next addition. For addition of two pixels each, 1024 results need to be stored. Two approaches are proposed to store this temporary data before output. The first approach involves three small band specific FIFOs, while the second approach requires one FIFO to store temporary results of all bands in one FIFO.

Storage should be adequate to store the result of two 10 bit pixels. The maximum value of a 10 bit pixel can be 1023. With two pixels, the maximum value of 2047 can be achieved, which needs to be stored in an 11-bit word. Although binning requires addition of four pixels there is no need to store this in FIFO. It will be temporarily stored in logic and then provided at output when 16 pixels are complete.

Thus, for sixteen channels, 88 bit wide FIFO is used to store eight results temporarily. For one band, depth of this FIFO required is 128. This FIFO requires 1.5 BRAMs each. Three such FIFOs require 4.5BRAMs. Additionally, special logic is also required to route pixels to separate FIFOs for writing and reading depending on band.

Another approach is to use the same FIFO for every multispectral band. This approach makes use of data consistency between multiple bands and through different lines. This ensures binning with less complexity which makes this solution more favorable for this work. It also requires less BRAMs as one FIFO with a depth of 512 is enough to handle all band data. This will take 1.5 BRAMs only. Thus for one line, each multispectral data is stored in a FIFO with addition of two neighboring pixels. For a second line, two neighboring pixels and stored data in the FIFO are added together and given output. The architecture of this component is shown in Figure 4-4.

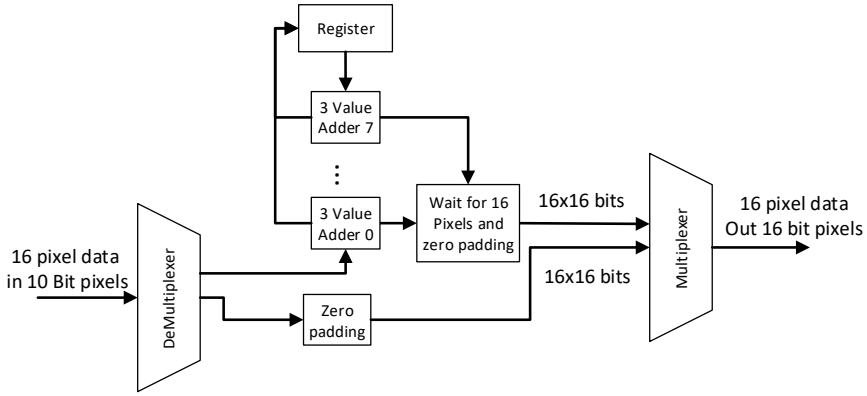



**Figure 4-4** Binning Architectural design.

### 4.2.1.4 Auxiliary Data Writing

There are two distinct auxiliary types information that need to be appended in image data based on the information source. Camera parameters are appended in image data to provide camera information useful for processing on the ground. This may include sensor configuration settings, total number of lines required, SIC status, and other information useful for image corrections and calibration. AOCS data is also embedded in image data for geometric correction. SIC parameters will be written in image data, but AOCS data can be written by the SIC or by final storage system of spacecraft. This serves as two alternate approaches for auxiliary data writing.

However, it is beneficial to add AOCS data in the SIC as this is right after image acquisition is done in the sensor. This information must be added as close to the sensor as possible providing exact position, attitude and time information at image acquisition. Adding this information in storage, which is often part of spacecraft, may decrease the accuracy of AOCS data due to associated delays. Thus both parameters of auxiliary data, SIC parameters and AOCS data, will be written by the SIC in baseline design.

Another aspect to consider for auxiliary data writing is its temporary storage before data provision on output. It can either be stored in registers or dedicated memory. The benefit of storing it in registers is fast retrieval. However, using FPGA logic to store data is not efficient as every LUT only provides four bits of storage. Thus just the auxiliary data storage will need 148 LUTs. Then another storage of 148 LUTs should also be provided which will store auxiliary data while data is read from this array. Complete auxiliary data is not updated in one clock cycle, thus auxiliary data read write conflict management is also required.

Another simpler approach is to use BRAMs as memories. This will ease complexity as data handling management is done by the IP core. The only downside is two additional BRAMs required and the slow read speed of memories. Slow read time is bearable because the update rate of auxiliary data is very slow. Read-write conflicts can also be managed by using two memories. Simplicity makes this approach much better than first one.

The architectural design of auxiliary data writing is provided in Figure 4-5. The FIFO used has a write and read width of 256 with the depth of 256. 4 BRAMs are used for this FIFO. Each memory has read and write width of 64 bits which takes 1 BRAM each. Thus in total this component requires 6 BRAMs.

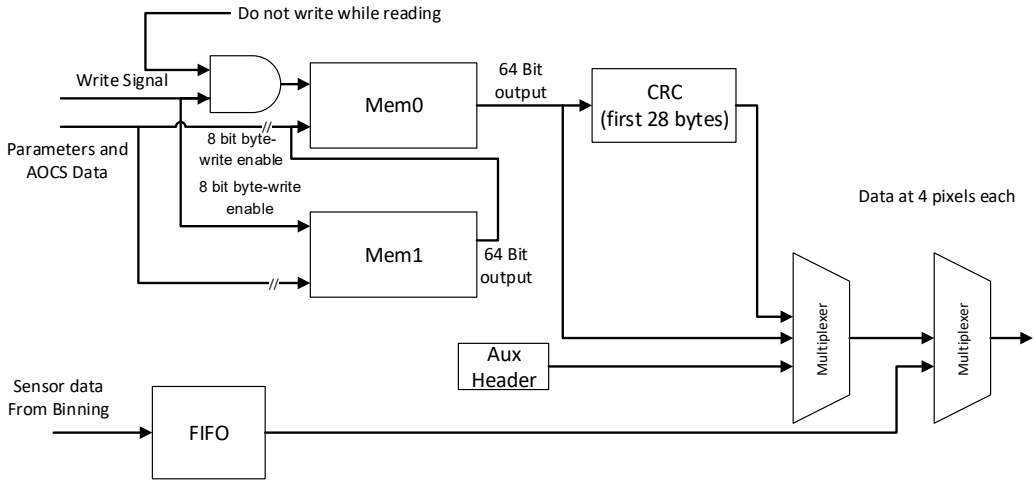



**Figure 4-5** Auxiliary Data Writing Architectural Design.

### 4.2.1.5 Image Data Transmission

The image data transmission budget is highlighted in section4.1.2. The main objective of this component is to handle image data transmission without data loss. Moreover for spacecraft data training, idle packets will be provided by this component as seen in section2.1.1. Thus it will output data at slightly higher rate. Multiple schemes are possible for this from which two schemes are highlighted, namely GTx and simple LVDS. GTx is a type of inverse SERDES (Parallel to Serial) protocol which handles complete words. Parallel to serial conversion alongwith clock handling is done by SERDES. This simplifies the user logic of the FPGA. However, these are generally used for very high data rates handling up to 6 Gbps for GTx. GTx is also among the low

end of SERDES. Such high speeds are not required by SIC firmware and will add unnecessary complexity.

Another approach is to use LVDS. This is an efficient and simple design adequate for SIC firmware data rates. Furthermore, it is also possible to get parallel to serial conversion of SERDES by using built in macros of FPGA. The architecture of Image Data Transmission is shown in

**Figure 4-6**. Image Data transmission will provide output in two variable frequencies to accommodate any future upgradability. Read and write width of 64 bits is used for this FIFO with depth of 1024. Thus it takes 2 BRAM.

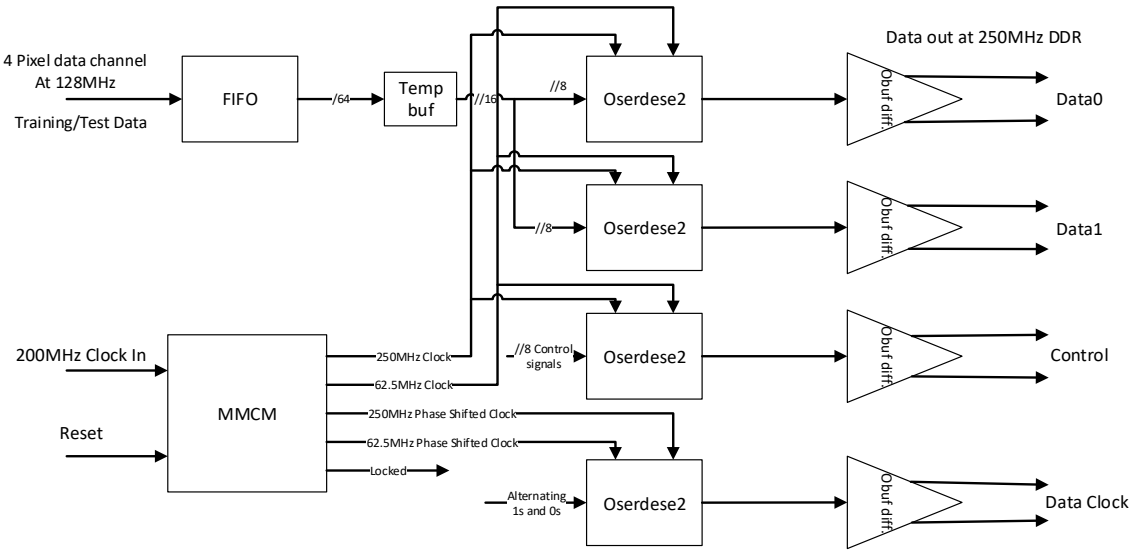



**Figure 4-6** Image Data Transmission: Baseline Architectural Design.

## 4.2.2 Reset Architecture

Reset is an important consideration of any firmware. It serves two purposes. Firstly, it provides the firmware an ability to initiate in a known state. Secondly, it provides the ability to restart and halt the processes if required. To achieve this, primary reset of the signal can be generated within the FPGA or provided by the Master that controls the SIC firmware. However, controlling reset from outside can be dangerous because then it can be prone to EMI which can generate false resets. This drives the reset control of the SIC firmware to be generated within FPGA as baseline architecture.

As firmware is running on a system clock, it should be in reset state when this clock is unstable. Clock stability status is provided by MMCM as locked output, which will be only active when clock is stable. Thus active low reset architecture is used in this work to automatically reset the system when the clock is unstable. An additional aspect of this architecture is that it is generated synchronous to internal MMCM frequency, which is different from the system clock frequency. Thus, it generates an asynchronous reset which will be reflected for the complete reset architecture.

The BIC is the master of SIC firmware which controls its operation. Thus it should provide a reset that should also not generate any EMI problems. This can be easily accommodated in communication without compromising EMI. However, it will not be possible to activate this reset fast enough to meet the requirement of reset. This drives additional reset generation capability in SIC firmware for timely monitoring of anomaly behavior of firmware or firmware controlled hardware. These sources will be discussed in more detail in Chapter5. This reset is termed as Error Reset. The reset Architecture is provided in Figure 4-7.The complete reset logic is running on the system clock to remove bad timing of asynchronous gated resets.

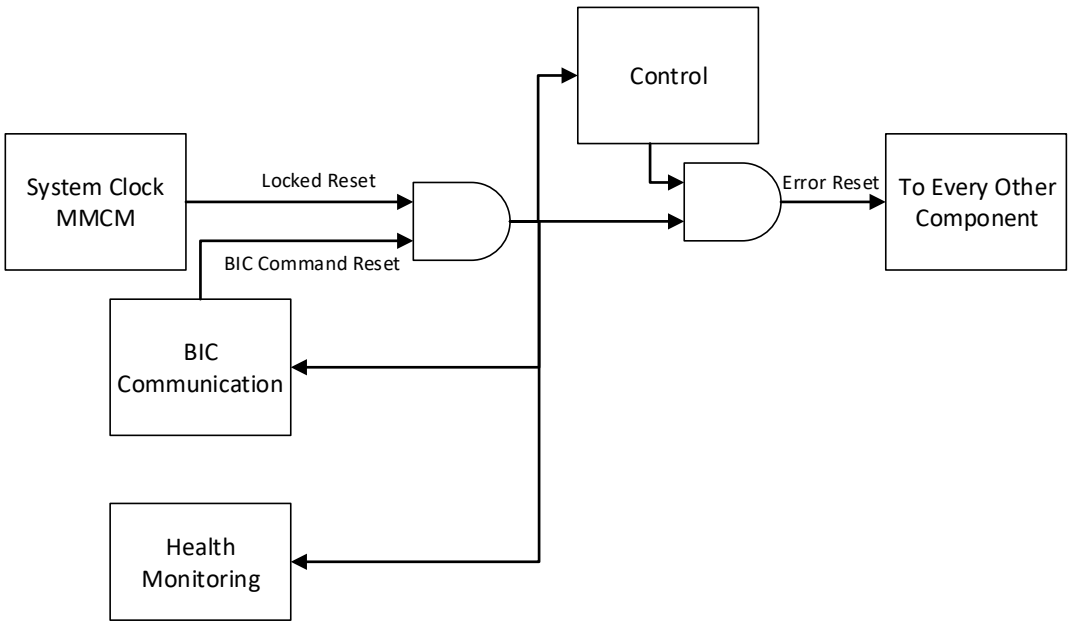



**Figure 4-7**SynchronizedReset Baseline Architecture.

### 4.2.3 Clock Architecture

Clocks are required for any operation in an FPGA. High speed clocks are used for data handling while low speed clocks are used mainly for communication. High speed clocks required are derived from the data handling chain described in section4.2.1. All other components have very small clocking requirements. However, all components of firmware will not be operated at the bare minimum clocks required. Operating every component on a special clock will require large number of clocking buffers. Every FPGA has limited number of clocking buffers available.

Thus the system clock is often introduced in FPGA firmware which runs almost every component. To deduce the system clock required for SIC firmware, clocks required for five components of data handling chain are considered. These are listed in Table 4-7.

**Table 4-7** Clock Considerations of Data Handling Chain.

| **Component** | **Write Clock** | **No. of Inputs** | **Input Sample Size** | **Input Data rate possible** | **Read Clock** | **No. of Outputs** | **Output Sample Size** | **Output Data rate Possible** |
|---|---|---|---|---|---|---|---|---|
| Sensor data reception | 40 | 16 | 2 | 1280 | 8 | 16 | 10 | 1280 |
| Pixel Sequencing | 8 | 16 | 10 | 1280 | 8 | 16 | 10 | 1280 |
| Binning | 8 | 16 | 16 | 2048 | 8 | 16 | 16 | 2048 |
| Auxiliary Data Writing | 8 | 16 | 16 | 2048 | 32 | 4 | 16 | 2048 |
| Image Data Transmission | 32 | 4 | 16 | 2048 | 200/250 | 2 | 2 | 800/1000 |

It can be observed that the read clock of Image Data Transmission is the highest frequency clock required. However, if that clock will be made the system clock, then it will be running throughout FPGA and clearly it's not required by other components. This will unnecessarily increase the EMC/EMI problems for the complete FPGA. The second highest frequency clock required is 32MHz which can easily accommodated by

all remaining components without creating major EMC/EMI issues. Thus, 32MHz is opted as baseline system clock frequency of SIC firmware. This clock will be routed to every component while a higher local clock will be used within Image Data Transmission component.

It can be seen from sections4.1.1and 4.1.2 that the data reception rate by firmware is higher than the sensor output data rate. This is especially the result of binning. The firmware output data rate highlighted in Table 4-7 is also higher than the minimum data rate required in the spacecraft data budget to accommodate training data. High speed clocks required for the design are provided in Table 4-8. Four of the high speed clocks will be reconfigurable while others are fixed. Thus at least one fixed and one reconfigurable MMCM is required for SIC firmware.

**Table 4-8** High Speed Clocks required by SIC Firmware.

| S.No. | Clock | Frequency (MHz) | Phase (degree) | Description |
|---|---|---|---|---|
| 1 | System Clock | 32 | 0 | Running throughout FPGA |
| 2 | Sensor Master Clock | 4 | 0 | Provided to Sensor |
| 3 | Delay Reference Clock | 200 | 0 | Provided to IDELAYE2 |
| 4 | OSERDESE2 input clock for data | 50/62.5 | 0 | Data input clock for ISERDESE2 |
| 5 | OSERDESE2 Output clock for data | 200/250 | 0 | Data output clock for ISERDESE2 |
| 6 | OSERDESE2 input clock for data clock | 50/62.5 | 22.5 | Phase shifted clock to provide clock input shifted to data |
| 7 | OSERDESE2 Output clock for data clock | 200/250 | 90 | Phase shifted clock to provide clock output shifted to data |

Low speed clocks include CCD, SPI and $I^2C$ communication clocks among others which will be discussed in Chapter 5. These low frequency clocks can be easily manufactured by running a counter on system clock. Clock architecture is shown inFigure 4-8.

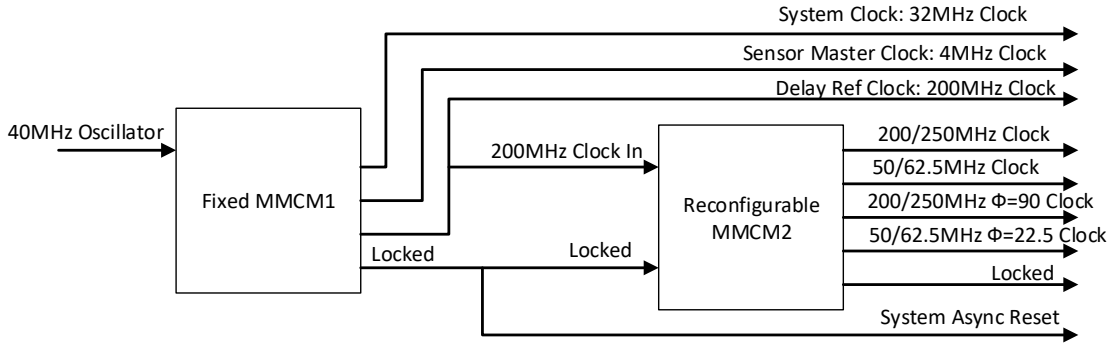



**Figure 4-8** SIC Firmware Clock Architecture

The last consideration required for clock architecture is the input oscillator for the primary Mixed Mode Clock Manager (MMCM). The MMCM houses multipliers and dividers which makes different frequency generations a lot easier. This ensures as a large subset of input oscillator frequencies possible. In that light, 40MHz is chosen which can generate output frequencies with integer multipliers and dividers. Although the MMCM can also use limited fractional multipliers and dividers, it is much easier to manipulate and reconfigure frequencies with integer multipliers and dividers.

## 4.3 FPGA Selection Criteria

Ever growing increases in the technology and applications of FPGAs have revolutionized industry. This pushes FPGA manufacturing processes in number of ways creating multiple vendors. FPGA selection has grown from a simple process with handful of considerations to a systematic process creating ripples throughout a project lifecycle. This calls for allocation of resources (time, budget and personnel) to set things right fromthe start. The purpose of this allocation is to compare[32] bare minimum requirements of a project with FPGAs currently available in the market. FPGA requirements are deduced from baseline firmware architecture discussed in section4.2. Some of the criteria used for FPGA selection trades available in market is provided below.

### 4.3.1 LUTs and Registers

FPGAs are made up of multiple LUTs and registers. These are used to store logic and its outcome, which is in turn fed to another logic or provided as output. The more LUTs and registers an FPGA has, the more freedom thedesigner has. This decreases the

engineering development time, which is often the dominant development cost. LUTs are also linked to the power consumptionof FPGA. The more concise the design is, the more power it utilizes. The trends in the satellite industry is always to go for lower power because power is a limited resource especially for smaller satellites. Thus the more an FPGA has capacity, the better it performs. However, to incorporate more resources in high count devices, the base power required to operate them also increases. So it's not always a linear curve.

Thus the number of LUTs required for a device should be higher than required, but not too high. This calls for estimation of required LUTs based on architecture. It is difficult to estimate LUTs based on requirements. The easiest way to estimate required LUTs is look at previous work. Then best approach is to estimate LUTs based on analogy. As a general rule of thumb, an FPGA should have fifty percent more LUTs than required at the start of project.

### 4.3.2 Block RAMs (BRAMs)

BRAMs are temporary storage devices provided in an FPGA. These are used for FIFO and temporary memory. Based on architecture, it's easy to estimate total number of BRAMs required. It can be estimated based on simple arithmetic of multiplying width by depth and comparing it to FPGA BRAM size. Alternatively, similar FIFO or memory can be made with IP and resultant BRAMs required can be seen in summary. BRAMs are available in the form of blocks. The step size available is 0.5 BRAM which is typically 18kbit chunks of memory. Similar to LUTs, the selected FPGA must have fifty percent more BRAMs than required at the start of project.

### 4.3.3 Power Consumption

As discussed in section4.3.1, power depends on the LUTs count and the satellite industry mostly strives for low power solutions. However, this may change depending on the mission objective. Capability based missions are mostly driven by work that is already done and the solution is often low power one. Needdriven missions require more power as the focus is often on performance. Power consumption is often estimated by HDL design suite based on the implemented design. Thus the user must keep power estimate in check throughout design and development.

### 4.3.4 Inputs/Outputs (I/Os) and bandwidth

I/Os define the links of an FPGA with theoutside world. This can define electrical characteristics of inputs and outputs as well as number of connections. But mostly the defining characteristic is I/O bandwidth. This defines how well FPGA can cope with a data budget and number of channels necessary to overcome that budget.

### 4.3.5 Device Cost

Cost can be a deciding factor in many cases but it may or may not be the main factor for FPGA selection. If an FPGA is used in a low volume product, the cost of a project may depend more on design and development, testing and qualification cost. However, if it's designed only once, then device cost can be ignored to some extent. The latter is mostly the case for satellite design but the satellite industry is moving towards bulk development these days.

### 4.3.6 Software Cost

Software provides a development environment for the FPGA. FPGA cost can be a lot but development software is used more often in a project than the actual device. With easier support and more realistic device modeling, it can be much more logical to go for limited devices with free development software. The devices supported are often at the low end of cost spectrum as well. However, free or low-priced software also tendsnot to be as robust or may overlook critical FPGA paths that may cause much higher cost later in projectby increasing development time and hence the overall project cost.

### 4.3.7 Packaging

The satellite industry requires the device form factor to be as small as possible. However, the smaller form factors often come with limited functionality. Packaging should also take into account pin count and heat dissipation requirements. Package chosen should provide good signal integrity and ease of board mounting for smooth operation.

### 4.3.8 Performance Optimization

Performance is a relative term and what is considered good for one project may not mean anything for another. An FPGA cannot give best performance in every way possible. Each device is optimized for either low cost, low power, best price performance or highest system performance.

### 4.3.9 Temperature Grades

FPGA devices come with different temperature grades. High quality devices will endure extreme temperatures or steep thermal gradients. Tagging of these temperature grades vary from one manufacturer to another. On broader terms, categories can be called Military, Industrial and Commercial.

### 4.3.10 Speed Grade of Device

Speed grade provides maximum operational frequency and latencies. An FPGA doesn't operate on one frequency, but multiple clock frequencies are used throughout the design. Speed grade provides different options including I/O and memory interface data rate, path delays, configurable delay in input, clock generator (MMCM and PLL in Xilinx) maximum operation frequency, clock cycle distortion and skew, etc. Five options are available in Xilinx including -1, -2, -2LE, -2LI and -3.

### 4.3.11 Clocking Resources

An FPGA generally has fixed clock generating resources. These are often Phase Locked Loops (PLLs) and Mixed Mode Clock Mangers (MMCMs). Their names may change with different vendors. These are used to generate clocks necessary for the design. In addition to that, it often has fixed global and regional buffers which are used to connect output/input of MMCMs and PLLs. The more clocking resources FPGA has, the more freedom user has.

### 4.3.12 Radiation Tolerance/Hardening

FPGAs, like all other hardware used in space, areexposed to high levels of radiation. However, the risk is more severe with all the data manipulation that takes place inan

FPGA. This is often a point of no compromise for the high-performancespace projects where a radiation-tolerant FPGA costs a tiny amount compared to the complete project cost. But for low-cost projects like this work, a heritage FPGA is normally used with results of radiation testing of commercially available FPGAs.

## 4.4 Selection of FPGA

Based on the FPGA selection criteria discussed in section4.3, Table 4-9 shows these criteria in light of this work. Required resources are estimated in light of firmware baseline architecture. At this stage, a 50 percent margin is provided in every resource for FPGA selection. This ensures accommodation of development changes and future work that may arise. Table 4-9shows the selection of the Xilinx FPGA XC7k70TFBG676C for this project as it meets resources estimated at this stage with enough margins. Furthermore, the device is selected in a package that makes it easy to migrate to high LUT count devices.

**Table 4-9** FPGA selection Criteria, required and XC7k70TFBG676C resources.

| S.No. | Criteria | Related to | Priority | Required | With Margin | XC7k70T FBG676C |
|---|---|---|---|---|---|---|
| 1 | LUTs, Registers Count | Firmware Architecture | 1 | 15000, 15000 | 30,000, 30,000 | 65,600, 82,000 |
| 2 | BRAMs | Firmware Architecture | 2 | 60 (36 kB) | 90 | 135 |
| 3 | Power Consumption | Power Budget | 7 | NA | NA | Implementation dependent |
| 4 | Useful for FW Inputs/Outputs and bandwidth | Firmware Architecture | 3 | 150 with maximum at 333Mbps | 225 with maximum at 500Mbps | 300 with maximum at 1.25Gbps |
| 5 | Device Cost | Budget | 8 | NA | NA | $178.1[33] |
| 6 | Software Cost | Budget | 10 | NA | NA | Freely available |
| 7 | Packaging | PCB design | 11 | NA | NA | FBG |
| 8 | Performance optimizati | Firmware Architecture | 4 | Optimized at High performan | Optimized at best | Optimized at best price |

| | | | | | | |
|---|---|---|---|---|---|---|
| | on | | | ce | price performance | performance |
| 9 | Temperature Grade | Operational Temperature | 9 | NA | NA | 0 to $85^0$C |
| 10 | Speed Grade | Firmware Architecture | 5 | -1 | -1 | -1 |
| 11 | Clocking Resources (CMTs) | Firmware Architecture | 6 | 2 | 3 | 6 |

# 5 DETAILED FIRMWARE DESIGN

In this chapter the firmware development is done for the baseline design selected in Chapter4. The firmware architecture has twelve components and each is described here. The XC7k70TFBG484C FPGA device selected in Section 4.4, belongs to the No. 7 series[34], which contains four families of Xilinx FPGAs. This device is within the Kintex-7[35] series which has following IO resources available.

- Four HR Banks
- Two HP Banks
- Two GTx Transceivers (eight each)

All of these resources are bonded out in this device but GTx transceivers will not be utilized in this work as data budget required is very low. Bonded out means that all these banks available in the FPGA die are also connected to FPGA pins for user access. Each HP and HR bank constitutes 50 pins, which comprise 48 differential pins (24 differential pairs) and 02 single-ended pins. Two of these IO pin form an IO tile with resources accessible by the user. The HP Bank IO tile constitutes of resources shown inFigure 5-1.

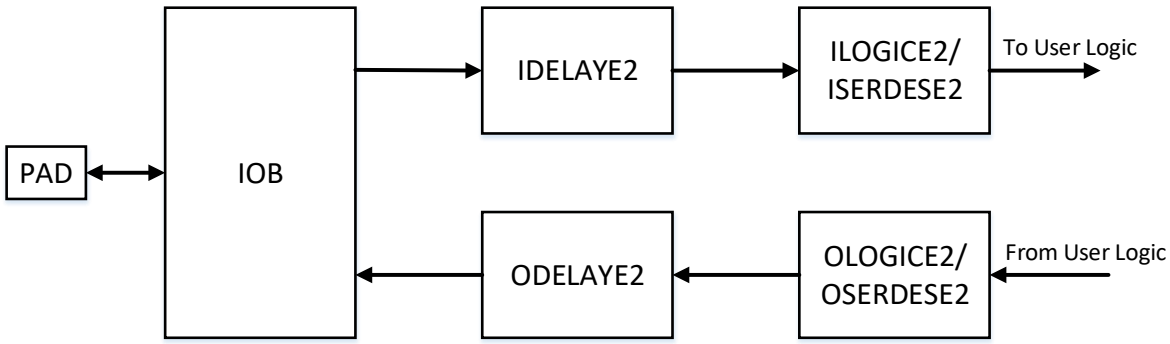



**Figure 5-1**HP IO Tile[36]

HR bank IO tiles don't have the ODELAYE2 component. The rest of the tile structure is same for the HR bank also. These IO tile resources are used extensively in the firmware discussed below. These are accessed by primitives which are configurable basic building blocks of the FPGA. Apart from IO tile resources, other primitives are also available[37]. By configuration, it implies that a primitive is targeted to provide specific function operating in specific mode. It is done by setting the attributes.

Furthermore, to define the signals on which these operation will be done, ports are used which are connected to user signals.

All user clocks are given inputs to theFPGA on specialized differential clock pairs called clock capable inputs. These pins provide dedicated, high speed access for internal global and regional clocks which provides necessary timings important for any clock. Every bank has four clock capable pins with two Single Region Clock Capable (SRCC) and two Multiple Region Clock Capable (MRCC) pins. Each FPGA is divided into clocking regions and SRCC pins can provide clock to only one region while MRCC can provide clock access to the complete FPGA. Details about clocking and dedicated device buffers for clock routing is provided in clocking[38] sources.

Firmware operates on clock edges and most of this firmware design operates on the rising edge of system clock. If any signal is not generated on the rising edge of system clock, it will be misinterpreted as having previous state and not present one. To remove such ambiguities, everything is generated on rising edge of system clock. However, interfacing to independent systems cannot be done following such strict timing requirements as the source is often not synchronized with system clock. These interface signals are synchronized to the system clock domain prior to its usage. One such important signal used for all components is asynchronous reset signal discussed in section4.2.2 which is synchronized to the system clock before usage. Every component uses this signal. Thus, a reset Synchronization Bridge is required in every component.

The data handling chain discussed in section4.2.1 has five components. Each of these components contain FIFOs which can either be independent or common clock FIFO. Each of these FIFOs requires asynchronous active high reset which needs to be activated before operation. Two prerequisites of this reset are that it should be given when operational clocks are stable and it should have a width of three[39] clock cycles of the slowest operation clock. It is seen in section4.2.2 that system reset is active low and is generated when clocks are unstable, which doesn’t meet FIFO reset requirements. A FIFO reset signal will be generated in their respective components. FIFOs also provide programmable full functionality which shows that FIFO is partially full up to certain threshold defined. This functionality simplifies working of state machines.

Similarly, reset signals for all primitives need to be generated because those are also active high resets. If multiple resets are required within a component, these are

combined into one reset if all have similar timing requirements. In case timings are different and both are asynchronous, then a long timing reset is generated and used by every primitive.

For five components of data handling chain seen in section4.2.1 and the image acquisition component, handling of stop and abort commands is also required. The working of these commands in all six components is similar, thus it is provided here in general. In case of the stop command received from operational control component, the component will wait for an even number of frame handling cycles before switching back to idle. A frame consists of four rows corresponding to every band data provided by sensor including PAN, Red, Blue and Green. This is done to ensure that binning component will have enough data to successfully complete its process.

Stop signifies a controlled end of operation, thus the behavior of the SIC firmware output will remain same for every stop command irrespective of the time it is given. Complete data will be provided at output while data generation by sensor will be stopped after even frame counts. After the reception of a stop command, it will also provide done flag depicting that the stop operation has been completed. However, in case of an abort, everything is reset instantly halting FIFO reading and writing. This will flush all FIFOs and component outputs will depend on the instance abort command is provided. Every component will promptly switched to “Idle”. This behavior is also highlighted in corresponding state machines shown in this chapter. All of these components will also provide acknowledgments of start, stop and abort commands.

## 5.1 Sensor Data Reception

This component will take input from sixteen sensor data channels along with the sensor clock. Each data channel provides 10 bit sample in DDR mode on sensor data clock. Data synchronization will also be done in this component. It will provide sixteen pixels output at system clock of 32MHz. Each pixel will be 10 bits wide. Detailed implementation of this component is provided in this section.

### 5.1.1 Buffer Input

As seen in section4.2.1.1, IBUFDS is chosen to take data and clock input from sensor. There are 17 IBUFDS used which is available in 17 IO tiles. All these use a single bank

thus facing relatively same path delays and power fluctuations. Sixteen buffers are used to take sensor data input while the seventeenth one is used to take sensor data clock input.

Attributes set for both clock and data buffers are same. The attributes used for this design are given inTable 5-1. The sensor provides data with 100 ohm differential impedance, thus it's required to provide matched impedance. This impedance should be as close to the receiver as possible. This buffer also provides 100 ohm impedance which eliminates the use of multiple resistors required. It also provides high performance which is useful to get high speed differential signal inputs. As sensor clock is very slow, this buffer is operated in low power low performance mode.

**Table 5-1** Sensor Data Reception: IBUFDS attributes.

| Serial No. | Attribute Name | Value |
|---|---|---|
| 1 | DIFF_TERM | TRUE |
| 2 | IBUF_LOW_PWR | FALSE |
| 3 | IOSTANDARD | "LVDS" |

The ports used for IBUFDS are given in Table 5-2. The sensor clock is provided on SRCC differential pair input because this clock is only used in this component.

**Table 5-2** Sensor Data Reception: IBUFDS Ports.

| Serial No. | Port Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | I | Input | 1 | P Channel |
| 2 | IB | Input | 1 | N Channel |
| 3 | O | Output | 1 | Single Ended |

### 5.1.2 Delay Control

As seen earlier, data and clock delays are required for synchronization. Buffer output is provided input to IDELAYE2 primitive which provides a controllable delay. To utilize delay primitive, delay control primitive (IDELAYCTRL) must be used to control it. It

provides continuous calibration of delay taps of IDELAYE2 and ODELAYE2 primitives by providing voltage bias. This makes the delay values independent[40] of process, voltage and temperature variations. The ports used by IDELAYCTRL components are given in Table 5-3. The reset port RST is active high reset, but system reset is active low. This reset is an asynchronous reset and it must be kept high for at least 50ns to reset this primitive. Once the system is out of reset, a reset for this primitive is generated with the help of counters.

**Table 5-3** Sensor Data Reception: IDELAYCTRL Ports.

| Serial No. | Port Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | REFCLK | Input | 1 | 200MHz reference Clock |
| 2 | RST | Input | 1 | Active high asynchronous reset |
| 3 | RDY | Output | 1 | Ready Signal Output |

Three reference clocks can be used for IDELAYCTRL primitive with different step sizes possible for every tap. With reference frequencies of 200MHz, 300MHz and 400MHz, step[34] sizes of 78ps, 52ps and 39ps are possible. For a slow speed of 40MHz, a maximum step size of 78ps is chosen. Furthermore, with a speed grade of "-1" chosen in 4.4, only 78ps step size is possible with 200MHz reference frequency input. This must be given reset after configuration of corresponding delay components is done and REFCLK is stable. After 3.22µs of reset pulse, ready signal of the component is given high. User logic will take care of this condition before utilizing delay primitives.

### 5.1.3 Delaying Sensor Channels

The IDELAYE2 primitive can operate in four different modes. Their operation varies with the ability to change delay taps. With FIXED mode, tap delays can only be configured when instantiating. Thus, delay tap values are hard-coded in Hardware Descriptive Language (HDL). With VARIABLE mode, tap delays can be changed but it is only possible in the form of increments or decrements. Tap delays cannot be loaded and it takes number of clock cycles to reach a specific delay. With VAR_LOAD or

VAR_LOAD_PIPE, any tap delay can be given and remote loading of this value is possible.

Provided that the maximum step size achievable is 78ps, this makes the achievable maximum delay with 31 taps to be 2.4ns. With sensor clock of 40MHz, time period of clock is 25ns. Operating in DDR, data changes after every 12.5ns. To achieve training, delays possible should traverse from rising edge to falling edge ensuring sample change in real time. As this is not possible, only the training measurement will be done. It will be seen in detail in which is done on the ground before integration and launch. Even after fixing delay tap values, it would be helpful to have the ability of changing delay taps with ground-based commands. Apart from operation in orbit, it can also smooth out firmware and hardware testing. Thus, this component will be operated in VAR_LOAD mode. Other attributes of IDELAYE2 component are given in Table 5-4.

**Table 5-4** Sensor Data Reception: IDELAYE2 Attributes.

| **Serial No.** | **Attribute Name** | **Value** |
|---|---|---|
| 1 | CINVCTRL_SEL | “FALSE” |
| 2 | DELAY_SRC | “IDATAIN” |
| 3 | HIGH_PERFORMANCE_MODE | "TRUE" |
| 4 | IDELAY_TYPE | "VAR_LOAD" |
| 5 | IDELAY_VALUE | 0 |
| 6 | PIPE_SEL | "FALSE", |
| 7 | REFCLK_FREQUENCY | 200.0 |
| 8 | SIGNAL_PATTERN | "CLOCK"/ “DATA” |

Inputs can be given either by DATAIN or IDATAIN. The former can only be driven from logic while the latter can be driven from an FPGA pin. As data/clock is generated by external source, input is given to IDATAIN port and attribute of DELAY_SRC is set to “IDATAIN”. Port connections along with input/output direction is given in Table 5-5. There are seventeen such IDELAYE2 primitives used for sixteen data channels and one data clock. A delayed clock is used to operate serial side of sixteen ISERDESE2 which takes input sixteen delayed serial data channels.

A delay can only be positive, but by driving sampling of ISERDESE2 input on a delayed clock, it is equivalent to providing negative delay to data channels. By only providing data channel delay, positive delay is introduced in data. A combination of both is used to simulate a range of delay twice the size achievable by 31 taps. This architecture of IDELAYE2 will be used in section5.1.6.

**Table 5-5** Sensor Data Reception: IDELAYE2 Ports.

| Serial No. | Port Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | CNTVALUEOUT | Output | 5 | Open |
| 2 | DATAOUT | Output | 1 | Delayed Sensor Data/ Clock |
| 3 | C | Input | 1 | System Clock |
| 4 | CE | Input | 1 | ‘0’ |
| 5 | CINVCTRL | Input | 1 | ‘0’ |
| 6 | CNTVALUEIN | Input | 5 | Delay Taps |
| 7 | DATAIN | Input | 1 | ‘0’ |
| 8 | IDATAIN | Input | 1 | Sensor Data/ Clock |
| 9 | INC | Input | 1 | ‘0’ |
| 10 | LD | Input | 1 | Load Delay Taps |
| 11 | LDPIPEEN | Input | 1 | ‘0’ |
| 12 | REGRST | Input | 1 | ‘0’ |

### 5.1.4 Clock division

Serial data, generated by the sensor, needs to be de-serialized in pixels. All processing operations defined in previous chapters are done on pixels. As each pixel is 10 bit and data is provided in Double Data Rate (DDR), a pixel is completed in five clock cycles. To perform operation on pixels, this clock should be five time slower. This could be easily achieved by diving sensor data clock by five. The simplest frequency divider primitive available in FPGA is IBUFR. Attributes of this primitive are given in Table 5-6.

**Table 5-6** Sensor Data Reception: IBUFR Attributes.

| Serial No. | Attribute Name | Value |
|---|---|---|
| 1 | BUFR_DIVIDE | 5 |
| 2 | SIM_DEVICE | “7SERIES” |

This primitive is only required for sensor data clock. This clock will serve as a parallel interface of ISERDESE2 discussed in5.1.5. FIFO data writing also uses this clock. Ports for this component are given in Table 5-7.

**Table 5-7** Sensor Data Reception: IBUFR Ports.

| Serial No. | Port Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | O | Output | 1 | Divided Clock Output |
| 2 | CE | Input | 1 | ‘1’ |
| 3 | CLR | Input | 1 | ‘0’ |
| 4 | I | Input | 1 | Delayed Clock Input |

### 5.1.5 De-serialization

A single ISERDESE2 primitive can convert 8-bit serial input to parallel data. Two ISERDESE2 primitives are available in every IO tile. Collectively, these can convert up to 14-bit serial data in DDR mode. As the sensor provides 10-bit pixel values, two ISERDESE2 primitives are used. This primitive can only be used with fixed length samples taken input as is the case with this sensor. One primitive will be used in Master mode while other will be operated in Slave mode. Attributes of ISERDESE2 are given in Table 5-8.

**Table 5-8** Sensor Data Reception: ISERDESE2 attributes for Master/Slave Mode.

| Serial No. | Attribute Name | Value |
|---|---|---|
| 1 | DATA_RATE | “DDR” |
| 2 | DATA_WIDTH | 10 |
| 3 | DYN_CLKDIV_INV_EN | "FALSE" |

| Serial No. | Attribute Name | Value |
|---|---|---|
| 4 | INIT_Q1 | '0' |
| 5 | INIT_Q2 | '0' |
| 6 | INIT_Q3 | '0' |
| 7 | INIT_Q4 | '0' |
| 8 | INTERFACE_TYPE | "NETWORKING" |
| 9 | IOBDELAY | "BOTH" |
| 10 | NUM_CE | 2 |
| 11 | OFB_USED | "FALSE" |
| 12 | SERDES_MODE | "MASTER"/ “SLAVE” |
| 13 | SRVAL_Q1 | '0' |
| 14 | SRVAL_Q2 | '0' |
| 15 | SRVAL_Q3 | '0' |
| 16 | SRVAL_Q4 | '0' |

ISERDESE2 can operate in various memory modes or networking mode depending on the interface. For normal input, as provided by the sensor, ISERDESE2 interface type will be set to Networking. NUM_CE is used primarily for memory interfacing, thus value of NUM_CE doesn’t matter. ISERDESE2 can either take input from IO pad or from IDELAYE2 component resulting in delayed or regular outputs. Various IOBDELAY settings are given inTable 5-9. The IFD and BOTH attributes of IOBDELAY have same effect for this work, thus “BOTH” is used. One ISERDESE2 is operated in Master mode converting eight least significant bits of pixel while second one operating in slave mode converts two most significant bits. This width expansion is only available in networking mode.

**Table 5-9** Sensor Data Reception: IOBDELAY Attribute Setting[38] of ISERDESE2.

| IOBDELAY Value | Combinatorial Output (O) | Registered Output (Q1-Q8) |
|---|---|---|
| NONE | D | D |
| IBUF | DDLY | D |
| IFD | D | DDLY |
| BOTH | DDLY | DDLY |

The ISERDESE2 ports used in this work are given inTable 5-10. Reset port RST is active high reset, but system reset is active low. System reset will be inverted before providing reset to ISERDESE2. This reset should be synchronous to CLK, thus an additional reset Synchronization Bridge is required to reset this primitive. Minimum width of this reset pulse should be one CLKDIV cycle.

DDLY input data is connected to delayed data output of IDELAYE2. The clock handling input side of ISERDESE2 is applied at CLK port which is driven by IDELAYE2 as discussed in section5.1.2 while output handling clock is applied at CLKDIV input driven by BUFR discussed in section5.1.4. These input clocks should be phasealigned[38] which is provided by BUFR. For ISERDESE2 operating in networking mode, output provided on Q1-8 have a latency of two CLKDIV cycles.

**Table 5-10** Sensor Data Reception: ISERDESE2 Ports for Master/ Slave Mode.

| Serial No. | Port Name | Input/ Output | Width | Master Signal | Slave Signal |
|---|---|---|---|---|---|
| 1 | O | Output | 1 | Open | Open |
| 2 | Q1 | Output | 1 | Pixel bit 0 | Open |
| 3 | Q2 | Output | 1 | Pixel bit 1 | Open |
| 4 | Q3 | Output | 1 | Pixel bit 2 | Pixel bit 8 |
| 5 | Q4 | Output | 1 | Pixel bit 3 | Pixel bit 9 |
| 6 | Q5 | Output | 1 | Pixel bit 4 | Open |
| 7 | Q6 | Output | 1 | Pixel bit 5 | Open |
| 8 | Q7 | Output | 1 | Pixel bit 6 | Open |
| 9 | Q8 | Output | 1 | Pixel bit 7 | Open |
| 10 | SHIFTOUT1 | Output | 1 | Slave SHIFTIN1 | Open |
| 11 | SHIFTOUT2 | Output | 1 | Slave SHIFTIN2 | Open |
| 12 | BITSLIP | Output | 1 | Bit slip control signal | Bit slip control signal |
| 13 | CE1 | Input | 1 | ‘1’ | ‘1’ |
| 14 | CE2 | Input | 1 | ‘1’ | ‘1’ |
| 15 | CLKDIVP | Input | 1 | ‘0’ | ‘0’ |
| 16 | CLK | Input | 1 | Sensor Clock (BUFIO) | Sensor Clock (BUFIO) |

| Serial No. | Port Name | Input/ Output | Width | Master Signal | Slave Signal |
|---|---|---|---|---|---|
| 17 | CLKB | Input | 1 | Inverse of Delayed Clock | Inverse of Delayed Clock |
| 18 | CLKDIV | Input | 1 | DividedSensor Clock | Divided Sensor Clock |
| 19 | OCLK | Input | 1 | ‘0’ | ‘0’ |
| 20 | DYNCLKDIVSEL | Input | 1 | ‘0’ | ‘0’ |
| 21 | DYNCLKSEL | Input | 1 | ‘0’ | ‘0’ |
| 22 | D | Input | 1 | ‘0’ | ‘0’ |
| 23 | DDLY | Input | 1 | Delayed Data Input | ‘0’ |
| 24 | OFB | Input | 1 | ‘0’ | ‘0’ |
| 25 | OCLKB | Input | 1 | ‘0’ | ‘0’ |
| 26 | RST | Input | 1 | Reset Synchronized to CLK | Reset Synchronized to CLK |
| 27 | SHIFTIN1 | Input | 1 | ‘0’ | Master SHIFTOUT1 |
| 28 | SHIFTIN2 | Input | 1 | ‘0’ | Master SHIFTOUT2 |

Bit shift operation [38] is synchronous to CLKDIV. The operation varies in SDR and DDR mode. This operation is controlled by BITSLIP port. When active in DDR mode, the bit slip operation alternates between a right shift of one bit and left shift of three bits. An example of bit shift operation is provided for an eight bit word in Table 5-11. Thus it can be seen that with seven bit operations on an eight bit word, complete bit shift is possible. The output of ISERDESE2 is ready to be provided to user logic for further processing which is handled as per section5.1.7.

**Table 5-11** Left and right shift bit operation in ISERDESE2 DDR Mode. Operation is illustrated with example where it can be seen that although DDR operation is not simple left or right, but every combination is achievable.

| Bitslip operation | DDR actual operation | Word | Left bitslip | Right bitslip |
|---|---|---|---|---|
| $0^{th}$ | No operation | 00011000 | 0 | 0 |
| $1^{st}$ | Right shift by 1 | 00001100 | 7 | 1 |
| $2^{nd}$ | Left shift by | 01100000 | 2 | 6 |

| Bitslip operation | DDR actual operation | Word | Left bitslip | Right bitslip |
|---|---|---|---|---|
| | 3 | | | |
| 3rd | Right shift by 1 | 00110000 | 1 | 7 |
| 4th | Left shift by 3 | 10000001 | 4 | 4 |
| 5th | Right shift by 1 | 11000000 | 3 | 5 |
| 6th | Left shift by 3 | 00000110 | 6 | 2 |
| 7th | Right shift by 1 | 00000011 | 5 | 3 |
| 8th same as 0th | Left shift by 3 | 00011000 | 0 | 0 |

### 5.1.6 Training Measurement

As seen in section5.1.2, delays achieved are less than the required half clock period. Thus, delay taps are provided and resulting data is analyzed manually to detect stability. This needs to be done in the testing phase of hardware as delays depend a lot on PCB tracks and may vary from one sensor to another. As hardware is not within the scope of this work, only procedure is highlighted below which will be used once hardware is complete.

- Delay taps of clock along with its Load delay port is mapped to Virtual Input Output (VIO) in firmware.
- Delay taps of all sixteen data channels are combined and mapped to VIO in firmware.
- Load delay port of all sixteen data channels are combined and mapped to VIO in firmware.
- Sensor will be powered on. It will start transmitting idle data.
- Clock delay will be set to maximum of thirty one taps.
- Data delay will be set to zero taps.
- Clock tap will be decremented to zero taps and on every decrement channel data is observed.

- Output is observed for every decrement to see whether any channel is unstable. Mark the taps and corresponding channel where data is unstable. Channel is defined as unstable when the output value keeps switching.
- Once clock taps have reached zero, start incrementing all channel taps. Mark the taps and corresponding channel where data is unstable.
- As delay taps are very small, so only one end of unstable channels is observed.
- For all channels, set tap values as far as possible (with 31 taps) from unstable region.

The end result of this procedure is the knowledge of delay settings of all data channels for unstable values. Delay taps will be set beyond these values where every channel is stable.

### 5.1.7 FIFO Writing State Machine

It is assumed in this section that the training measurement has been done and delay taps are incorporated in the design. This component has independent clock FIFO with a write clock of 8MHz. the sensor provides idle, pixel start marker and pixel data. Idle data is used for training sensor interface to get ready for data reception. For FIFO writing, firmware needs to know what data pattern will be used by thesensor as idle and pixel start marker. FIFO data writing state machine is given inFigure 5-2.

This state machine will be running on a CLKDIV clock. With reception of start from Control component, start is acknowledged and state is switched to “Ready to write”. In this state, it will wait for the sensor to transmit data. Upon reception of data from ISERDESE2, it will check every data sample received. Upon receiving an idle sample followed by pixel start marker, the state is switched to “FIFO writing”. Every channel data received is packed into one word of 160 bits. FIFO writing is done by providing this and write enable to FIFO at the same clock cycle. This will be carried on until sensor keeps on providing row data. Upon completion of row data, state is switched back and the cycle continues.

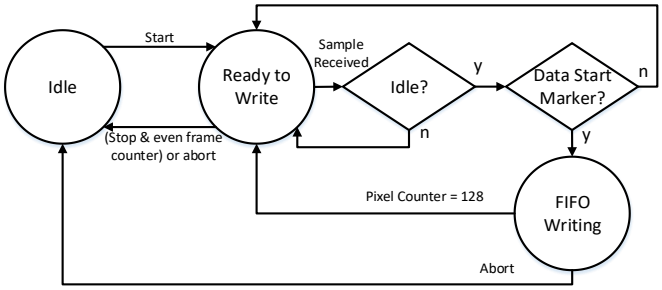



**Figure 5-2** Sensor Data Reception: FIFO Writing State Machine.

### 5.1.8 FIFO Reading State Machine

FIFO reading will be done using the system clock. As the data reading clock is faster than the data writing clock, complete row data is written before reading operation begins. Row completion status will be provided by setting the FIFO core programmable full status port to 128. Upon reception of start, “Ready to read” state is achieved. It will wait in this state for the programmable full signal after which it will switch to “FIFO Reading” state

In this state, FIFO read enable is provided and data will be given output in the next clock cycle. In addition to data output, status is also provided about data. Data validity, row number, last sample of row and last row of the frame is provided as the status. This will facilitate pixel sequencing component and cross verifies data reception. The state machine is given in Figure 5-3.

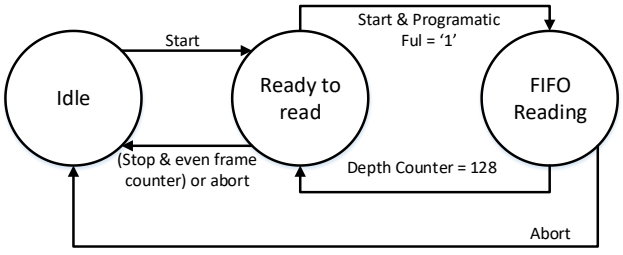



**Figure 5-3** Sensor Data Reception: FIFO Reading State Machine.

## 5.2 Pixel Sequencing

This component will take input sixteen data channels. Each channel provides pixel values of 10 bits each on the rising edge of the system clock. Pixels are sorted by pixel number in this component. It will provide sixteen pixels at output at the system clock.

Each pixel at output will be 10 bits wide. Detailed implementation of this component is provided in this section.

### 5.2.1 FIFO Writing State Machine

After the start reception from Operational Control component, it will wait for data validity. Once the first valid data sample is received, it will start writing these samples in FIFOs assigned for odd pixel reception. Every channel data is written to its corresponding odd FIFO. Every FIFO is 10 bits wide, thus every channel data will be written without any manipulation. When programmable full status of every FIFO is activated, i.e. sixty four pixels written, state will be switched to "Even FIFO writing". With data reception of a complete row, state is again switched to "Idle" where it will wait for next valid row data. Its state machine is given in Figure 5-4.

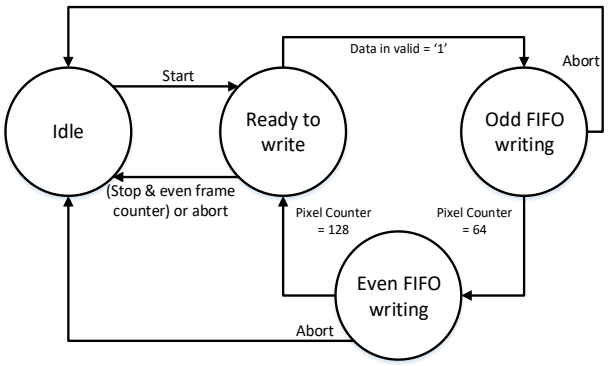



**Figure 5-4** Pixel Sequencing: FIFO Writing State Machine.

### 5.2.2 FIFO Reading State Machine

The read width of FIFO is set to 80 bits. This state machine will work on the system clock. It will wait for pixel sequencing start command reception from operational control component. With this command reception, it will go to "Ready to sequence" state. It will then wait in this state for programmable full flag of all FIFOs which is set to 64. When this flag is set, it will go to "Reading" state.

Odd and Even FIFOs are read simultaneously, each providing 8 pixels. These pixels are rearranged to provide output of sixteen channels in a sequence. A total of sixteen depths are read for first channel and then it switches back to "Ready to sequence" state. As programmable full of all FIFOs are set high at same time, by checking this flag for FIFOs of next channel, it will switch to "Reading" state in consecutive clock cycle. This

process will be repeated for all sixteen channels. The state machine is shown in Figure 5-5. No additional data status is added apart from status of Sensor Data Reception.

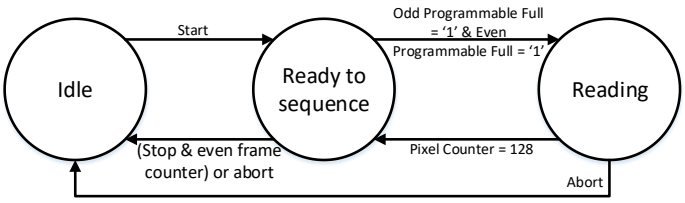



**Figure 5-5** Pixel Sequencing: FIFO Reading State Machine.

## 5.3 Binning

Sixteen inputs are taken by this component. On the rising edge of system clock, each channel will provide 10 bit pixel data. This component will take PAN data and convert 10 bit pixel data to 16 bit pixel data with zero padding. This is directly sent to output. For MS data, four consecutive pixels are added, zero padded and given output. Sixteen pixels are given output with sample size of 16 bit each. Its implementation is discussed below.

### 5.3.1 Binning State Machine

Upon reception of a start command from the operational control component, the state is switched from "Idle" to "Ready to accept data" state. Upon reception of the first data in valid, "PAN Output" state is achieved. It will take samples received, append six zeros and provide this to output. Upon reception of the last sample of PAN, it will switch to "FIFO writing". The next data received will be MS data. Two neighboring pixels are added and written in FIFO. This will take input sixteen 10 bit pixels and write eight summation results of 11 bits in 88 bit wide FIFO. This will be done for all MS bands. State will be switched back to "PAN output" upon reception of last sample of last row. PAN data of the next frame will be handled in this state and upon reception of last sample of last row for even frame, it is switched to "FIFO Reading and Output" state. State Machine of binning is given in Figure 5-6.

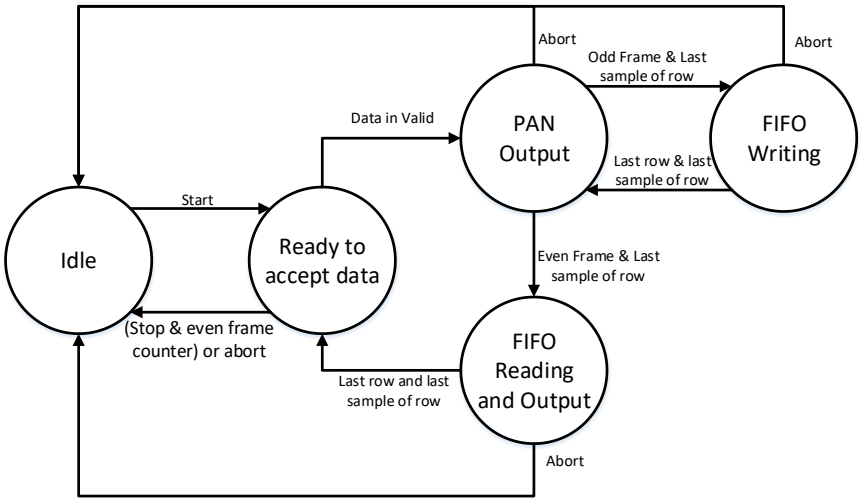



**Figure 5-6** Binning: Binning State Machine.

There is already data written in FIFO. Two neighboring MS pixels along with data read from FIFO will be added. The resultant of this addition will be zero padded and temporarily stored in FIFO. With binning of next eight pixels, sixteen pixels are accumulated and provided on output. This process will be repeated until the last sample of the last row is received. Upon its reception, the state will be switched to "Ready to accept data". Subjected to data availability, this process will be repeated multiple times until the reception of a stop or abort command. Following this approach, the PAN output will be provided for every frame while MS will only output for even frame numbers. In addition to data and its status output mentioned in 5.1.8, PAN/MS flag is also added in data status.

## 5.4 Auxiliary Data Writing

This component takes binning output data and appends auxiliary information in it. 16 pixels with width of 16 bits are taken input while 4 samples of 16 bit are provided output at a time. Auxiliary data is also provided output along with sensor data separated in time.

### 5.4.1 Memory

Auxiliary information will be stored in memory as the baseline design proposes in section4.2.1.4. Memory, generated with Vivado IP[41] is configured with multiple options. The resulting ports of Memory are provided in Table 5-12.

**Table 5-12** Auxiliary Data Writing: Memory Ports.

| Serial No. | Port Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | CLKA | Input | 1 | Operational Clock |
| 2 | RSTA | Input | 1 | Reset |
| 3 | ENA | Input | 1 | Enable Port A |
| 4 | WEA | Input | 8 | Write Enable Port A |
| 5 | ADDRA | Input | 4 | Address for Port A |
| 6 | DINA | Input | 64 | Data In for Port A |
| 7 | DOUTA | Output | 64 | Data Out for Port A |

Memory is configured to be single port RAM. Thus irrespective of mode selection, there will be no collisions possible. Mode selection also provides the state of the output port when memory is being written. Although it has no effect, mode is set to be Read First. The primitive output register helps to reduce the impact of clock to out delay. The core output register isolates the delay of the core output multiplexer, thereby also improving clock to out[41] delay. However, addition of these registers provide latency at output. Thus, output is only available after four clock cycles. As auxiliary data can also change one byte at a time, byte access is easier to manage. This will enable writing of one byte instead of complete 8 byte word which is achieved by enabling byte write. Memory configured has a depth of 10 words. Two such memory cores are used.

### 5.4.2 Cyclic Redundancy Check (CRC)

A CRC is used for error detection. Algorithm is applied on raw data input and upon reception of data with CRC, algorithm is applied again to verify CRC. This algorithm involves bunch of XOR operations which vary depending on the input width. Longer input requires more intensive algorithm and thus requires more FPGA resources. CRC is used only for essential configuration data of sixteen bytes. CRC operation would be done in one clock cycle by using 16 byte long CRC algorithm. But it would be impractical to assign such resources when easier alternate is present. A sixteen bit algorithm is suited for this application as it provides less complexity without a lot of delay. The polynomial used for CRC is:

$$f(x) = x^{16} + x^{12} + x^5 + 1$$

Thus CRC will be performed as a recursive operation. It will be provided eight time separated inputs and the resultant will be sent with data. Like every recursive operation, the starting point should be known to get to meaningful output. Thus, the algorithm needs to be reset before row data input is provided. This initializes internal register to all 1s. Resultant of CRC is set as last two bytes of auxiliary data.

### 5.4.3 Memory Handling State Machine

Memory handling will be done in this state machine which will be operating on system clock. By default, the starting point of this state machine is "Writing both memories". In this state, both memories will be written with same auxiliary data. Auxiliary data needs to be provided at output before the start of any band data. Upon reception of the first valid data, the state is switched to "Output one memory". Read enable signal is given to first memory (MEM0), meanwhile auxiliary data is written in MEM1. As seen earlier, memory configured has latency of four clock cycles, data will appear on output of Auxiliary Data writing component after four cycles. Once auxiliary data is completely provided at output, the auxiliary output complete flag is set which switch state to "Copy". This state copies data from MEM1 to MEM0. It takes a total of 15 clock cycles to copy data, four read latency of MEM0, one write latency of MEM1 and ten clock cycles for ten depths. The state Machine for auxiliary data writing is given in Figure 5-7.

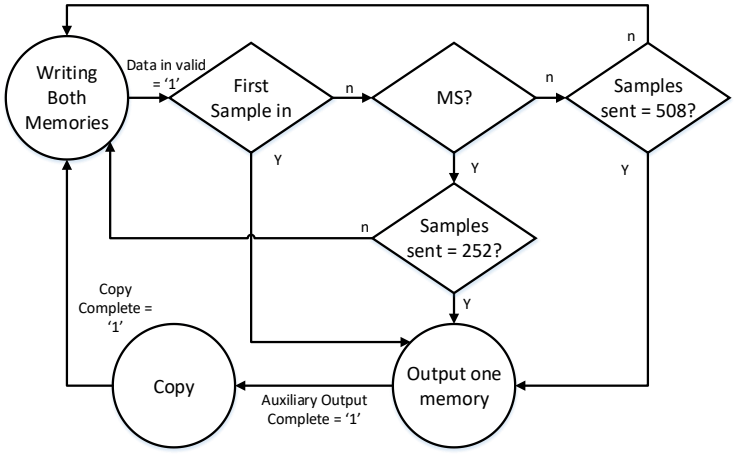



**Figure 5-7** Auxiliary Data Writing: Memory Handling State Machine.

For the subsequent row data, it is checked on data status whether it is MS or PAN. State switching is done four cycles prior to last sample of row to provide time for read latency. During the “copy” state, new auxiliary data is not accepted. But, given that it only takes 15 clock cycles to copy and auxiliary data rate is very slow, auxiliary data is not practically lost. When providing output, this state machine adds a data status of Auxiliary data/ image data in addition to data status provided before in binning.

### 5.4.4 FIFO Writing

FIFO writing is done by concatenating every data sample into one big word of 256 bits and connecting it to data in port. Write enable is connected to the data valid flag of input.

### 5.4.5 FIFO Reading State Machine

This state machine will be running on a system clock. There is one channel to output data from this component and FIFO as well as memory data has to use the same channel. Upon reception of a start command, this state will switch to “Ready to read”. It will wait for the memory handling state machine to provide nine auxiliary data samples. Then it switches state to “Reading”.

One clock cycle read latency of FIFO is taken into consideration to switch one cycle before auxiliary data completion. Four pixels are given output in one clock cycle, thus it takes 256 and 512 clock cycles for MS and PAN respectively to provide output. In any case, it is enough time to complete the copy process of the memory handling state machine. Upon providing last sample at output, the state is switched back to “Ready to Read”. State machine is given in Figure 5-8.

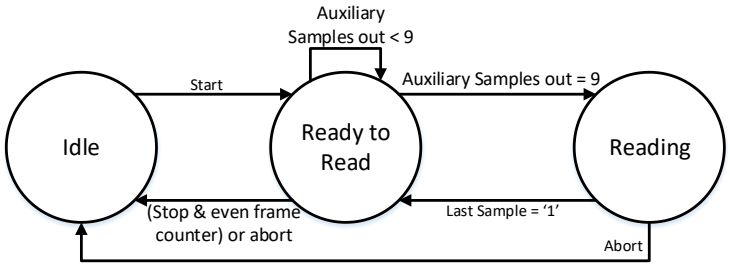



**Figure 5-8** Auxiliary Data Writing: FIFO Reading State Machine.

## 5.5 Image Data Transmission

This component takes an input four pixels of 16 bits each. This data will be transmitted on two data channels. Data status will be provided in a control channel. A data clock is also provided with these three channels. On rising and falling edge of data clock, receiver should sample these three channels.

### 5.5.1 Parallel to Serial Conversion

OSERDESE2 is used to convert parallel data to serial data. This primitive is provided by Xilinx. OSERDESE2 can convert 8 bit parallel data to serial data. Similar to ISERDESE2 discussed in 5.1.5, width expansion of OSERDESE2 is possible up to 14 bits. However to transmit 16 bit pixel data and also data status, multiple LVDS channels with their separate OSERDESE2 are used. As seen in section4.2.1.5, four OSERDESE2 primitives are used for two LVDS data channels, one control channel and one data clock. Attributes for OSERDESE2 are given in Table 5-13.

**Table 5-13** Image Data Transmission: OSERDESE2 Attributes.

| **Serial No.** | **Parameter Name** | **Value** |
|---|---|---|
| 1 | DATA_RATE_OQ | "DDR" |
| 2 | DATA_RATE_TQ | "DDR" |
| 3 | DATA_WIDTH | 8 |
| 4 | INIT_OQ | ‘0’ |
| 5 | INIT_TQ | ‘0’ |
| 6 | SERDES_MODE | “MASTER” |
| 7 | SRVAL_OQ | ‘0’ |
| 8 | SRVAL_TQ | ‘0’ |
| 9 | TBYTE_CTL | “FALSE” |
| 10 | TBYTE_SRC | “FALSE” |
| 11 | TRISTATE_WIDTH | 1 |

OSERDESE2 is operated in DDR mode. It can also provide 3-state control at OSERDESE2 output. As there is a point to point connection with spacecraft for every

SIC, this mode is not required. CLK port is used to drive serial side of ISERDESE2 while CLKDIV port is used for parallel side. The ports used for one LVDS data channel are given in Table 5-14.

**Table 5-14** Image Data Transmission: OSERDESE2 Ports.

| Serial No. | Parameter Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 1 | OFB | Output | 1 | open |
| 2 | OQ | Output | 1 | Serial Output |
| 3 | SHIFTOUT1 | Output | 1 | open |
| 4 | SHIFTOUT2 | Output | 1 | open |
| 5 | TBYTEOUT | Output | 1 | open |
| 6 | TFB | Output | 1 | open |
| 7 | TQ | Output | 1 | open |
| 8 | CLK | Input | 1 | Output Sampling Clock |
| 9 | CLKDIV | Input | 1 | Input Sampling Clock |
| 10 | D1 | Input | 1 | Parallel Input bit 0 |
| 11 | D2 | Input | 1 | Parallel Input bit 1 |
| 12 | D3 | Input | 1 | Parallel Input bit 2 |
| 13 | D4 | Input | 1 | Parallel Input bit 3 |
| 14 | D5 | Input | 1 | Parallel Input bit 4 |
| 15 | D6 | Input | 1 | Parallel Input bit 5 |
| 16 | D7 | Input | 1 | Parallel Input bit 6 |
| 17 | D8 | Input | 1 | Parallel Input bit 7 |
| 18 | OCE | Input | 1 | ‘1’ |
| 19 | RST | Input | 1 | OSERDESE2 Reset |
| 20 | SHIFTIN1 | Input | 1 | '0' |
| 21 | SHIFTIN2 | Input | 1 | '0' |
| 22 | T1 | Input | 1 | '0' |
| 23 | T2 | Input | 1 | '0' |
| 24 | T3 | Input | 1 | '0' |
| 25 | T4 | Input | 1 | '0' |

| Serial No. | Parameter Name | Input/ Output | Width | Signal |
|---|---|---|---|---|
| 26 | TBYTEIN | Input | 1 | '0' |
| 27 | TCE | Input | 1 | '0' |

Based on CLK, CLKDIV and parallel input data, the OSERDESE2 primitive provides different outputs for two LVDS data channels, the control channel and data clock. the output of each OSERDESE2 is connected to the differential output buffer OBUFDS.

**Table 5-15** Image Data Transmission: Differentiating features of Data Channels, Control Channel and Data Clock.

| Ports | LVDS Channel 1 | LVDS Channel 2 | Control Channel | Clock Channel |
|---|---|---|---|---|
| CLK | 200MHz Phase 0 | 200MHz Phase 0 | 200MHz Phase 0 | 200MHz Phase 22.5 |
| CLKDIV | 50MHz Phase 0 | 50MHz Phase 0 | 50MHz Phase 0 | 50MHz Phase 90 |
| D1 | Pixel bit 0 | Pixel bit 8 | 1 | 1 |
| D2 | Pixel bit 1 | Pixel bit 9 | 0 | 0 |
| D3 | Pixel bit 2 | Pixel bit 10 | Last sample of row | 1 |
| D4 | Pixel bit 3 | Pixel bit 11 | Auxiliary/Image | 0 |
| D5 | Pixel bit 4 | Pixel bit 12 | Valid | 1 |
| D6 | Pixel bit 5 | Pixel bit 13 | Band Number (bit 2) | 0 |
| D7 | Pixel bit 6 | Pixel bit 14 | Band Number (bit 1) | 1 |
| D8 | Pixel bit 7 | Pixel bit 15 | Band Number (bit 0) | 0 |

## 5.5.2 MMCME2_ADV

The MMCME2_ADV primitive is used for clock generation that drives four OSERDESE2 in this component as discussed in section5.5.1. It is also possible to generate required clocks with the help of Vivado clocking wizard which generates fixed clocks. However by using this primitive, clock reconfiguration is possible which is discussed in section0. The attributes used for this primitive are provided in Table 5-16. It provides the ability to set MMCM output frequency at instantiation.

**Table 5-16** Image Data Transmission: MMCME2_ADV Attributes.

| **Serial No.** | **Parameter Name** | **Value** |
|---|---|---|
| 1 | CLKFBOUT_MULT_F | 45.0 |
| 2 | CLKFBOUT_PHASE | 0.0 |
| 3 | CLKIN1_PERIOD | 5.0 |
| 4 | CLKIN2_PERIOD | 5.0 |
| 5 | CLKOUT1_DIVIDE | 12 |
| 6 | CLKOUT2_DIVIDE | 3 |
| 7 | CLKOUT3_DIVIDE | 12 |
| 8 | CLKOUT4_DIVIDE | 1 |
| 9 | CLKOUT5_DIVIDE | 1 |
| 10 | CLKOUT6_DIVIDE | 1 |
| 11 | CLKOUT0_DIVIDE_F | 3.0 |
| 12 | CLKOUT0_DUTY_CYCLE | 0.5 |
| 13 | CLKOUT1_DUTY_CYCLE | 0.5 |
| 14 | CLKOUT2_DUTY_CYCLE | 0.5 |
| 15 | CLKOUT3_DUTY_CYCLE | 0.5 |
| 16 | CLKOUT4_DUTY_CYCLE | 0.5 |
| 17 | CLKOUT5_DUTY_CYCLE | 0.5 |
| 18 | CLKOUT6_DUTY_CYCLE | 0.5 |
| 19 | CLKOUT0_PHASE | 0.0 |
| 20 | CLKOUT1_PHASE | 0.0 |
| 21 | CLKOUT2_PHASE | 90.0 |
| 22 | CLKOUT3_PHASE | 22.5 |

| Serial No. | Parameter Name | Value |
|---|---|---|
| 23 | CLKOUT4_PHASE | 0.0 |
| 24 | CLKOUT5_PHASE | 0.0 |
| 25 | CLKOUT6_PHASE | 0.0 |
| 26 | CLKOUT4_CASCADE | FALSE |
| 27 | COMPENSATION | “BUF_IN” |
| 28 | DIVCLK_DIVIDE | 12 |
| 29 | REF_JITTER1 | 0.01 |
| 30 | REF_JITTER2 | 0.01 |
| 31 | STARTUP_WAIT | FALSE |
| 32 | SS_EN | "FALSE" |
| 33 | SS_MODE | "CENTER_HIGH" |
| 34 | SS_MOD_PERIOD | 10000 |
| 35 | CLKFBOUT_USE_FINE_PS | FALSE |
| 36 | CLKOUT0_USE_FINE_PS | FALSE |
| 37 | CLKOUT1_USE_FINE_PS | FALSE |
| 38 | CLKOUT2_USE_FINE_PS | FALSE |
| 39 | CLKOUT3_USE_FINE_PS | FALSE |
| 40 | CLKOUT4_USE_FINE_PS | FALSE |
| 41 | CLKOUT5_USE_FINE_PS | FALSE |
| 42 | CLKOUT6_USE_FINE_PS | FALSE |

An input clock of 200MHz is provided to this MMCM by the system clock MMCM as shown in section4.2.3. As the phase of output clock is very important for generation of the data clock, it is important to utilize MMCM for clock networkdeskew[38]. BUFGs are used to provide a feedback clock to MMCM and user logic. In addition to input and output clocks provided by MMCM, it also provides MMCM reconfiguration ports. These reconfiguration ports are operated at rising edge of system clock that is provided input at DCLK port. Ports of this primitive are provided in Table 5-17.

**Table 5-17** Image Data Transmission: MMCME2_ADV Ports.

| Serial No. | Parameter Name | Input/ Output | Width | Value |
|---|---|---|---|---|

| Serial No. | Parameter Name | Input/ Output | Width | Value |
|---|---|---|---|---|
| 1 | CLKOUT0 | Output | 1 | CLK0 output to BUFG |
| 2 | CLKOUT0B | Output | 1 | open |
| 3 | CLKOUT1 | Output | 1 | CLK1 output to BUFG |
| 4 | CLKOUT1B | Output | 1 | open |
| 5 | CLKOUT2 | Output | 1 | CLK2 output to BUFG |
| 6 | CLKOUT2B | Output | 1 | open |
| 7 | CLKOUT3 | Output | 1 | CLK3 output to BUFG |
| 8 | CLKOUT3B | Output | 1 | open |
| 9 | CLKOUT4 | Output | 1 | open |
| 10 | CLKOUT5 | Output | 1 | open |
| 11 | CLKOUT6 | Output | 1 | open |
| 12 | DO | Output | 16 | Output port of reconfiguration data |
| 13 | DRDY | Output | 1 | Status of reconfiguration operation |
| 14 | PSDONE | Output | 1 | open |
| 15 | CLKFBOUT | Output | 1 | Feedback buffer input |
| 16 | CLKFBOUTB | Output | 1 | open |
| 17 | CLKFBSTOPPED | Output | 1 | open |
| 18 | CLKINSTOPPED | Output | 1 | open |
| 19 | LOCKED | Output | 1 | MMCM locked status |
| 20 | CLKIN1 | Input | 1 | Output of input clock buffer |
| 21 | CLKIN2 | Input | 1 | ‘0’ |
| 22 | CLKINSEL | Input | 1 | ‘1’ |
| 23 | PWRDWN | Input | 1 | '0' |
| 24 | RST | Input | 1 | Reset Control of MMCM |
| 25 | DADDR | Input | 7 | Address of reconfiguration register |
| 26 | DCLK | Input | 1 | System clock |
| 27 | DEN | Input | 1 | Enable pin for reconfiguration operation |

| Serial No. | Parameter Name | Input/ Output | Width | Value |
|---|---|---|---|---|
| 28 | DI | Input | 16 | Input port of reconfiguration data |
| 29 | DWE | Input | 1 | Write/Read control of reconfiguration data |
| 30 | PSCLK | Input | 1 | '0' |
| 31 | PSEN | Input | 1 | '0' |
| 32 | PSINCDEC | Input | 1 | '0' |
| 33 | CLKFBIN | Input | 1 | Feedback buffer output |

### 5.5.3 Clock Reconfiguration State Machine

MMCME2_ADV provides runtime reconfiguration of its parameters, mainly dividers and multipliers. This enables reconfiguration of clocks generated. A set of procedures is required to reconfigure MMCM, which is highlighted in a Xilinx Application note[42]. The registers are primarily set based on MMCME2_ADV addresses provided. By receiving a start reconfiguration command from operational control component, first step is to read previously stored register values. This is done in the “Reading Old Parameters” state by providing the address of register and read enable to MMCME2_ADV. Addressing and register map is provided in application note[42]. To perform any reconfiguration operation, MMCM should be in reset state. This state puts MMCME2_ADV in reset state. This reading is required because some bits are reserved and these values should be retained when writing new parameters.

Reception of active signal on DRDY port of this primitive signifies that reading operation has been done. State is switched to “Parameter Updates”. New parameters are updated and reserved bits are kept intact. When a word is updated, the state is switched to “Writing New Parameters”. Address is provided again with write enable and upon reception of DRDY active signal, state is switched again to “Reading Old Parameters”. Following this cycle eight more times, nine registers are written signifying parameter update of all required registers. Upon completing the reconfiguration operation, the primitive is taken out of reset. After some time, MMCM locked signal is set high which signifies that newly set output clocks are stable. State is switched back to “Idle” for the next reconfiguration if required. The state machine is given in Figure 5-9.

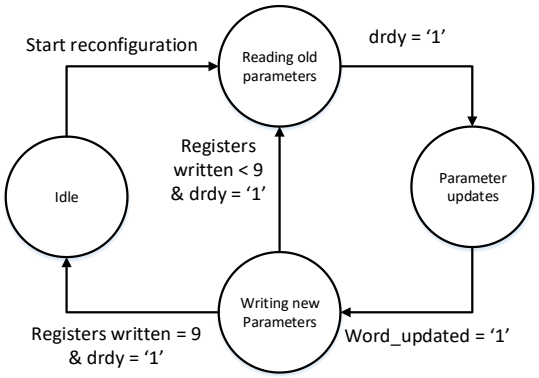



**Figure 5-9** Image Data Transmission: Clock Reconfiguration State Machine.

## 5.5.4 FIFO Writing

FIFO write enable is connected to data input valid. Data in port of FIFO is provided by concatenating all four data channels. Thus FIFO writing will be done at system clock.

## 5.5.5 FIFO Reading State Machine

This state machine is working on clock generated for data OSERDESE2 parallel interface as given in Table 5-15. FIFO used in this component is independent clock FIFO. This state machine switches to "Idle Data Output" state where fixed repetitive pattern is provided input to both data and control OSERDESE2. This idle data is used for training of receiver as discussed in2.1.1. It waits in this state for deactivation of a FIFO Empty flag for switching to "Data Output" state. Data output port provides 64 bit data which contains four pixel information. This is temporarily stored in a register and data is provided to OSERDESE2 pixel by pixel. Lowest byte of pixel is sent to data channel 1 and higher byte to channel 2 OSERDESE2. Control channel input provides the status of data channels. State Machine is provided inFigure 5-10.

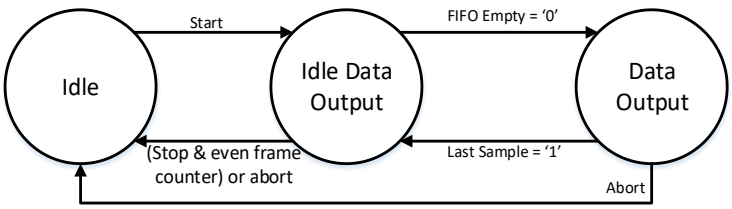



**Figure 5-10** Image Data Transmission: FIFO Reading State Machine.

## 5.6 Spacecraft Test Data Generation

An incremental pattern is generated for the SIC to spacecraft link verification. It is generated on a sixteen bit counter that starts at zero upon reception of Start spacecraft data generation and goes to $2^{16}$-1. Data is provided to Image Data Transmission and that component is configured to take test pattern input. Similar data needs to be generated in thereceiver which will be checked bit by bit to received data for link verification. Upon completion of test data, state is switched back to “Idle”. State machine is given in Figure 5-11.

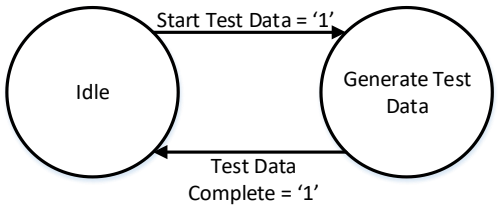



**Figure 5-11** Spacecraft Test Data Generation - State Machine.

## 5.7 BIC Communication

This component enables the SIC firmware to have a communication link with the BIC. Apart from the communication interface, it provides interface with Health Monitoring, Auxiliary Data Writing and Operational Control component. Every SIC firmware has a separate ID that is provided with the help of resistors. Based on that ID, communication is done with the BIC. This component is not controlled by Operational Control and will always be running upon power up. The BIC acts as master which is connected to multiple slave SICs. The BIC can either broadcast its signal to all SICs or communicate directly to a single SIC.

### 5.7.1 SPI Slave Core

This state machine provides SPI handling. This state machine will run on a system clock. The SPI core runs in slave mode as it is operated by the BIC. This core is running in mode 0, which means CPOL is 0 and CPHA is 0. This means that data will be sampled on the rising edge and generated on falling edge. This simple state machine enables transmission and reception of 1 byte data on SPI. Furthermore, it also checks SPI timeout counter. This timer is reset upon detecting the falling edge on SPI clock.

It also enables the control of this core by providing an interfacing to the SPI slave handler discussed in section5.7.2. Upon activating the CS (active low) and detection of rising edge on SPI clock, “SPI Rising Edge Detected” state is switched. Once this switching is done, it signals SPI slave handler that core is busy now and data input provided from now on will not be handled. It will wait for the falling edge of the SPI clock in this state. The Bit counter is incremented upon reception of falling edge and state is switched to “SPI Falling Edge Detected”. This will be done multiple times until one byte is received and transmitted. After handling the complete byte, the state machine will switch to “Continue Check” state. Based on input provided by the SPI slave handler, it will either go back to “Idle” immediately or wait for next SPI rising edge as signaled by SPI slave handler. A Busy signal is deactivated in the idle state until new rising edge is received with active CS. The state machine is given in Figure 5-12.

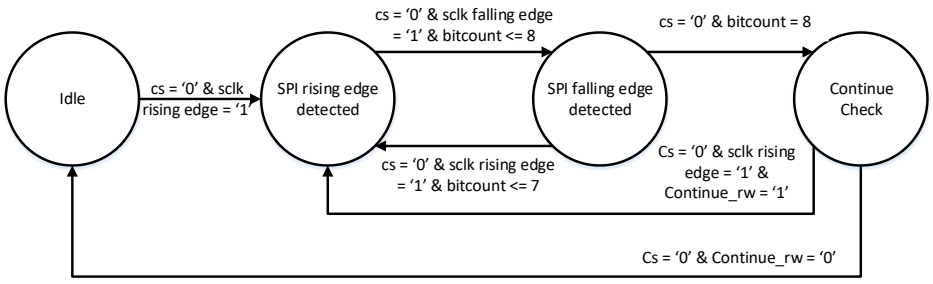



**Figure 5-12** BIC Communication: SPI Slave Core State Machine.

## 5.7.2 Slave Handler

This state machine is responsible to provide control between the SPI slave core and BIC communication state machine. It hosts a tristate control buffer for output port of MISO. By default this buffer is put to a high impedance state. The first byte received by the SPI slave core has the information about broadcasting or dedicated SIC communication. With reception of the first valid byte from SPI core, state is switched to “Slave Address Received”. Upon switching, this buffer is either put in high impedance for broadcasted packet handling or activate signal output for dedicated SIC communication. The slave handler has the packet format knowledge and thus drives SPI slave core. The slave address is provided to the core for transmission. By the time of reception of the next valid data from core, register address is received. This register address is provided to BIC communication state machine. SPI slave handler waits for packet data provision from BIC communication interface. When packet data is provided, it is sent to the BIC

in four SPI cycles corresponding to four byte long register value. After complete transmission, state is either switched to “Idle” or “Data Provided Output” based on Read/write status. After data provision, state is switched to “Idle”. State machine is given in Figure 5-13.

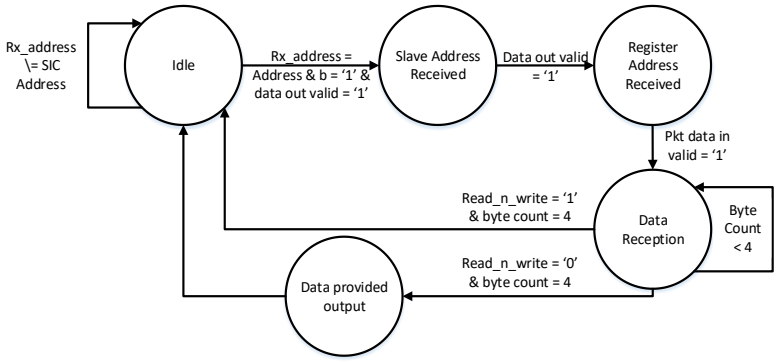



**Figure 5-13** BIC Communication: SPI Slave Handler State Machine.

## 5.7.3 BIC Communication State Machine

Based on SPI slave handler discussed in section5.7.2, the BIC communication interface will wait for register address provision. When a register address is provided, the “Packet Data Provided” state is switched. Data for requested register address is provided to SPI Slave Handler state machine. Based on read/write status, state will either switch back to “Idle” or “Wait for Data”. In case of write request, provided data is written to register address in “wait for Data” state. “Idle” state is immediately switched after data reception. The state machine for BIC Communication is given in Figure 5-14.

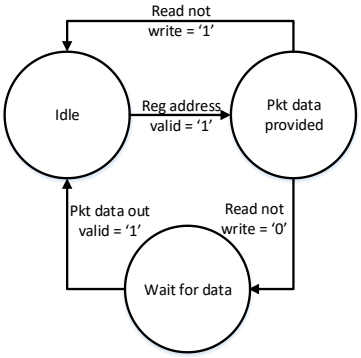



**Figure 5-14** BIC Communication: State Machine.

## 5.8 Initiate Image Acquisition

This state machine operates on system clock. Upon reception of a start command, “Ready to Initiate” state is achieved. Next transition is based on Input provided by the Operational Control Component. If this firmware is set as master, then thecounter increment is done on every clock cycle. When counter becomes equal to frame counter provided, state is transitioned to “Master Enable Acquisition”. In this state, the Master firmware generates two signals, first is Initiate Image Acquisition signal that is going to image sensor signaling start of exposure. Another signal, synchronize acquisition, is provided to other SICs which are operating in slave mode to provide initiate image acquisition signal to their corresponding sensors.

In case SIC firmware is configured as a Slave on SPI, firmware will switch to “Slave Enable Acquisition” state upon reception of this signal. In this state, state machine will wait for a synchronize acquisition signal from the master SIC firmware. Upon reception of this signal, the image acquisition signal is generated. Next state for master and slave enable acquisition states are same. The switching is done within one clock cycle to “Exposure Control” state. Based on input from the exposure control counter provided by operational control, the width of initiate acquisition signal is decided. When this counter is reached, state is switched to “Ready to Initiate” for next frame capture. The state machine is provided in Figure 5-15.

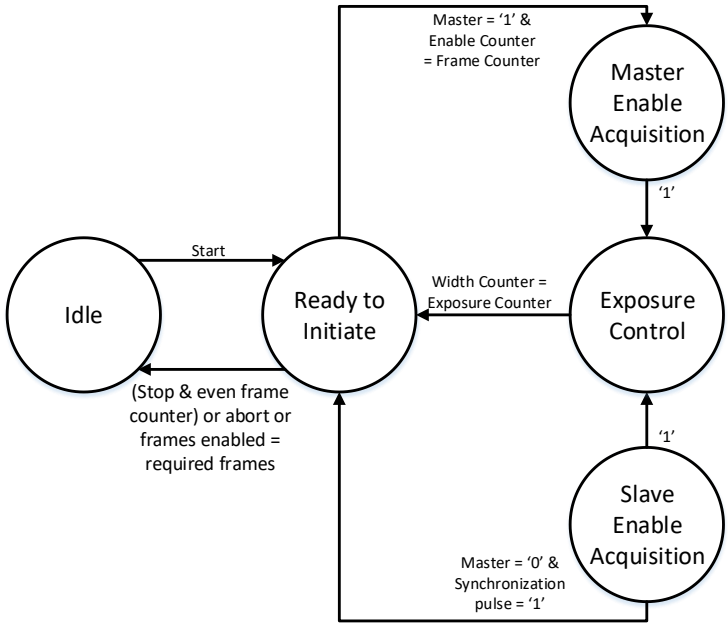



**Figure 5-15** Initiate Image Acquisition: State Machine.

## 5.9 Configure Sensor

This component is used to set parameters specific to an imaging scene, which may vary in number of conditions that are beyond the scope of this work to consider. A simple to understand example of varying light level when satellite passes from ocean to land is analyzed for illustration. Ocean is very good reflector of solar energy, thus a very bright reflection will be incident on sensor. To store such light in limited pixel depth of a fixed number of bits, sensor gain is reduced. After complete imaging, satellite may cross the coast into land. Now if similar gain settings are used, the pixel values generated are suddenly too small to distinguish targets on ground. To compensate for this, gain is increased. This can only be possible if sensor could be configured from one imaging scene to another. Additionally, this component will also generate CCD clock with the help of counters with an accuracy of system clock period.

### 5.9.1 Configure Sensor State Machine

Every sensor provides a communication link which could be used for this purpose, which is the SPI in this work. The sensor will act as a slave while this component will be master. Once the sensor is powered up, operational control component will provide start configuration command to this component. To check that sensor is ready to communicate, the sensor ID is inquired. The sensor should provide a known ID in response to that. Receiving a correct ID signifies that communication is ready and there is no error in the communication interface. In case of an incorrect ID, state is switched to “Error” state. Otherwise, it is switched to “Initial Addressing”. The sensor provides number of memory pages which contain multiple configurable registers. A special SPI packet sequence is provided to sensor in this state which assigns addresses to every memory page. Once this is done, state machine is switched to “Packet Writing” state.

All sensor configuration parameters are provided inputs by the operational control component. This component stores the addresses of every register as constant. In this state, inputs are taken and packet is formed. This packet is then provided to the SPI master core to send to sensor as discussed in section5.9.2. With complete packet transmission, the state is switched to “Packet Reading” where same register is read. If the read value matches the written value, the state is switched back to “Packet writing” for the next register write. If not, state transition is same but the packet is written again.

If there is still a mismatch in data, the state is switched to “Error”. Otherwise this procedure is repeated until every register is written and verified. A configuration done signal is provided to Control Component at the end. The state machine is provided in Figure 5-16.

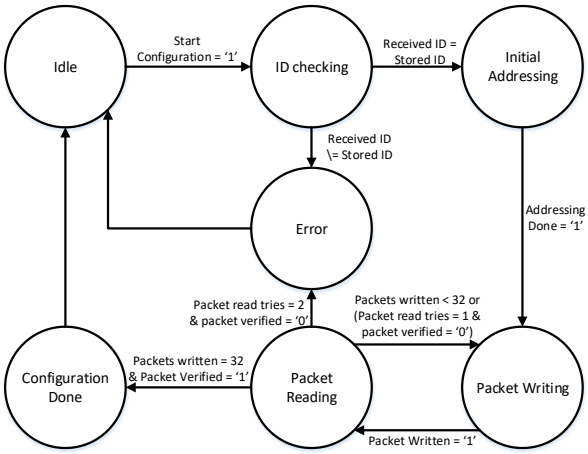



**Figure 5-16**Configure Sensor - State Machine

## 5.9.2 SPI Master Core State Machine

A packet formulated by the Configure sensor state machine is given to this state machine for transmission. A part of this is to generate a Chip Select (CS) and SPI clock signal as this component is acting as SPI master. CS is activated (active low), followed by toggling SPI clock. The state machine for SPI master core runs on system clock but toggling between states is done on the CS and SPI clock. This core generates data on falling edge and samples data on rising edge. However, the first sample on MOSI should be generated before SPI clock rising edge so that slave (sensor) can sample it.

Upon reception of first rising edge, state is switched to “Rising Edge Detected”. It will wait here for falling edge. Upon getting the falling edge, the bit counter will be incremented showing that one bit is handled for both MISO and MOSI. State is then switched to “Falling Edge Detected”. State is toggled in between these states until a 32 bit long SPI packet is complete. After completion of packet, CS is switched to inactive state by CS and SPI clock generation logic. Following that, state is switched to “Idle”. This state machine is independent to the contents and read or write packet type. The state machine is given in Figure 5-17.

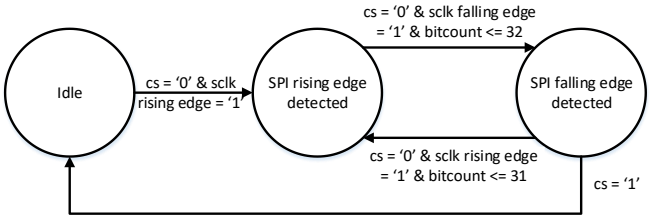



**Figure 5-17** Configure Sensor - SPI Master Core State Machine.

## 5.10 Sensor Power Management

This component is used to provide power management of a sensor. It requires sixteen sequencing switches for power on and off. Provided the large number of sequencing required and the long distance often seen in sensor and sensor controlling electronics, hard lines going from FPGA to sensor is not feasible. Thus it is done using $I^2C$ General Purpose IOs IC (PCF8575[43]).

### 5.10.1 $I^2C$ Bus Control State Machine

$I^2C$ is a multi-master serial communication protocol. Although the architecture used in this work doesn't include multiple masters for communication, handling this is required by standard protocol because of standardized General Purpose IO. This protocol works on two signals, clock (SCL) and data (SDA). This state machine is driven by master core state machine. It only takes control of the bus when the other master is not driving these lines. Sensing other masters and accessing control is described in section5.10.2. The bus control state machine is shown inFigure 5-18.

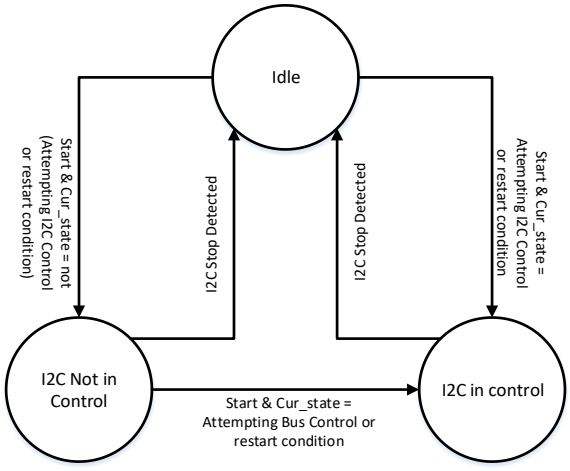



**Figure 5-18** Sensor Power Management: I2C Bus Control State Machine.

## 5.10.2 $I^2C$ Master Core State Machine

Before discussing the master core, it is important to analyze the working of $I^2C$ as its working is different from normal serial communication interfaces. Both signals act as input/output which are pulled up[44] in PCB. To provide '1' at output, high impedance is provided while '0' is driven as is by the FPGA. This state machine is driven by the sensor power management state machine. When start packet transmission is given, this state machine looks for SCL and SDA signal states in "$I^2C$ Bus waiting" state. If both signals are high, it can be deduced that these are not driven by any other device. Master then puts SDA line in low state, this condition (SCL = '1' and falling edge is observed on SDA) is called start[44] condition.

The address of slave is then transmitted. With completion of the address byte, the sensor power management state machine could provide restart condition or stop condition. If either conditions are not provided, master stop driving SDA line but keep driving SCL. The slave provides acknowledgment based on reception of address byte. This byte also contains reading/writing bit control and the slave will either provide data for read request or accept data for write request. Acknowledgment is given after reception of a complete byte. The restart condition is used to keep bus occupied for next packet without releasing it. Stop condition provides bus back in pool for use by other Masters. State Machine is shown in Figure 5-19.

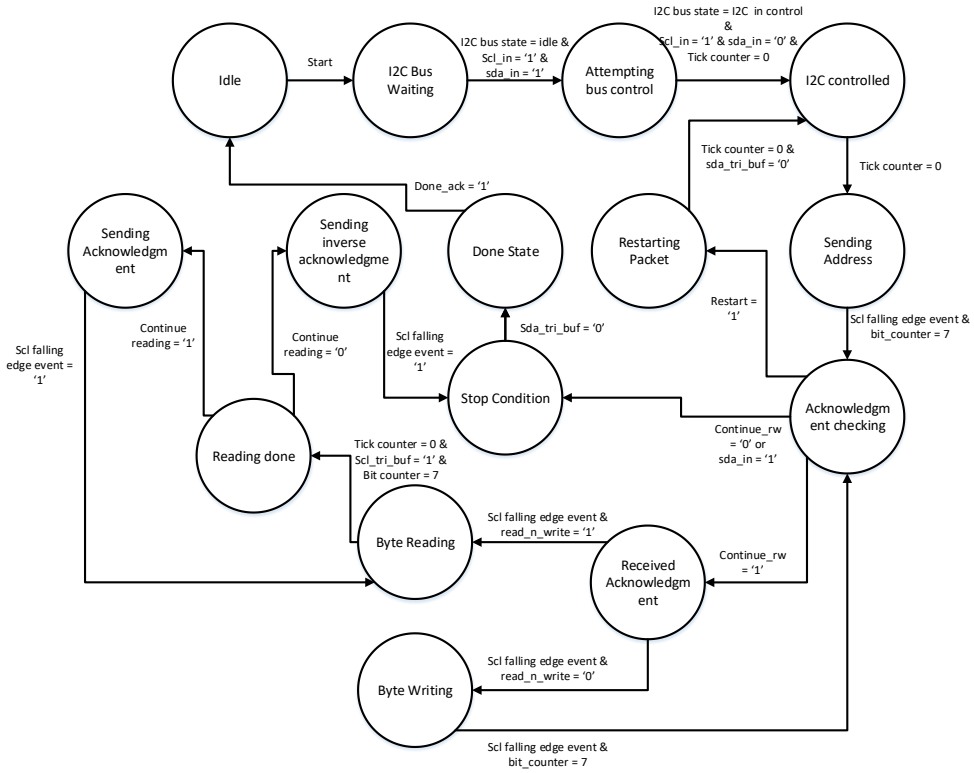



**Figure 5-19** Sensor Power Management: I2C Master Core State Machine.

## 5.10.3 Sensor Power Management State Machine

This state machine is running on a system clock. It is driven by Operational control component and it drives $I^2C$ Master Core state machine. This state machine provides a sequence of steps for powering the image sensoron and off. As per IO Expander, an address byte is provided first along with read/write bit. If read/write bit is high, port values are provided. If low, the provided data is given as output on port. As port has 16 channels, total packet length is 27 of which 7 bits are for address, 1 bit for read/write, 16 bits data and 3 acknowledgment bits.

Upon reception of a power on command, state is switched to “Power on Switches”. This state signals the core to transmit switch enable packet transmission to slave. Data is given to the core along with control signals throughout the duration of packet with read/ write bit kept to ‘0’. First switch bit is kept high and all others are kept low in data field. Once the data packet is transmitted, state is switched to “Reading Written Values” where all switch values are read with read/ write bit kept at ‘1’. Switch values are compared to written values. If data is matched, other switches are powered on one by one in “Power on Switches” state. The counter is incremented in between packet

transmission to provide adequate timings necessary by sensor. Previously powered on switch bits are kept high while powering subsequent switch. Once every switch is powered on and verified, the state is switched to “Powered On”. Similar procedure is used to power off image sensor where each switch is turned off one by one with proper timings ensured with counters. State machine is given in Figure 5-20.

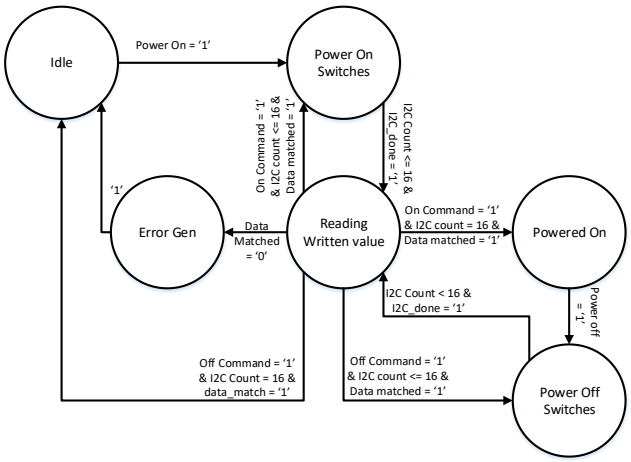



**Figure 5-20** Sensor Power Management: Main State Machine.

## 5.11 Health Monitoring

Health monitoring is done with the help of ADCs that are used in the SIC and sensor PCB. These are controlled by $I^2C$ communication protocol. Its implementation detail is similar to5.10. The only difference is that write packet is provided first which provides ADC with port number. Once sample is taken, value is stored in output register which is read.

## 5.12 Operational Control

This component provides complete control of the SIC firmware to the BIC. Every command and configuration parameters are provided to this component via the BIC Communication component discussed in section5.7. This component provides commands to respective components along with configuration parameters needed as seen in previous sections. This component also stores all configuration parameters in the form of registers. These registers are updated from BIC communication registers when a corresponding command is given. This component is responsible for control of operations highlighted in Figure 5-21 as separate states. This component also checks for

any error in operation. In case of an error, the corresponding override can be provided from the BIC on SPI, which will restore this state machine to the “Idle” state.

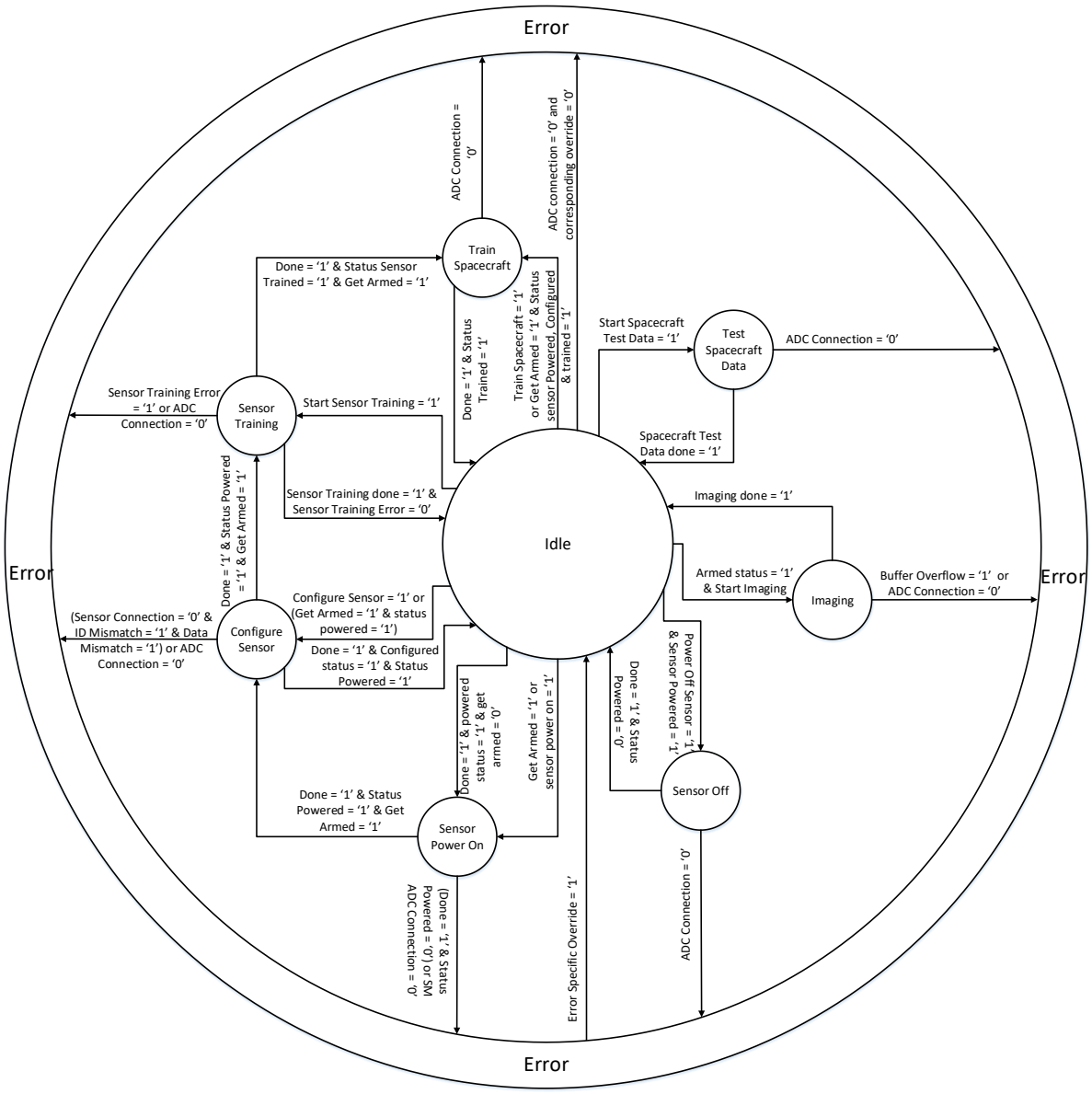



**Figure 5-21** Operational Control: State Machine.

# 6 FIRMWARE VERIFICATION

This chapter focuses on the verification of requirements provided in Chapter 3. This covers the behavioral simulation of SIC firmware as a verification mechanism. This chapter will close the loop of requirements. The strategy used fors requirement loop closing is similar to the strategy used for requirement finalization.

## 6.1 Sensor Handling

Data is provided to the SIC firmware from the test bench. Sensor data is simulated which includes generation of idle data followed by data start marker. After this, sensor data is provided. Data is generated at 40MHz. LVDS data channels and data clock provided input to firmware is given in **Figure 6-1**. It can be seen from the figure that data clock time period is 12.5ns which corresponds to 40MHz. It can also be observed that data is provided on both edges of this clock (DDR).

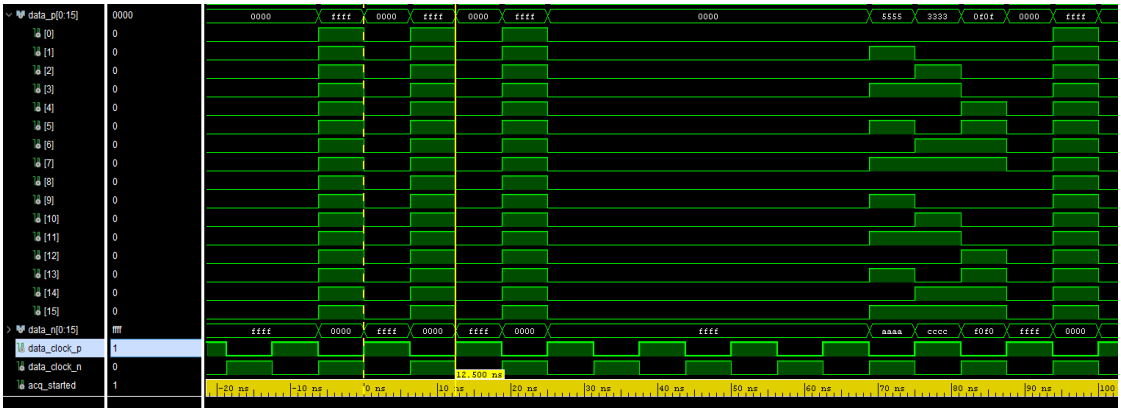



**Figure 6-1** Sensor Handling - Data Channels and Clock provided input.

Data generation is started by the reception of the initiate acquisition signal from the firmware. After the idle and data start marker, data is provided. Idle data and data start marker is not written in FIFO as these are only meant for sensor data reception component. Pixels are provided in the form of counters which correspond to their pixel number. This is shown in Figure 6-2.

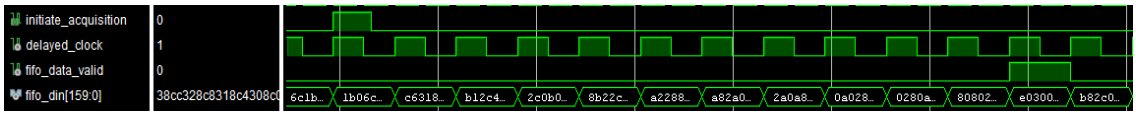



**Figure 6-2** Sensor Handling - Idle and data start marker.

Valid data is written in FIFO. As data is only valid after ten bit reception, only valid data is written in FIFO. This ensures that subsequent components only work on valid data. This is shown in Figure 6-3.

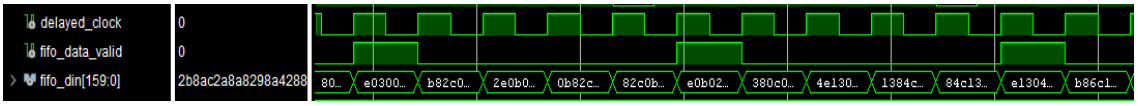



**Figure 6-3** Sensor Handling - Valid Data Writing in FIFO.

FIFO Reading is done in system clock and data is divided in the form of pixels. This pixel data is provided to next components. Output of sensor data reception component is given in Figure 6-4.

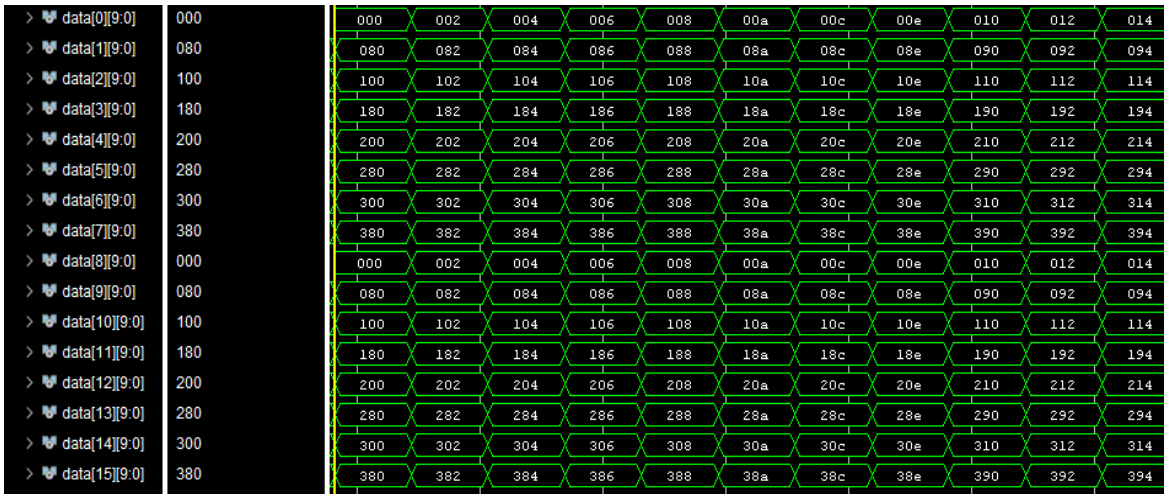



**Figure 6-4** Sensor Handling - Sensor Data Reception output for subsequent components.

Switch control to power on sensor was done using I2C. Communication is shown in Figure 6-5.

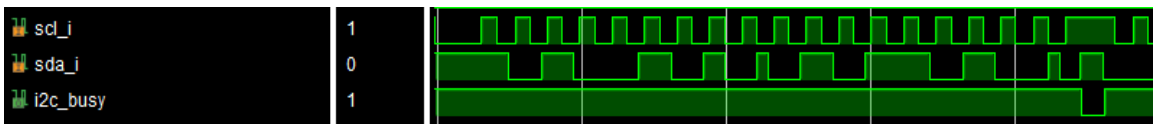



**Figure 6-5** Sensor Handling - Sensor Power On using I2C.

## 6.2 TMTC

TMTC is possible with SPI communication. As SIC firmware acts as a slave, it takes input CS_n, SPI Clock and MOSI. MISO is provided output. Two distinct communication packets are possible which are broadcast write and dedicated read/write. SCLK, MOSI and MISO are common for all SICs. Thus every SIC firmware is addressed by a unique address and CS. In broadcast write, all CS are kept low and data is written irrespective of the slave address provided. Every CS is kept active and every MISO is kept at high impedance at the start. Based on broadcast bit provided, MISO of all SICs are kept at high impedance to avoid driving same signal from multiple slaves. Simulation of a broadcast packet is shown in Figure 6-6.

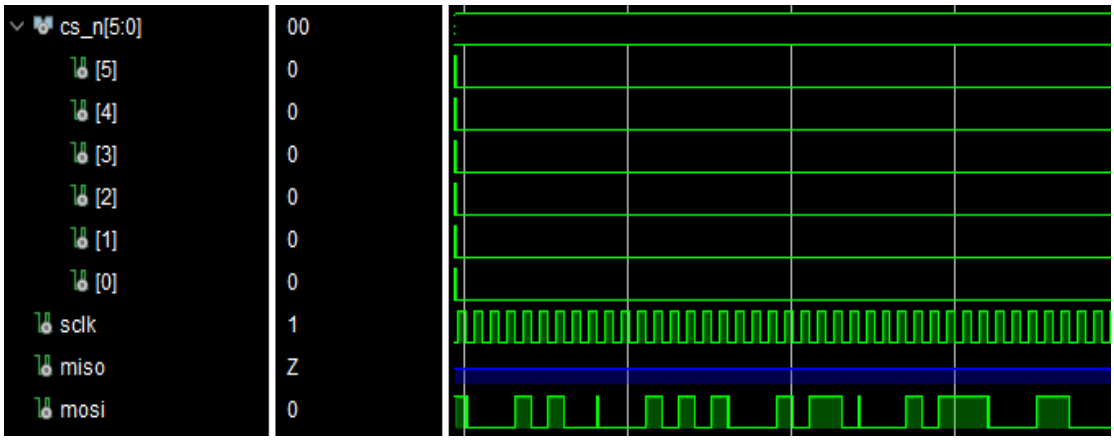



**Figure 6-6** TMTC - Broadcast Packet behavioral simulation.

As far as communication is concerned, read and write packets are similar. Read packet provides current register data on MISO and receive 0s in MOSI. Write data receives new register data on MOSI and provide current register data on MISO. The address of particular SIC firmware is provided along with corresponding active CS. Other CS are kept inactive. MISO will be kept at high impedance and after reception of valid address, it will be activated to provide data output. A simulation of read request is shown in Figure 6-7.

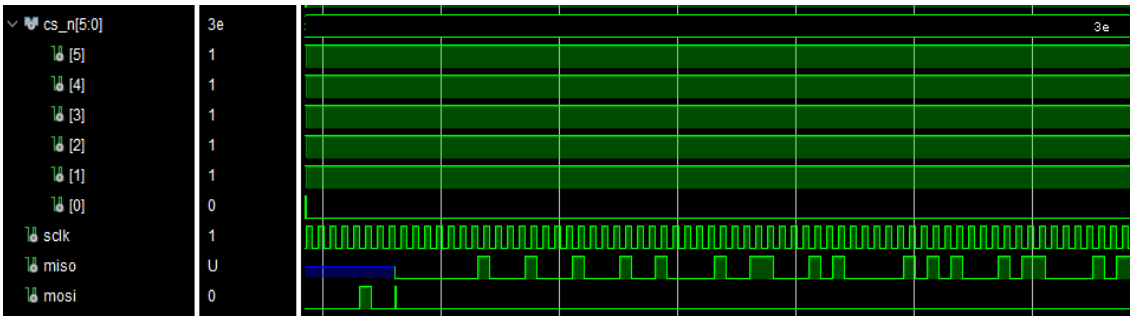



**Figure 6-7**TMTC - Read Request Behavioral Simulation.

## 6.3 Data Manipulation

After reception of Start Image Data transmission signal, Clock configuration provided by operational control signal is loaded into MMCM. Based on configuration value provided, MMCME2_ADV is configured. MMCM is in reset state during this process while the output locked signal is deactivated. Output clocks are also kept low during this process. Reconfiguration process is shown in Figure 6-8.

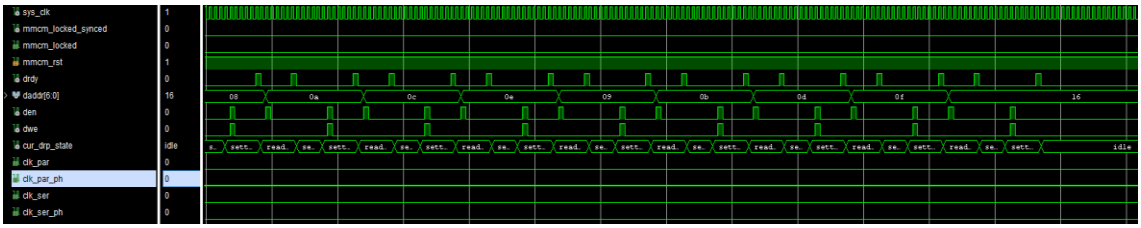


**Figure 6-8** Data Manipulation - Clock Reconfiguration process.

When MMCM is locked, image data transmission start signal is acknowledged. The start signal is received on rising edge of system clock and acknowledgment will also be given on the same clock. However, MMCM locked is an asynchronous signal. Thus this signal is synchronized with the system clock before decision can be made. This acknowledgment is shown in Figure 6-9.

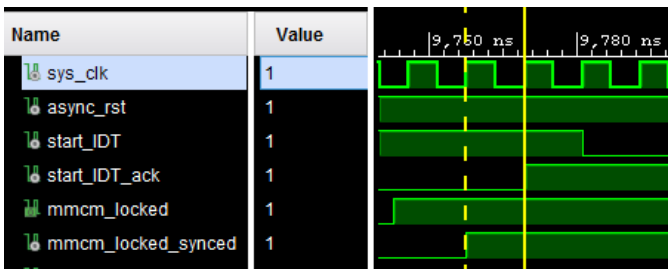



**Figure 6-9** Data Manipulation - Start Acknowledgment for Image Data Transmission.

OSERDESE2 reset needs to be generated on its parallel interfacing clock. As two types of OSERDESE2 are used one operating on “CLK_PAR” clock and other operating in “CLK_PAR_PH” clock, two types of resets will be used. These resets should be provided when input clocks are stable which are generated by MMCM. When the MMCM locked signal is provided, it needs to be synchronized with corresponding clocks. Once a synchronized signal is activated, reset is generated for the minimum period of five clock cycles of OSERDESE2 parallel interface. the reset generation process is highlighted in Figure 6-10.

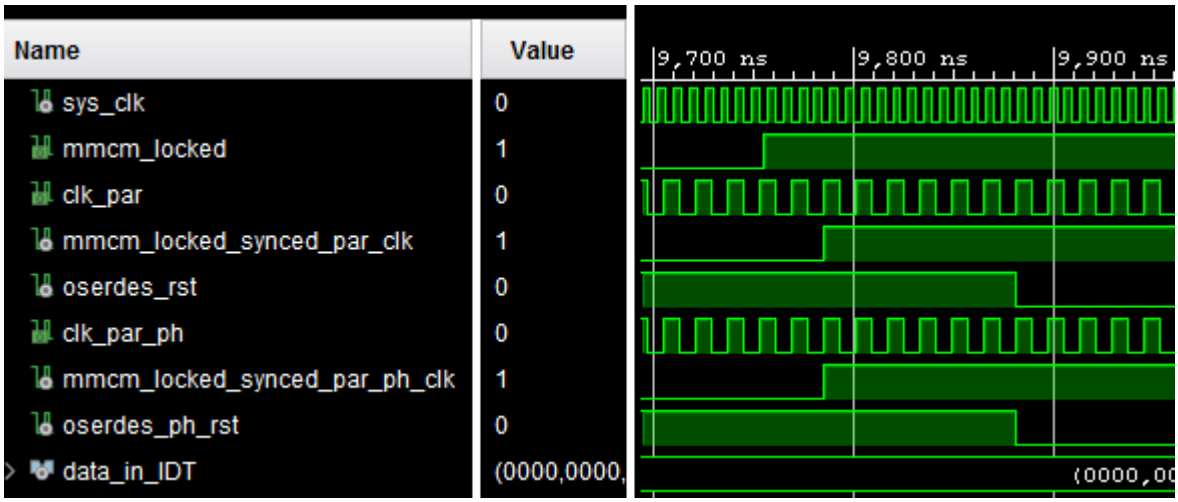



**Figure 6-10** Data Manipulation - OSERDESE2 Reset Generation Process Behavioral Simulation.

Data transmission is done after the reset process of OSERDESE2 is complete. Data clock is transitioned in the middle of data ensuring stability at clock edges. Thus only minor adjustments is needed at the receiver due to PCB and harnessing. This is highlighted in Figure 6-11.

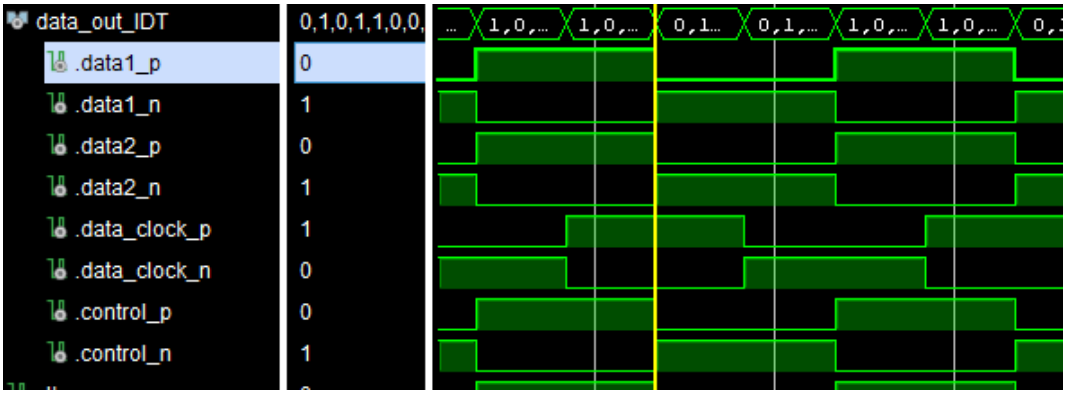



**Figure 6-11** Data Manipulation - Data Clock and Data Timing.

Fixed data is generated before actual data transmission and in between data transmission bursts. It is used by training algorithm of receiver. This fixed pattern is shown on the left side of Figure 6-12. After training data transmission on two data channels and control channel, auxiliary data is provided on both data channels. The higher byte is sent on data channel 2 and the lower byte is sent on data channel 1. The last bit of control channel shows the validity of data provided. Start of auxiliary data is also shown in Figure 6-12 when data is valid. Auxiliary data is shown in the form of counters.

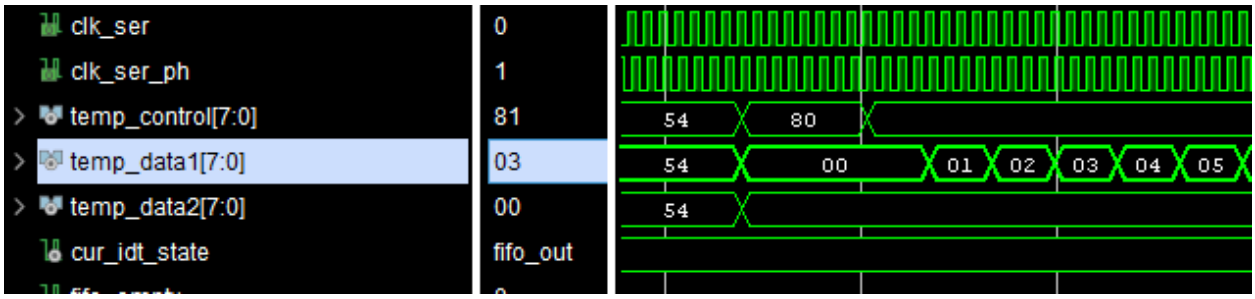



**Figure 6-12** Data Manipulation - Training Data Transmission before Auxiliary Data transmission.

Auxiliary data is 80 bytes long. Thus the last word of auxiliary data is 0x27. After auxiliary data transmission, image data is sent. Data is provided in a perfectly sequenced way as counter is provided as data to highlight pixel number. End of auxiliary data along with start of image data is shown in Figure 6-13.

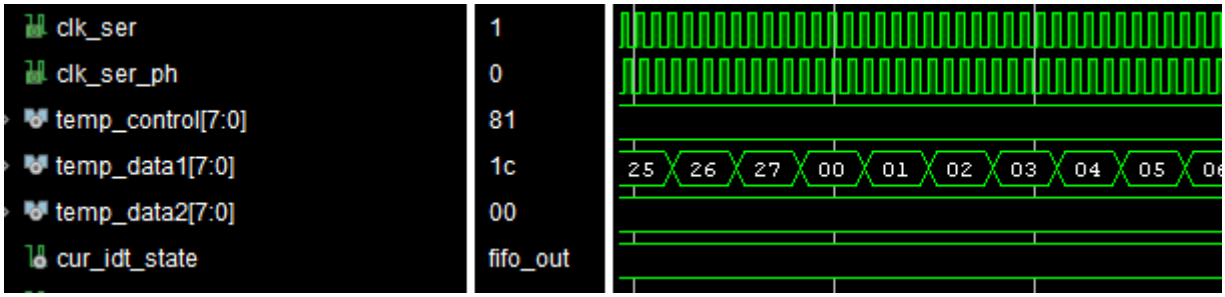



**Figure 6-13** Data Manipulation - End of Auxiliary Data and Start of Image Data.

PAN data is given at the output of SIC firmware as it is without any data change. The test bench provides PAN data in the form of incremental counter. Data at output is shown in Figure 6-14.

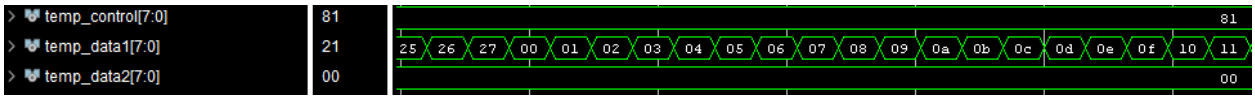



**Figure 6-14** Data Manipulation - PAN data.

2x2 binning is done in the firmware which is reflected in the output. As counter is given input, addition of four pixels will result in a pattern as shown in Figure 6-15.

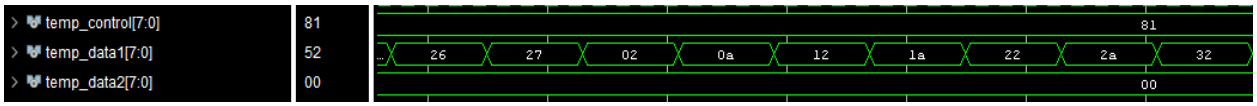



**Figure 6-15** Data Manipulation - Start of Binning Data after Start of Auxiliary Data.

Test Data is also generated to test the validity of link between SIC and spacecraft. Fixed data is generated which is bit reversal of counter data. A brief snapshot of the behavioral simulation for test data provided at output is shown in Figure 6-16.

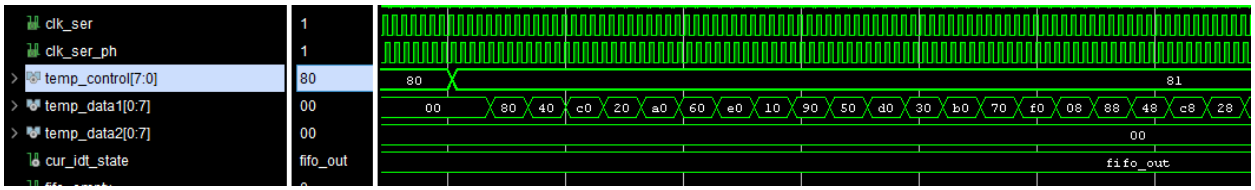



**Figure 6-16** Data Manipulation - Spacecraft Test Data Generation.

## 6.4 Results and Conclusion

The SIC firmware was verified behaviorally ensuring that it is providing all the functionalities highlighted in Chapter 4. A number of verification methods are possible, including demonstration, testing, analysis and inspection. As this work is based on simulation only, analysis was chosen as the key verification procedure. Every quantitative requirement was analyzedand the requirement loop was met based on the behavioral scope tool inbuilt in Vivado.

# 7 CONCLUSIONS AND FUTURE WORK

We have developed requirements for a firmware design for an FPGA used for data handling and processing on a small satellite. The firmware design of the FPGA is proposed, which drives firmware architecture. This architecture serves as a guideline for detailed firmware which was tested analytically as per requirements presented in this work.

An aspect noted is the diversity of the work involved in that effort has been done to optimize every item involved in the chain. The sensor, support electronics, quantum efficiency improvement, line rate enhancement, noise removal and SNR improvement are just few of many aspects considered.

The choice of processing chip is also very critical and,like sensors,the continuous improvement of the performance of FPGAs make such projects practicable. Such high speeds also need very fine PCB development and a number of works are seen which optimize this important aspect as seen in Literature Review. Firmware development is also not straight forward and number of algorithms were analyzed for their suitability for this particular application.

The literature review provides low level requirements which served as a starting point for this work. Based on these preliminary requirements, the space mission engineering (SME) process has been followed. As seen in Chapter 3, this process was the result of multiple projects done in the space industry, streamlining a number of processes which are crucial to make such complex projects realizable.Functional and requirements analysis, an important and often overlooked step in SME process, was done in Chapter 3which provides the result of multiple iterations. Although only the final output of these iterations is documented in Chapter 3, it doesn't negate the importance of all the efforts put into multiple iterations.Requirements were finalized based on this iterative process and functionalities were highlighted at the end of that chapter.

This work contains multiple aspects of the design, so two approaches would not suffice as alternate approaches. An alternate approach has been provided for every aspect of the design, as stressed in Chapter 2. Trade-off analysis has been done for these alternate approaches. Although only one aspect was considered at a time, multiple variables were still involved. Every such relation was discussed and the concept chosen in Chapter 4is

the one that is most in line with the requirements given in Chapter 3. This drives theBaseline architecture of firmware design given in Chapter 4. Components of the baseline architecture weregivenin that chapter, which allowed FPGA selection.Analyzing every aspect before jumping into development significantly reduces the development time.

State machine diagrams were discussed in Chapter 5. This translates the trade-off analysis and architecture of Chapter 4 into key roles, its activation strategies and permissible transitions. Keeping check of such considerations maintains the design and resultant firmware always in a defined state, therefore exciting the FPGA resources with defined stimuli. All other stimuli are rejected by firmware and this is only possible if proper care is given to states in design and development. This is an important step to make the system reliable.This also simplifies the design resulting in firmware simplification, which has a far reaching effect in terms of allowing the use of simpler parts and a lower cost solution.

Based on this design process, behavioral simulation was performed using many possible stimuli, including functional as well as erroneous inputs of interfaces. Such stimuli are defined in test benches. The timing relationship between such stimuli is also important as firmware may work smoothly for an individual scenario but may behave quite differently in the combination of two or more stimuli. Such scenarios and their results are presented in Chapter 6.

This work may initiate multiple future projects. Strictly speaking from a firmware point of view, two ways forward are proposed. The first includes the development of similar work for the BIC, which provides communication control thus that work can be developed on the microcontroller, but it can be finalized following a similar SME process.BIC firmware may seem simpler as evident from Chapter 1, but its reliability is very important as this is the only interface with the satellite (i.e.the user at ground). The SME process will guide the designer to effectively address all the risks involved and their timely resolution.

Another offshoot of this work is to carry on SIC firmware development. This can go on to finalization of place and route simulation. Timing analysis can be done to minimize the gap between simulation and actual hardware. After all these steps, FPGA implementation can be done. It could be then tested on aKintex 7 evaluation board or

could be done in SIC PCB subjected to PCB development.This will make the design practically realizable.

# REFERENCES

[1] Vipin, K. and Fahmy, S., "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications," in *ACM Computing Surveys*, Warwick, UK, 2018.

[2] Contreras, M., "Design of an FPGA-Based Smart Camera and its Application towards Object Tracking," Masters of Engineering Thesis, Massey University, Manawatu, New Zealand, 2016.

[3] Olivieri, S., Aarestad, J., Pollard, L., Wyglinski, A., Kief, C. and R. Scott Erwin, "Modular FPGA-Based Software Defined Radio for CubeSats," in *IEEE International Conference on Communications (ICC)*, Ottawa, ON, Canada, 2012.

[4] El-Rayis, A. and Melnyk, A., "Localized Payload Management Approach to Payload Control and Data," in *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Edinburgh, UK, 2007.

[5] Yong, S. and Ra, S., "The Design of MSC (Multi-Spectral Camera) System Operation," in *International Geoscience and Remote Sensing Symposium*, Anchorage, AK, USA, 2004.

[6] Hofmann, A., Wansch, R., Glein, R. and Kollmannthaler, B., "An FPGA based On-Board Processor Platform for space applications," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Erlangen, Germany, 2012.

[7] Duan, Y., Wen, D., Gao, W. and Xian, B., "Two Approaches to Improve FPGA Performance for the Stereo Camera of the Chang'E-1 Satellite," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Erlangen, Germany, 2012.

[8] Gaikwad, A., "Eye Diagram Assessment Platform for Fiber-Optic Communications," Master's Thesis, Rochester Institute of Technology, Henrietta, New York, United States, 2018.

[9] Texas Instruments, "ADC12DJ3200QML-SP data sheet," 2018. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?genericPartNumber=adc12dj3200qml-sp&fileType=pdf. [Accessed 03 April 2019].

[10] AMS, "CMV12000 datasheet-v2.14," February 2018. [Online]. Available: https://ams.com/documents/20143/36005/CMV12000_DS000439_3-00.pdf/f6217e40-f30f-7e28-6d89-1245405bcf87. [Accessed 03 April 2019].

[11] Baker, E., "The Design of a CMOS Sensor Camera System for a Nanosatellite," Master's Thesis, Departmentof Electrical and Electronic Engineering, University of Stellenbosch, Stellenbosch, South Africa, 2006.

[12] Hu, T. and Chen, Y., "Design of the Linear Array CCD Acquisition System that Line Frequency and Integration Time Adjustable," International Conference on Electronics and Optoelectronics, Dalian, Dalian, 2011.

[13] Srinivasan, R., Anupama, K. , Suneeta, Saha, S. and Rao, A., "FPGA Based ASM implementation for CCD Camera Controller," International Conference on Emerging Trends in Electronic and Photonic Devices & Systems, Varanasi, India, 2010.

[14] Hoshino, K. and Nishimura, T., "A Model and Verification for Temperature Dependency of the noise in large size CCD image sensor," in *32nd Annual Conference on IEEE Industrial Electronics*, Paris, France, 2007.

[15] Zhou, J., Chen, X., Zhou, W. and Shen, W., "Design and implementation of timing

generator of frame transfer area array ccd sensor," *Proc. of SPIE,* vol. 6833, 2007.

[16] Shah, P., Soni, B, Waris, M., Kumaran, R., Mehta, S. and Chowdhury, A., "Generic and Programmable Timing Generator for CCD Detectors," in *International Conference on Advances in Computing,,* New Delhi, India, 2014.

[17] Jai.com, "PULNIX Imaging Products, TM-75/76 High-Resolution CCD Camera, Revision A.," PULNIX, 27 June 2001. [Online]. Available: https://www.jai.com/uploads/documents/Discontinued-Products/English-Manuals-Datasheets/TMSeries/Manual_TM-75_76_Discontinued.pdf. [Accessed 15 May 2019].

[18] Teledyne Dalsa, "IT-K3-04120 Datasheet, Single Line Mono CMOS Imaging Sensor," 23 May 2018. [Online]. Available: www.teledynedalsa.com. [Accessed 15 May 2019].

[19] Zhiyong, L., Weihua, Y. and Xiance, D., "The analog front end of ultra-high resolution CCD design based on AD9920A," in *8th International Conference on Intelligent Computation Technology and Automation,* Nanchang, China, 2016.

[20] Si, G., Li and Y., Guo, Y., "Timing Generator of Scientific Grade CCD Camera and Its Implementation Based on FPGA Technology," *Proceedings of SPIE,* vol. 7658, 2010.

[21] Clancy, S., Shihabi, M and Angkasa, K., "Using SpaceWire Time Codes for Spacecraft Time synchronization," in *International SpaceWire Conference (SpaceWire),,* Yokohama, 2015.

[22] Cinar, E., Turhan, O., "Command and Data Handling Subsystem of GÖKTÜRK-2 Flight Model," in *7th International Conference on Recent Advances in Space Technologies (RAST),* Istanbul, 2015.

[23] Angadi, C., Manjiyani, Z., Dixit, C, Vigneswaran K, Avinash G., Narendra, P., Prasad, S. and Ramavaram, H., "STUDSAT: India's First Student Pico-Satellite Project," in *IEEE*, 2011.

[24] Mora, J., "Payload Data Handling, Telemetry and Data Compression system for GAIA," UNIVERSITAT POLITÈCNICA DE CATALUNYA, Barcelona,, 2005.

[25] Cahyono, D. and Nugroho, A., "Design and Realization of Camera Controller for a Remote Sensing Payload of Nanosatellite FPGA (Field Programmable Gate Array)," in *6th International Conference on Recent Advances in Space Technologies (RAST)*, Istanbul, 2013.

[26] Gulzar, K., "Camera Design for Pico and Nano Satellite Application," Lulea University of Technology, Lulea, Sweden, 2009.

[27] "COBAN: Design of Multi-Spectral Opto-Electronic Satellite Camera System," in *IEEE*, 2007.

[28] Wertz, J., Everett and D., Puschell, J., Space Mission Engineering: The New SMAD, Hawthorne, CA: Microcosm Press, 2011.

[29] Xilinx, "UltraFast Design Methodology Guide for the Vivado Design Suite UG949 (v2018.2).," 27 July 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug949-vivado-design-methodology.pdf. [Accessed 19 May 2019].

[30] Akin, D.L., "Akin's Laws of Spacecraft Design," [Online]. Available: https://spacecraft.ssl.umd.edu/akins_laws.html. [Accessed 11 August 2019].

[31] NASA, "NASA Systems Engineering Handbook, NASA/SP-2007-6105," [Online]. Available:

https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080008301.pdf. [Accessed 11 August 2019].

[32] Altera White Paper Version 1.0, “Selecting the Ideal FPGA Vendor for Military Programs,” Februrary 2009. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01094-select-military-vendor.pdf. [Accessed 23 May 2019].

[33] Digikey, “FPGA prices,” [Online]. Available: https://www.digikey.com/product-detail/en/xilinx-inc/XC7K70T-1FBG676I/122-2075-ND/3641767. [Accessed 08 September 2019].

[34] Xilinx Product Specification, “7 Series FPGAs Data Sheet: Overview, DS180 (v2.6),” [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. [Accessed 08 September 2019].

[35] Xilinx Product Specification Kintex 7, “Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics, DS182 (v2.18),” [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds182_Kintex_7_Data_Sheet.pdf. [Accessed 08 September 2019].

[36] Xilinx User Guide, “7 Series FPGAs SelectIO Resources, UG471 (v1.10),” [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf. [Accessed 08 September 2019].

[37] Xilinx Vivado User Guide, “Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide, UG953 (v2016.4),” [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug953-vivado-7series-libraries.pdf. [Accessed 30 September 2019].

[38] Xilinx Clocking User Guide, “7 Series FPGAs Clocking Resources, UG472 (v1.14),” [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf. [Accessed 30 September 2019].

[39] Vivado Design Suite, “FIFO Generator v13.2 LogiCORE IP Product Guide, PG057 (v13.2),” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_2/pg057-fifo-generator.pdf. [Accessed 15 September 2019].

[40] Xilinx Product Brief IDELAYCTRL, “LogiCORE IP Utility IDELAYCTRL Logic, PB044 (1.0),” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/util_idelay_ctrl/v1_0/pb044-util-idelay-ctrl.pdf. [Accessed 08 September 2019].

[41] Vivado Design Suite, “Block Memory Generator v8.4 LogiCORE IP Product Guide, PG058 (v8.4),” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf. [Accessed 21 September 2019].

[42] Xilinx Application Note, “MMCM and PLL Dynamic Reconfiguration, XAPP888 (v1.8),” [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp888_7Series_DynamicRecon.pdf. [Accessed 22 September 2019].

[43] Texas Instruments, “PCF8575 Remote16-BIT I2C AND SMBus I/O Expander with Interrupt Output,” [Online]. Available: http://www.ti.com/lit/ds/symlink/pcf8575.pdf. [Accessed 2019 September 2019].

[44] Valdez, J., and Becker, J., “Texas Instruments, SLVA704 Application Report,” [Online]. Available: http://www.ti.com/lit/an/slva704/slva704.pdf. [Accessed 22

September 2019].