# Design and Implementation of a Test Automation System to Test the Micro Star Tracker Video Processing Unit

Akshay C
1510110035
ac140@snu.edu.in

Shiv Nadar University
NH91, Tehsil Dadri
Gautam Buddha Nagar
Uttar Pradesh - 201 314

Project Duration: 10th July 2017 to 9th August 2017

Under the guidance of:

Mr. K. R. Muralidhara
Division Head, CSSD, CSSG


Laboratory for Electro-Optics Systems
Indian Space Research Organisation
Bangalore - 560 058

**Abstract**

This project was carried out to design and implement a test automation system to test and evaluate the Video Processing Unit of the Micro Star Tracker. These tests act as a fast way to test the behaviour of the Video Processing Unit and to see if it is acquiring the images correctly in each of it's modes. The acquired images are received by a console, and then they are extensively manipulated and tested using techniques like template matching and kernel convolution. A report is generated after all testing which lists the results of each test. To achieve this four software modules were developed and implemented. 1– APU Test Module, 2– Console Module, 3– Processing and Analysis Module, 4– Report Generation Module

# Contents

# List of Figures

# List of Tables

# Acknowledgements

I am grateful to Shri. Laxmiprasad A. S., Director LEOS-ISRO for providing this wonderful opportunity.

I thank my project guide Shri. K. R. Muralidhara, Divn. Head, CSSD, CSSG for his wonderful encouragement and guidance.

I thank my mentor Smt. Suja P. V., Sci/Engr. 'SE' CSSD for her constant support and guidance.

I am grateful to Shri. H. Ganesha Shanbhogue, GD PPEG/Admin and Shri. V. V. Ramana Reddy, PPEG for their encouragement.

I sincerely thank Dr. Anuradha Ravi, Computer Science Dept. Shiv Nadar University for her immense encouragement and support.

I thank Smt. Roopa M. V., Sci/Engr. 'SF' Spacecraft Operations Area, ISTRAC-ISRO for her constant encouragement.

Lastly I'd like to thank my Parents who have continuously supported me.

# Chapter 1

# Introduction

## 1.1 Attitude Computation

Once a satellite is placed in orbit, it's orientation or attitude needs to be known. Without accurate attitude data the satellite may not be pointing in the right direction. Without the right orientation the satellite may lose power as the solar panels may be facing in the wrong direction, or the satellite's antennas and cameras may be facing the wrong direction. Therefore one may easily see that knowing the attitude of a satellite is of prime importance. In space, a satellite is in constant motion with respect to the celestial body it is orbiting, and that celestial body itself will be in motion with respect to some other body. And thus establishing a proper frame of reference is not easy. Considering all the constantly changing factors computing a satellite's orientation becomes a non trivial problem.

There are many methods by which a satellite's attitude may be computed, data from the Gyroscopes and Accelerometers may be extrapolated and used, but perhaps the best and most robust method is by imaging the stars and computing the attitude based on their relative positions.

To achieve this a device called the Star Tracker(Star Sensor) is used. In it's core it is a digital camera attached to a computer. This device continuously captures images of the stars and computes the satellite attitude relative to their positions.

## 1.2 The Need for Test Automation

During the development phase of the Star Tracker, it's behaviour needs to be tested several times continuously. It's ability to acquire images needs to be evaluated. Doing so manually everytime is time consuming and tedious. Thus there has to be one shot solution that acquires various images in several modes of operation and tests them.

# Chapter 2

# Micro Star Tracker Implementation Details

The Micro Star Tracker has been implemented as a single SOC Computer with one Camera Unit, thus making it a Star Tracker in the Single Head Configuration

The contents of this chapted have been taken from the LEOS Technical Documents, refer the Bibliography for further details.

SPARC V8 FT is a Fault Tolerant soft VHDL IP core Processor system (System-On-Chip), based on AMBA (Advance Micro-controller Bus Architecture) Bus. The processor is fully compliant to SPARC V8 specification with SPARC V8 Integer Unit and a Floating Point Unit as its co-processor. A development was initiated in Star Sensor Group, LEOS to develop a Fault Tolerant 32-Bit SPARC V8 based Soft IP core system for Sensor systems that is fully tolerant to Single Event Upsets (SEUs) for usage in Low Earth Orbit, GEO Stationary and GEO Synchronous operational and scientific programmes. The advantage of a Soft IP core system is customization of the processor for specific applications by integrating the application specific IP core modules with the processor. The cores that are only neccessary and required for a particular design needs to be configured and thus making the system compact with no overheads. The Soft IP core processor systems are implemented as System On Chip (SOC) designs, either in FPGAs or ASICs. The other devices on a system board are only interface tranceivers, high density memories, power supplies and clock generators.

## 2.1 System Overview

Micro Star tracker is a single-head configuration star sensor. Here Video Processor Unit (VPU ) and Attitude Processing Unit (APU) logics are integrated in single FPGA. The Camera Head Unit ( CHU) is not separated from APU. The system block diagram is shown in the figure below.

Figure 2.1 shows the System Block Diagram.

The function of Camera Head ( CH ) is to acquire the Star Images from the sky and provide the data to the APU for attitude computation. To acquire the star images, each CH consists of a 1k x1k APS HAS2 as image detector.

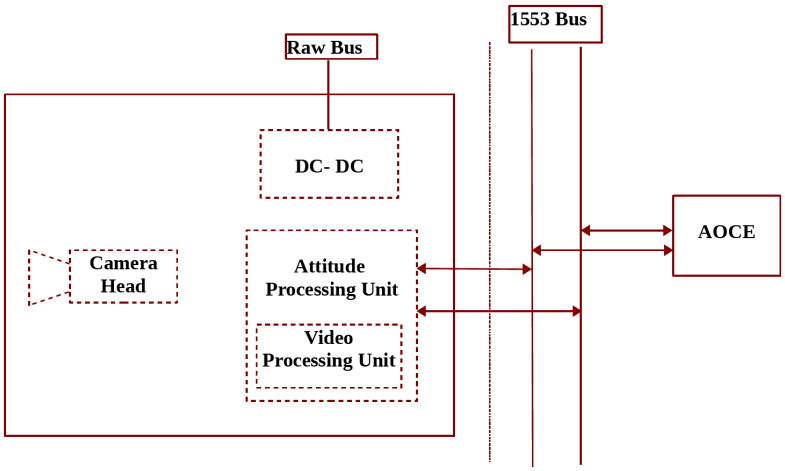


Figure 2.1: System Block Diagram

To readout the data acquired by the APS, a sequence of operations are to be executed with device specific implementation definitions.

The sequence of operations is dictated by the APU to the VPU, which is configured as an module inside APU, based on APS and sensor operation requirements. The VPU consists of a customized sequence generator which generates timing signals in the format demanded by the device. The block diagram is as shown in Figure.

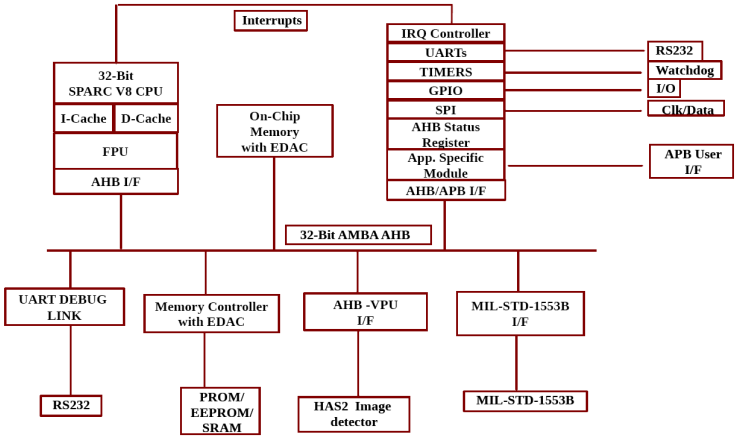


Figure 2.2: APU Block Diagram

Figure 2.2 shows the Block Diagram of the APU.

## 2.2 Operation

The operation of the Micro Star Tracker is as follows. The APU is SPARC–V8 Processor which is implemented with AMBA bus architecture. The VPU is configured as an APB slave for loading CH parameters and is also configured as AHB master for storing data acquired from CH to secondary memory through DMA. The VPU is configured through APB registers and execution starts in the programmed mode of operation when pulse command bits are set. The Sequencer module in VP generates required clocks sequentially to acquire star image data and health data. For this, it generates pixel readout clocks, and sampling and control signals to the ADC to obtain the digitized data of the APS image. This acquired data is collected in an on-chip FIFO and stored in RAM available on the board for further processing. The FIFO is 32 bit wide and have a depth of 4K, and is EDAC protected with an additional 8 bits.

### 2.2.1 APU Architecture

Fault tolerance to SEUs is achieved through

- Triple Modular Redundant (TMR) registers for registers
- Error Detection And Correction (EDAC) logic in memories for correcting single/double errors and detecting and flagging multiple errors
- Synchronous design for transients mitigtion of combinational circuits

The Basic non-FT version of SPARC V8 available in open source domain is inherited for the development with the configuration

- SPARC V8 Integer Unit
- Icache & Dcache Controller
- AMBA AHB Controller
- AMBA APB Controller
- Interrupt Controller
- General Purpose & WatchDog Timer

General Purpose Input Output (GPIO)

- Serial Periphral Interconnect (SPI)
- UART

The Basic non-FT version is converted to FT version

- By implementing the design in RTAX 2000 FPGA for TMR of registers
- EDAC for Processor Register File (136, 32-Bit registers) memory
- Parity for I-Cache, D-cache memories
- EDAC for external memory

New IP core modules have been developed and integrated to the Basic version.

- Floating Point Controller
- Floating Point Unit
- FT Memory controller
- SpaceWire link
- AMBA SpaceWire DMA
- 1553RT core from Actel
- AMBA 1553 DMA

Application specific Auxillary Module

### 2.2.2 Memory Map

The memory map is given in the Table below for the standard AHB slaves used in the micro-controller. SPARC V8 Proceccor Memory map

| S.No. | Core | Address Range | Area |
|---|---|---|---|
| 1 | FTSRCTRL | 0x00000000- 0x20000000 | PROM area |
| | - | 0x20000000- 0x40000000 | I/O area |
| | - | 0x40000000- 0x60000000 | SRAM area |
| 2 | APBCTRL | 0x80000000- 0x81000000 | APB Bridge |
| 3 | AHB plug & play | 0xFFFFF000- 0xFFFFFFFF | Registers |

Table 2.1: Memory Map

## 2.3 MIL-STD 1553B Remote Terminal

MIL-STD -1553B allows communication between Bus Controller (BC) and Remote Terminal (RT). It is a serial data bus based on message transmission where each message format is made of control words called command and status.

-The BC's main function is to provide data flow control for all transmissions on the bus. It is the sole source of communication. The system uses a command/response method. All messages are initiated by the BC using command word.

- The embedded RT consists of interface circuitry located inside a sensor or subsystem directly connected to the data bus. Its primary job is to perform the transfer of data in and out of the subsystem as controlled by the BC. The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core 1553BRT core. At a high level, the interface simply provides a set of memory mapped sub-addresses that 'receive data written to' or 'transmit data read from'. The interface requires 2,048 16-bit words of memory, which can be shared with a local processor. The interface consists of six main blocks: 1553B Encoders, decoders, backend interface, command decoder, RT controller blocks and a command legalization block as shown in the figure below.

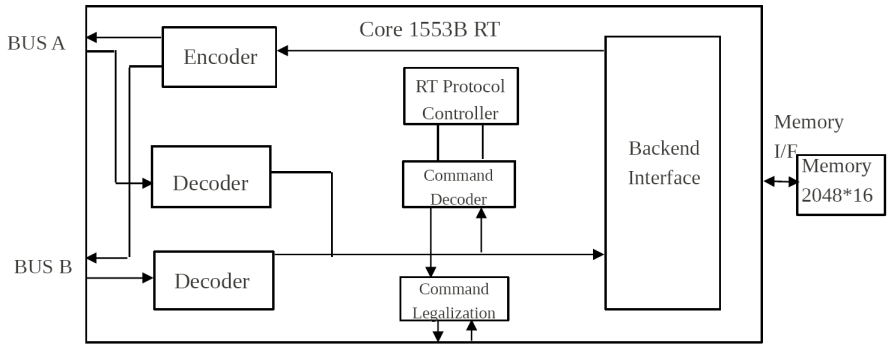


Figure 2.3: MIL-STD-1553B Block Diagram

## 2.4 AHBVPU- AMBA AHB Video Processing Unit

Overview AHB Video Processing Unit (AHBVPU) IP Core is a part of Attitude Processing Unit of Micro Star Tracker. It is configured by SPARC-V8 processor through AMBA APB bus.

The parameters and commands for the Star Image capture operation are written to APB registers by software. AHBVPU processes the commands and generates HAS2 (High Accurace Active Pixel Sensor Version 2) signal sequences for reading the HAS2 Image, HK_parameters, and stores the same in RAM via DMA interface. AHBVPU is an AHB master with second highest bus grant priority.

### 2.4.1 Operation

The AHBVPU module comprises a set of registers interfaced through APB interface. The bits in control registers are either to set or reset to operate Video Processing Unit in different modes. The AHBVPU module also has a state machine to control DMA operations. It requests for bus ownership when there are more than 32 words in the FIFO or when one frame acquisition is over. The bus arbitration priority for AHBVPU is second to that of 1553 Remote terminal. The end of acquisition is indicated to software by setting a bit.

### 2.4.2 AHBVPU Address Space

The following table gives the VPU address space. The location of each of the configuration registers, control registers and the Status registers can be found in Table 2.2.

### 2.4.3 AHBVPU Registers' Configuration

1. AHBVPU Control register

   [31:3] Reserved

   [2] GET_FRAMES: Generates a frame run pulse of one clock cycle. This signal triggers image capture operations.

   [1] GET_HK: Generates a get_hk pulse of one clock cycle. This signal triggers the HK parameter reading through ADC in the APS chip.

   [0] RESET : Generates a reset pulse of one clock cycle.

2. APS parameters register1 [31:24] WM: Set read out mode,Window size and time code rate.

   [26] – '0' Full frame mode

   [25:24] = 00,11,01 - Full frame image mode(Acq) 10 - Full frame sobel mode (Acq)

   [26] – '1' Window mode (Track)

   [25:24] – 00 - 10 x 10

   01 - 20 x 20

| S.No. | APB Address Offset | Register |
|---|---|---|
| 1 | 0x00000000 | AHBVPU control Register |
| 2 | 0x00000004 | APS Parameters Register1 |
| 3 | 0x00000008 | APS Parameters Register2 |
| 4 | 0x0000000C | Full frame coordinate register1 |
| 5 | 0x00000010 | Full frame coordinate register2 |
| 6 | 0x00000014 | Threshold and exposure register |
| 7 | 0x00000018 | Window coordinate register1 |
| 8 | 0x0000001C | Window coordinate register2 |
| 9 | 0x00000020 | Window coordinate register3 |
| 10 | 0x00000024 | Window coordinate register4 |
| 11 | 0x00000028 | Window coordinate register5 |
| 12 | 0x0000002C | Window coordinate register6 |
| 13 | 0x00000030 | Window coordinate register7 |
| 14 | 0x00000034 | Window coordinate register8 |
| 15 | 0x00000038 | Window coordinate register9 |
| 16 | 0x0000003C | Window coordinate register10 |
| 17 | 0x00000040 | Window coordinate register11 |
| 18 | 0x00000044 | Window coordinate register12 |
| 19 | 0x00000048 | Window coordinate register13 |
| 20 | 0x0000004C | Window coordinate register14 |
| 21 | 0x00000050 | Window coordinate register15 |
| 22 | 0x00000054 | Window coordinate register16 |
| 23 | 0x00000058 | Window coordinate register17 |
| 24 | 0x0000005C | Window coordinate register18 |
| 25 | 0x00000060 | Window coordinate register19 |
| 26 | 0x00000064 | Window coordinate register20 |
| 27 | 0x00000068 | DMA start address register |
| 28 | 0x0000006C | DMA end address register |
| 29 | 0x00000070 | Status Register |

Table 2.2: AHBVPU Address Space

10 - 30 x 30

11 - 40 x 40

[27] = '0' – Disable HK_data acquisition

'1' – Enable HK_data acquisition

[30] = '0' – Time code rate 128 us

'1' – Time code rate 256 us

[23:16] OFFSET: HAS2 column amplifier's offset correction

[15:8] BLACK: NDR mode black level

[7:0] MODE:

[1]: '0' - DR mode, '1' - NDR mode

3. APS parameters register2

   [31:24] Reserved

[23:16] WM_GO :Window mode Gain and Offset

[15:8] TEST: Window mode read data type

[9:8] = 00/11 : CDS level

01 : Reset level

10 : Read level

[7:0] WINDOW COUNT: Number of windows (stars identified in acquisition mode) to be read in window mode of operation.

4. Full frame coordinate Register1

   [31:16] FF_Y1: Full frame image line start address.

   [15:0] FF_X1: Full frame image pixel start address.

5. Full frame coordinate Register2

   [31:16] FF_YS: Full frame image line end address.

   [15:0] FF_XS: Full frame image pixel end address.

6. Threshold and Exposure Register

   [31:16] Threshold: Background intensity threshold for Sobel mode

   [15:0] Exposure: Full frame and window mode integration time.

7-26 Window coordinate Register1 to Register20

   [31:24] Reserved

   [23:12] WM_Y1: Window mode image line start address.

   [11:0] WM_X1: Window mode image pixel start address.

27 DMA Start Address Register

   [31:0] DMA START ADDRESS: Memory address from which DMA interface of AHBVPU starts writing Star image data.

28 DMA End Address Register

   [31:0] DMA END ADDRESS: Last memory address t which DMA interface of AHBVPU has written data.

29 Status Register

   [31:1] Reserved

   [0] End of Acquisition: Indicates end of window or full frame image acquisition.

## 2.5 HAS2 Structure and Readout Operations

HAS2 APS sensor is an advanced version or version2 of CMOS Active Pixel Sensors developed for ESA. HAS2 similar to STAR1000 is a monolithic integrated circuit comprising 1024 x 1024 pixels array, X and Y addressing logic, column amplifiers, on-chip FPN correction with double sampling and destructive read out, Programmable gain amplifier, 6-channel multiplexer and 12-bit internal ADC.The pixel array is square array that contains 15 um x 15 um pixels. Each pixel comprises of a photodiode and three transistors.

### 2.5.1 Line Addressing

The sensor operates line wise. A line of pixels can be selected and reset, and a line of pixels can be selected for readout into the column amplifiers. There is no frame reset or frame transfer operation. Image acquisition is done by sequencing over all lines of interest and applying the required reset and or readout control to each line selected. The sensor array contains two vertical shift registers for line addressing. These registers are one-hot at each time pointing to one line of pixels. The figure 2.4 shows the Line addressing structures.

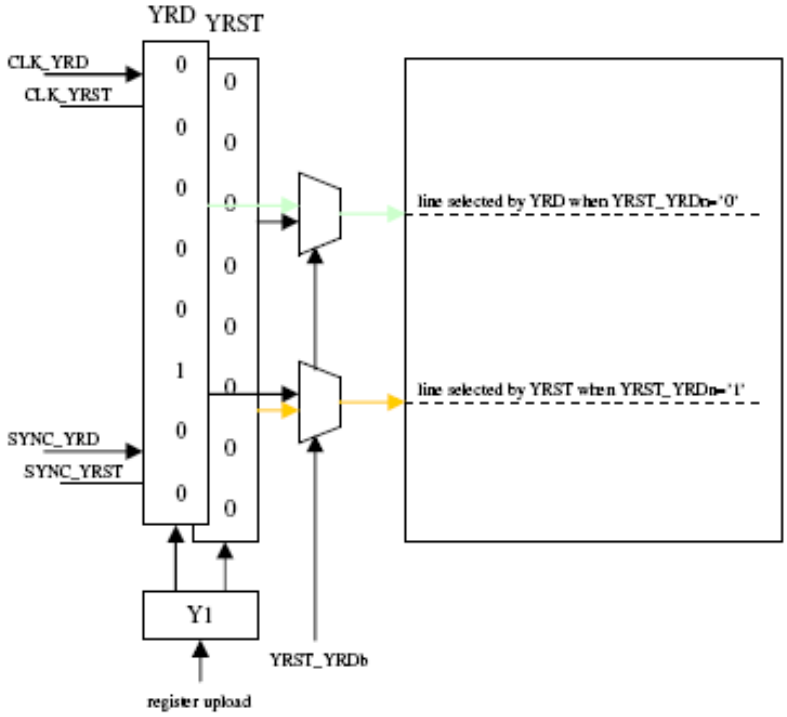


Figure 2.4: HAS2 Sensor: Line Addressing

### 2.5.2 Analog to Digital Converter

The on-chip ADC is a 12-bit pipelined one. It has a latency of 6.5 pixel clock cycles. If the signal is sampled on the first rising edge of the clock, the converted output is available 6 clock periods afterwards on the falling edge. The ADC contains it own SPI serial interface for the optional upload of calibration settings, enhancing its performance. The ADC is electrically isolated from sensor core and need not be powered when not used. The input range of ADC is set with between the voltages V_LOW_ADC and V_HIGH_ADC.

### 2.5.3 Readout Mechanisms

The HAS2 sensor has two readout mechanisms Double Sampling – Destructive Readout (DS-DR) and Correlated Double Sampling – Non Destructive Readout (CDS-NDR).

In DS-DR mode one of these registers is typically dedicated to addressing the lines to read and the other is used for addressing the lines to reset as a part of electronic rolling shutter mode of operation. In CDS-NDR mode, either one or both can be used. Both the Y-registers can be initialised to a position indicated by the on-chip address register by programming the register through parallel sensor programming interface and the registers can be advanced one position through user control.

### 2.5.4 Double Sampling – Destructive Readout

In Double Sampling – Destructive (DS-DR) mode, the YRST pointer runs over the frame from top to bottom, each time resetting the line it addresses. Lagging behind this runs the YRD pointer, each time reading out the line it addresses. The distance between the YRST pointer and YRD pointer is then proportional to the exposure time. This operation is electronic rolling shutter. At line readout the signal levels of the pixels in the addressed line are copied onto the column amplifier's signal sample nodes. After this the line of pixels is reset and the pixel's black levels are copied onto the column amplifier's reset sample nodes. The readout is destructive. The column amplifiers / PGA then subtract the black levels from the signal levels during sequential pixel out. The sampling is uncorrelated eliminating any static pixel to pixel offsets of the sensor array. Figure 2.5 shows the double sampling process.

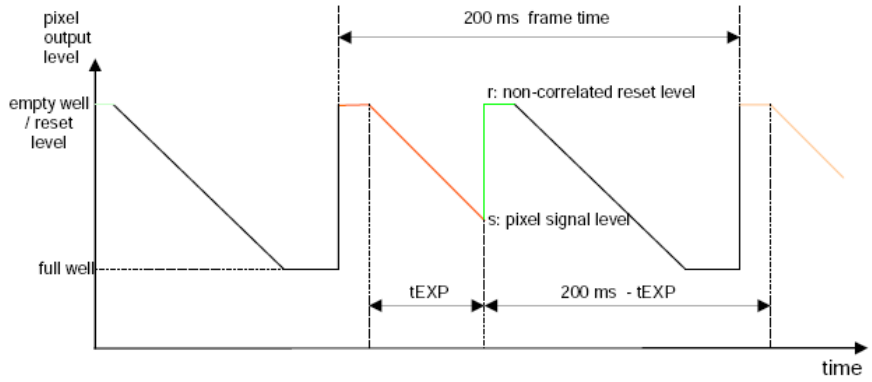


Figure 2.5: DS-DR Readout Process

### 2.5.5 Correlated Double Sampling –Non- Destructive Readout

In Correlated Double Sampling – Non Destructive Readout (CDS/NDR) mode the YRST or YRD pointer quickly runs over the frame top to bottom resetting

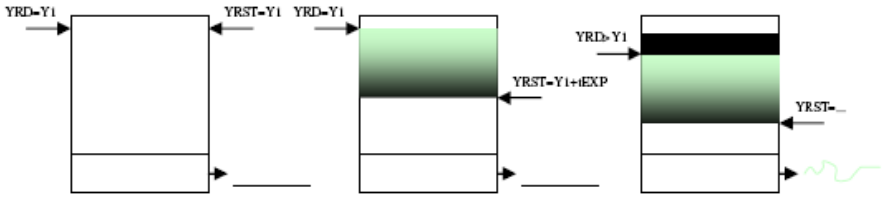


Figure 2.6: DS-DR Readout Scanning

each line it addresses. This leaves the pixel array drained of charges in black or dark state. Then YRD or YRST pointer run over the region of interest of the frame and of each line addressed the pixel's black levels are read out and passed on to the ADC. The user stores these black levels in an off-chip frame sized memory. Then the system is held idling during the exposure time. After the exposure time is elapsed, the frame is scanned once more with the YRD or YRST pointer, and each line is read out again. These signals are passed onto ADC and then to the end user. At the same time, the user retrieves the corresponding black levels from the memory and subtracts them from the signal levels. This is correlated double sampling, eliminating FPN as well as KTC noise. Figure 2.7 shows the Correlated Double sampling process.

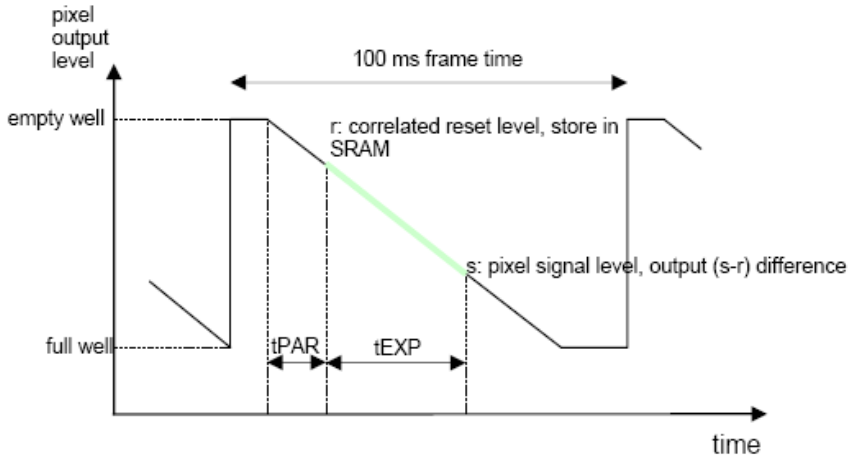


Figure 2.7: CDS-NDR Readout Process

## 2.6 VPU Operation Modes

The following are the operation modes of the VPU

- Full Frame Threshold - This mode gives a full image of the sensor and sends the line_no, pixel_no, and intensity of the pixels that cross a certain threshold intensity

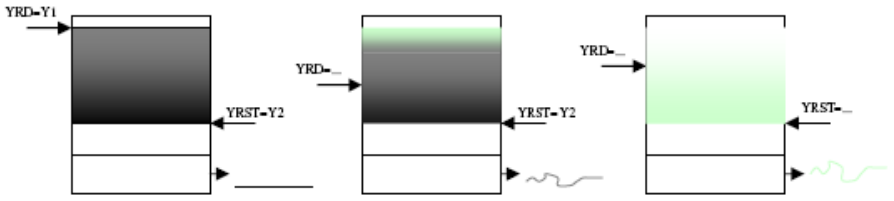


Figure 2.8: CDS-NDR Readout Scanning

- Full Frame Sobel - This mode gives a full image of the sensor and sends the line_no, pixel_no, and intensity of the pixels on an edge i.e whose $ints[pixel+1] - ints[pixel-1]$ crosses a certain threshold
- Window Mode - 10x10, 20x20, 30x30, 40x40 windows can be configured to be read, after specifying their top left co-ordinate. The figure 2.9 shows the window readout.
- The exposure time, the threshold limit along with the number of frames wanted can be configured.

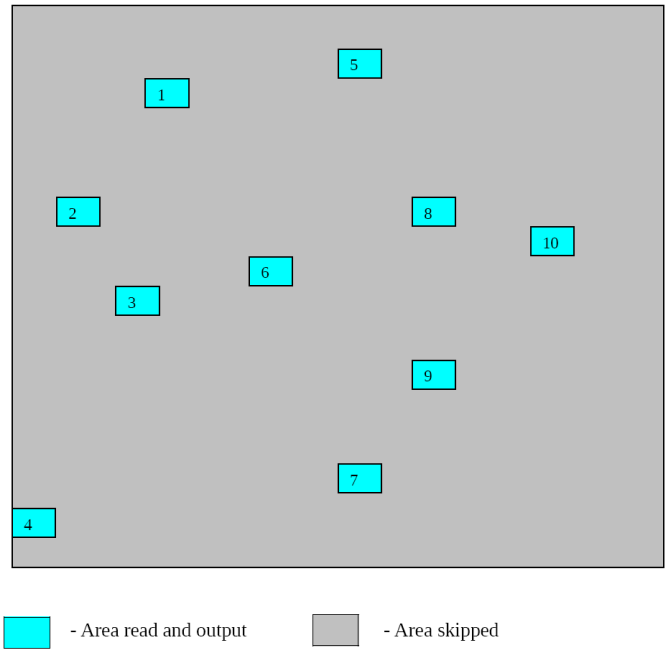


Figure 2.9: Window Mode Readout

# Chapter 3

# Testing Methodology

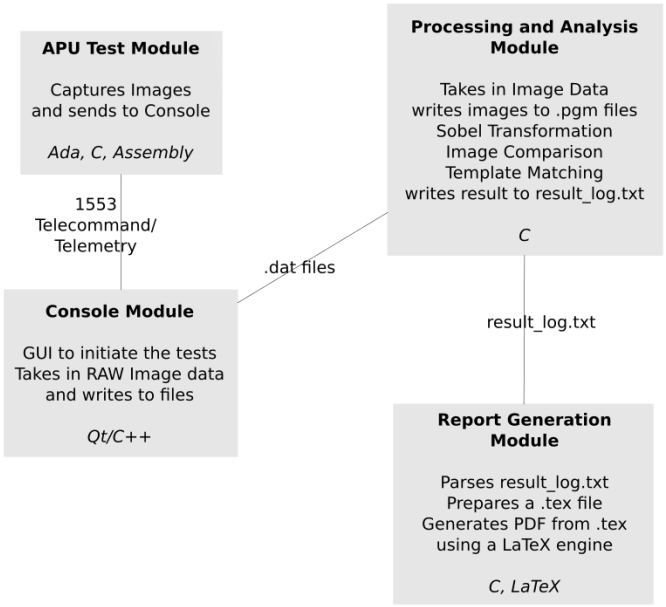


Figure 3.1: Testing Software Modules

## 3.1 Introduction to the methodology

The test system developed primarily deals with acquiring images and House Keeping Data from the Micro Star Tracker and then checking if it is behaving as expected or not.

A series of 27 tests were identified to be of prime importance and test cases were generated accordingly. The first three deal with the acquisition of House Keeping Data, Full Frame Threshold Image and Full Frame Soble Image. The next eight are the acquisition of window mode images and template matching

the with the full frame image. The next eight are the same with NDR mode and checking their size to match with the previous test cases. The next eight are test cases dealing with acquisition of multiple windows in the window mode of operation and the checking their size to match with the expected value.

Four software modules were developed as shown in Figure 3.1. Each with its own structure and function. One software module generates data for the next. Data is sequentially parsed and results are printed in the end as a PDF file.

The different modules are as follows:

1. APU Testing Module- Ada, C, and Assembly
2. Console Module- Qt/C++
3. Processing and Analysis Module- C
4. Report Generation Module- C, and LaTeX

Automation is achieved by using bash shell scripting to execute each of the tasks sequentially

## 3.2 Software Modules

The details of design and implementation of each of the software modules are as followed:

### 3.2.1 APU Testing Module

This is the software module that is uploaded to the EEPROM of the Micro Star Tracker. This module was designed from the legacy code base of the System Program of MST. This code base was written in Ada, C, and Assembly. Only the necessary modules and subroutines of the legacy code were taken to design the MST Testing Module. This module is a lean system program which is small enough to be quickly uploaded to the MST EEPROM.

#### Structure

This module is primarily written in Ada, with a few low level modules written in C and Assembly. The modules required to initialize and drive the MST are in Ada. This module contains modular subroutines to handle each test case. The VPU registers are written using a Buffer. Modular subroutines are written to populate the buffer and get the image data.

#### Function

This module takes in Telecommands from the Console Module and sends in the required image data back to the Console Module as telemetry. After receiving the proper telecommand it populates the registers of the VPU as required by each test case and then commands the VPU to capture images. It waits for image capture and then uploads the raw image data to the Console Module as telemetry through the MIL-STD-1553B Bus.

To advance to the next test the Stand By Command must be sent. To reset and go back to the first test, any other command can be sent.

### 3.2.2 Console Module

This module is a Graphical User Interface which is there in the Star Tracker Console. The Star Tracker console is a computer which connects to the MST using a MIL-STD-1553B bus. This contains a 1553 hardware module which is used as the interface. This console module is an extension of the already present GUI interface software. This GUI interface software is written in C++ and uses the Qt framework for the GUI.

Modifications were made to the legacy code to extend it to handle the VPU test cases.

## Function

When the VPUTEST option is selected under the HAS_TEST option(CMD menu) this module initiates testing. It sends in a SBM(Stand By Mode) command. Waits for the telemetry to arrive and writes the raw telemetry data into a .dat file. It repeats the same exhaustively for all the test cases. It stores all the .dat files produced for each test in the ./vputest folder.

## Structure

This module is an extension written for the Star Tracker Console System. The legacy code base was in Qt/C++ and thus even the added code was also written in the same. The option to initiate the testing environment is in the HAS_TEST menu. The HAS_TEST menu is under the Telecommand menu. This module has a single subroutine which sends in the appropriate telecommand and upon receiving the telemetry it writes the raw data into the file.

All the additional software changes have been made to the tmproc.cpp and tmproc.h files to suit this new test setup.

Figures 3.2 and 3.3 show the screenshot of the console software running and completing the image aquisition.

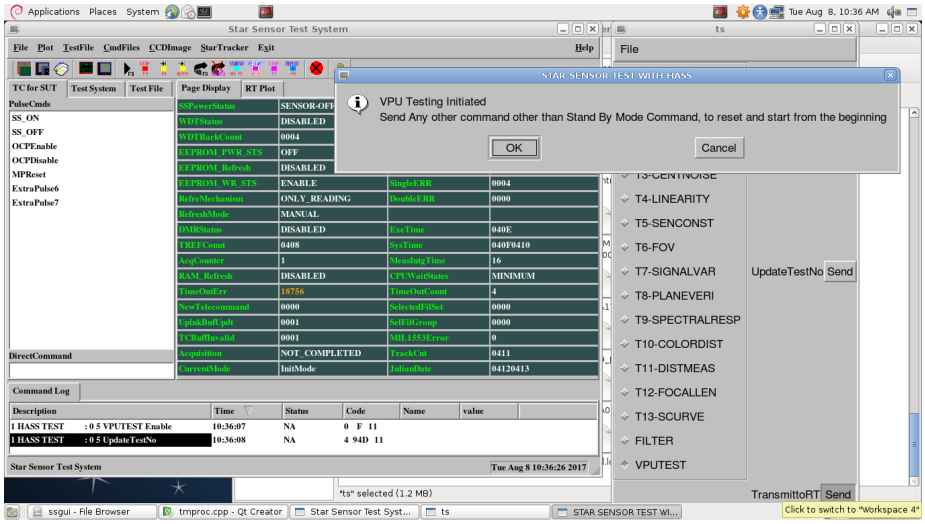


Figure 3.2: Screenshot of the Console GUI

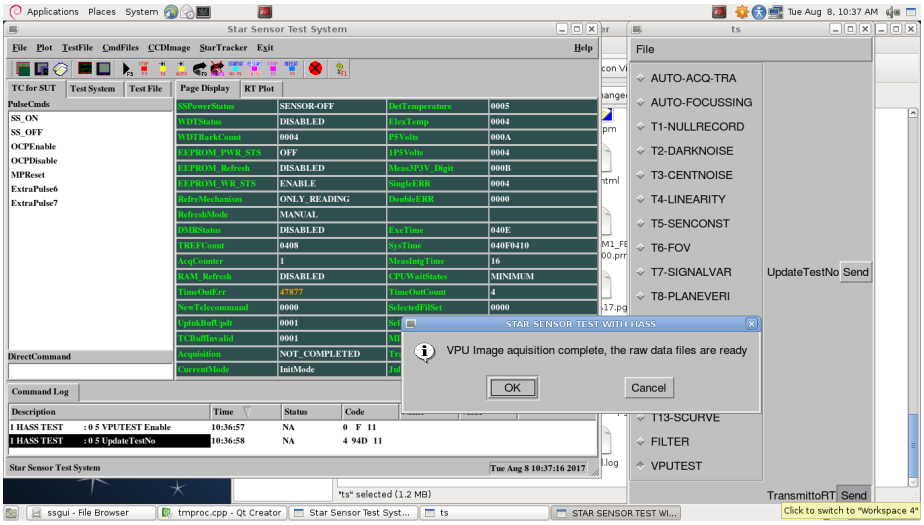


Figure 3.3: Screenshot of the Console GUI after image aquisition

### 3.2.3 Processing and Analysis Module

This module is written in C. It takes in the .dat files generated by the console module and then manipulates and tests them thoroughly.

#### Structure

This module comprises of many functions which implement the required functionalities for each test case. The following major functions implemented in this module and are used in testing

- To write the result to the result_log.txt file

  ```
  void result_write(int check);
  ```

- To validate the given HK by cheking if it is within range

  ```
  int validate_HK(HK_struct hk);
  ```

- To sobel transform a given array

  ```
  void sobel ( int height, int width,int image[][width], int output[][width]);
  ```

- To compare 2 image arrays

  ```
  int compare( int height, int width,int im1[][width], int im2[][width]);
  ```

- To match a template in an image

  ```
  void template_match(int imheight, int imwidth,int image[][imwidth],
  int tempheight, int tempwidth, int temp[][tempwidth], coord *
  ret);
  ```

- To make a pgm image from a given intensity array

  ```
  void make_pgm(char * filename, int height, int width, int max_grey,int
  image[][width]);
  ```

- This returns the filesize of the given file

  ```
  int filesize(const char *filename);
  ```

- To create an intensity array from a raw .dat file

  ```
  int create_fullframe(char * filename, int width, int height, int
  image[][width]);
  ```

- To create an intensity array from a dat file. Note: This writes 0 to HK_struct if HK is not enabled

  ```
  int create_window(char * filename, int width, int height, int image[][width],
  int HK_enable, HK_struct * HK);
  ```

- To create filenames for writing pgm files

  ```
  int filenExtract(char * path, char * filename);
  ```

### Function

This module is the main module which does the actual testing. This uses various techniques for matching and verifying the acquired images. It also checks if the acquired data is of the right size and if the acquired HK data is within range.

This module contains the following functionalities that can be reused for any further testing and test case generation.

1. Building full frame images from the acquired raw data. Full frame data is stored as a 32 bit word containing line number, pixel number, intensity values of pixels crossing a certain threshold. This data is used to build full frame images
2. Sobel transforming a image for edge detection. The image is convolved with a Sobel kernel to produce the transformed image.
3. Template Matching- This is a technique for finding small parts of an image which match a template image. This is especially useful in window mode testing where the acquired window is tested against the full frame image to verify it's starting co-ordinates.
4. Production of images which can be viewed by Users easily. All acquired images are stored as .pgm files which can be opened as images for viewing.

## 3.2.4 Report Generation Module

This module is written in C. It reads the report_log.txt file to see which tests have been passed and which of them have failed. It generates an appropriate .tex file with all the details of the test. This .tex file is further passed to any LaTeX engine(here pdfLaTeX) which generates the report of the test.

### Structure

This module has a modular subroutine which writes YES/NO to the .tex file after parsing the test result This subroutine can be called after writing the appropriate statements to the .tex file.

### Function

The module generates a .tex file and then produces a easily readable PDF file which lists all the Test cases and the result. An example of such a PDF Report is given in the Appendix.

## 3.3 Test cases

The following 27 tests were identified and implemented. These tests were chosen to vary in the testing methodology and to cover the various basic modes of operation of the VPU.

Each of the tests are written in a modular manner to facilitate any further additions to the test cases. Results are written to *result_log.txt* file.

The tests are:

1. Only test the HK data. Only VPU HK data is recieved as telemetry. It is varified to be within bounds and the results are printed.

2. Full Frame threshold mode test. Here one full frame (1024 x 1024) threshold values are taken as telemetry from the Star Tracker. These threshold values are used to build the full image with the background at perfect black. This image is checked for size and stored as a *.pgm* file. This file can be opened with an image viewer.

3. Sobel Mode Testing. Here one full frame image (1024 x 1024) pixel intensity values are taken as telemetry from the Star Tracker in the Sobel Mode. These values are used to build a full image. The full frame threshold image acquired in the previous test is convolved with a Sobel Kernel to produce an Edge Detected Image with which the acquired Sobel Image is matched. The result is printed after veryfying the match. The sobel image is stored in a file as *.pgm* image as well.

4. Tests 4 to 11 are Window Mode Tests in the Destructive Readout(DR) mode. In test 4 a single 10x10 frame is acquired. This image is template matched with the full frame image to see if the starting co-ordinates are as expected. For this the `void template_match(int imheight, int imwidth,int image[][imwidth], int tempheight, int tempwidth, int temp[][tempwidth], coord * ret)` function is used. The result is written.

5. Test 5 is the same as test 4, except now HK Data is acquired along with the image and checked for correctness.

6. Test 6 is similar to test 4, but for a 20x20 frame.

7. Test 7 is the same as test 6, except with HK Data.

8. Test 8 is similar to test 4, but for 30x30 frame.

9. Test 9 is the same as test 8, except with HK Data.

10. Test 10 is similar to test 4, but for 40x40 frame.

11. Test 11 is the same as test 10, except with HK Data.

12 – 19. Tests 12 to 19 are the same as those from 4 to 11, except here they are done in the Non–Destructive Readout Mode.

20. Tests 20 to 27 are multi window tests. 20 frames of images are received and the size of the received data is verified. This is done for all available window sizes in both DR and NDR modes.

    Test 20 is to test the acquisition of 20 frames of 10x10 windows

21. This is the same as test 20 but in NDR mode.

22 – 23. These are tests for 20 frames of 20x20 images in DR and NDR mode.

24 – 25. These are tests for 20 frames of 30x30 images in DR and NDR mode.

26 – 27. These are tests for 20 frames of 40x40 images in DR and NDR mode.

# Chapter 4

# Conclusion and Future Work

### 4.0.1 Conclusion

In this project a Test Automation system to test and evaluate the Micro Star Tracker Video Processing Unit was designed and implemented. The system was designed in a modular form for easy customization and reuse for future applications. The need for a test automation system has been proven and one such system has been designed and implemented.

### 4.0.2 A Note on the Experience

This project has been a fantastic learning experience and has taught various techniques and methods employed in the industry. Examples for which include:

–Debugging in a professional setting.

–Working with legacy Code, and modifying it to extend its capabilities.

–A familiarity with professional systems software

–Study and Writing of Technical Literature

–Best practices in the Industry regarding various design and development processes

–The Methods employed in the space industry to build fault tolerant systems.

### 4.0.3 Future Work

The code is written in a modular form. This enables one to easily add more test cases or remove existing ones. Each of the software modules is separate and thus can be individually reused after minor changes to suit other Star Tracker variants. The scope of future work lies in adding more test cases to further evaluate the VPU behaviour. While twenty seven test cases were implemented here, there is always room for more. Various other methods can be employed to manipulate and compare images, which may further increase the accuracy of the tests.

# Bibliography


[1] AIM GmbH *MIL-STD-1553 Tutorial v1.3*. AIM GmbH Sasbacher Str. 2, 79111 Freiburg, Germany

[2] *CHU Command and Data Interface Document*. CSSG/SDA LEOS-ISRO

[3] *Gsat-19 Star Sensors Initial Onboard Performance Report*. Doc No: LEOS-SDA-CSSD-GS19-DOC-17-20-00

[4] *Mark-III Star Sensor Design Document*. SDA/CSSD/CSSG LEOS-ISRO

[5] S Padmasree, *User Manual for Mark-III Star Sensor(Multi Head Configuration)*. CSSD/CSSG LEOS-ISRO

[6] *SPARCV8 Ip Core Processor Configuration Document For Micro Star Tracker* CSSG LEOS-ISRO

[7] *The SPARC Architecture Manual Version 8* SPARC International, Inc.

[8] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice* Wiley, ISBN 978-0-470-51706-2, 2009

[9] Ludwig, Jamie (n.d.). *Image Convolution* Portland State University.

[10] Irwin Sobel, *History and Definition of the Sobel Operator*, 2014

# Appendix I - A sample test report