



UPPSALA
UNIVERSITET

IT 13 002

Examensarbete 30 hp
Januari 2013

M2 Simulator Model

Rahbee Alvee



UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet
UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
<http://www.teknat.uu.se/student>

Abstract

M2 Simulator Modul

Rahbee Alvee

M2 is a mixed analog/digital Application Specific Integrated Circuit (ASIC) that is used as an IO controller at Ruag Space. The thesis aims to develop a simulation model of an M2 ASIC for using in Ruag simulator and examine how this model can be made compatible with Simulation Model Portability (SMP) version 2. SMP2 is a standard developed by the European Space Agency (ESA) to simulate models together with European companies in the aerospace industry. The purpose of SMP2 is to enable portability between different simulation platforms. The outcome of this thesis is an M2 simulation model based on known theories and an investigation of SMP2 that implies how M2 can be adopted to be SMP2 compliant. The result of the SMP2 investigation also indicates partially how the existing simulation models in the current Ruag simulator can be converted to SMP2 models.

Handledare: Anders Petersén
Ämnesgranskare: Leif Gustafsson
Examinator: Philipp Rümmer
IT 13 002
Sponsor: RUAG Space AB

Tryckt av: Reprocentralen ITC

Acknowledgement

First and foremost I would like to offer my sincerest gratitude to my thesis supervisor Anders Petersén who guided and supported me throughout the whole project. Working under his supervision was a great pleasure as I had freedom to work and think on my own. I would like to thank my thesis reviewer Leif Gustafsson for helping me in the report writing.

I am thankful to Jan Georgesson, Olle Martinsson and Patrik Sandin since without their help it would be impossible to complete the thesis project successfully. Jan Georgesson guided me to get familiar with the Ruag simulator and to implement the M2 ASIC simulator model. Olle Martinsson helped me understand the M2 ASIC functionalities. Patrik Sandin helped me to understand the implications of SMP2 over the current simulator.

Thanks to my parents for encouraging me for the Master studies in Uppsala University, Sweden and for supporting me throughout my entire life.

Contents

- 1 Introduction 12
 - 1.1 Background 12
 - 1.2 Purpose 12
- 2 M2 ASIC 12
 - 2.1 Functional Summary 14
- 3 Implementation of M2 ASIC Model 16
 - 3.1 Modes of M2 18
 - 3.2 Interrogation 18
 - 3.2.1 Memory Load (ML) Interrogation 18
 - 3.2.2 Data Acquisition (DA) Interrogation 19
 - 3.2.3 On/Off Command (OO) Interrogation 19
 - 3.3 Response 20
 - 3.4 Interrogation Parsing 20
 - 3.4.1 ML Interrogation Parsing 21
 - 3.4.1.1 Write Operation for M2 Configuration Registers 22
 - 3.4.1.2 ML16 Operations 24
 - 3.4.1.3 Write Operation on Output port 25
 - 3.4.1.4 OO Interrogation Parsing 27
 - 3.4.1.5 HLC Command 27
 - 3.4.2 Data handling through TCP/IP 28
 - 3.4.3 DA Interrogation Parsing 30
 - 3.4.3.1 AN/TH Acquisition 30

3.4.3.2 DB/DR Acquisition.....	31
3.4.3.3 DS16 Acquisition.....	32
3.4.3.4 Input port registers	33
3.4.3.5 Output port registers read operation	34
4 Simulation Model Portability	34
4.1 Background.....	34
4.2 SMP2 Architecture	36
4.2.1 Simulation Environment.....	36
4.2.2 Operational phases	38
4.3 SMP2 Mechanism.....	39
4.4 Inter model communication	40
4.4.1 Interface based design	40
4.4.2 Event based design	40
4.4.3 Dataflow based design	41
4.5 SMP2 adaptation of M2 ASIC Model.....	41
4.6 SMP2 adaptation for the current SMU simulator.....	44
5 Tests and Results.....	45
5.1 Test Environment.....	45
6 Discussion	47
7 Conclusion	48
7.1 M2 Simulator Model.....	48
7.2 Simulation Modeling Portability	48
8 References	49

Appendix A : Abbreviations.....	50
Appendix B: Interrogation List	52
Appendix C: Test results.....	57

List of Figures

Figure 1	Functional block diagram of M2 ASIC	13
Figure 2	M2 application example	14
Figure 3	M2 ASIC Environment	17
Figure 4	State diagram of M2	17
Figure 5	Basic Interrogation Format	18
Figure 6	Interrogation format in memory	18
Figure 7	Memory Load Interrogation Format	19
Figure 8	Data Acquisition Interrogation Format	19
Figure 9	On/Off Interrogation Format	19
Figure 10	Response format 13/21 bits	20
Figure 11	Interrogation Parsing	21
Figure 12	ML Interrogation Parsing	22
Figure 13	Write operation for M2 Configuration Registers	23
Figure 14	ML16 operation	25
Figure 15	Write operation Output port	26
Figure 16	Port out 1234	26
Figure 17	Port out 5678	26
Figure 18	OO Interrogation Parsing	27

Figure 19	HlcLength\$ Config register	27
Figure 20	HLC Command operation	28
Figure 21	DA Interrogation Parsing	30
Figure 22	AN/TH Acquisition	31
Figure 23	DB/DR Acquisition	32
Figure 24	DS16 Acquisition	33
Figure 25	Input port registers	33
Figure 26	Output port registers Read operation	34
Figure 27	SMP2 architecture	36
Figure 28	State Diagram of Simulation Environment	37
Figure 29	SMP2 operational phases	39
Figure 30	SMP2 Mechanism	39
Figure 31	Interface based communication	40
Figure 32	Event based design	41
Figure 33	Dataflow based design	41
Figure 34	SMP Component	42
Figure 35	IObject interface	42
Figure 36	IComponent interface	43
Figure 37	IModel interface	43
Figure 38	SMU core simulator	44
Figure 39	SMP2 compliant SMU core simulator	45
Figure 40	SMU Core Simulator	46
Figure 41	External Application	46

Figure 42	Result output for interrogation list	47
Figure 43	Ouput for ML interrogations	59
Figure 44	AN interrogation	60
Figure 45	TH interrogation Results	61
Figure 46	DB/DR Interrogation Results	62
Figure 47	DS16 acquisition	63
Figure 48	OO Interrogations and results	64
Figure 49	ML16 interrogations and results	65

List of Tables

Table 1	IO Group Configuration Table	23
Table 2	TCP/IP Message Data Format	29

1 Introduction

1.1 Background

Ruag Space develops and manufactures equipment to use in space, mainly computers and computer equipment and related software, control systems, antennas and microwave. The equipments are part of launchers, satellites and other spacecraft. Ruag Space is developing a data management system software which consists of hardware drivers. To support the development of this software a simulator is designed based on the Tsim Aeroflex Gaisler [1], which is a cycle-based simulator for a ERC32 [2] /LEON [3] processor.

The M2 ASIC [4] constitutes a highly integrated, low-power core component for digital I/O and analog data acquisition in spacecraft data handling systems. M2 contains AD converters, analog channels for thermistor measurements, digital IO functions for generating command pulses, a general IO port and a control interface that can be either an OBDH [5] bus or an UART.

Simulation Modeling Portability version 2 [6] was released in 2004 and after that it has updated several times. The purpose of the SMP2 Standard is to promote portability of models among different simulation environments and operating systems, and to promote the reuse of simulation models.

1.2 Purpose

The simulator that Ruag Space has developed is written in C++ and executes on a standard Linux computer. The link to the outside world is done using TCP / IP sockets. Being an internal tool for software development, the simulator has evolved to become a product for Ruag's customers. So it is becoming important that peripherals such as IO cards are represented in the simulator too. To serve this purpose the thesis aims to develop a simulator model of the M2 ASIC using C++ programming language and it also involves investigation of the SMP2 standard. Ruag Space does not have any previous experience on SMP2. Investigation of the SMP2 standard should give direction on how to make SMP2 compliant simulator models. It should also indicate how to make the existing SMU simulator SMP2 compliant.

2 M2 ASIC

This section focuses on how the M2 ASIC works and its functionalities that Ruag space uses right now. The functional block diagram of M2 is given below. The OBDH Remote Terminal (RT) block receives interrogations from OBDH Central Terminal (CT) and transmits responses according to the specifications. The OBDH control block compares the Terminal Address Field

(TAF) per interrogation to the remote terminal address RtAddr(4:0) inputs. In case of no match, the Terminal Data Field (TDF) field is ignored and no response is generated. In case of matching address the interrogation is further handled by the CONFIG AND CMD CTRL block for ML or OO interrogations, otherwise, i.e. in case of DA, by the ACQ CTRL block. The UART / OBDH Bridge receive bytes on the UART Rx and convert them into interrogations. Responses are converted into bytes which are transmitted over the UART Tx. There are 8 IO groups which can be configured for intended functionalities. Each IO group contains one input and four outputs. The analogue block is responsible for providing analogue data from sources and then converts it to digital data.

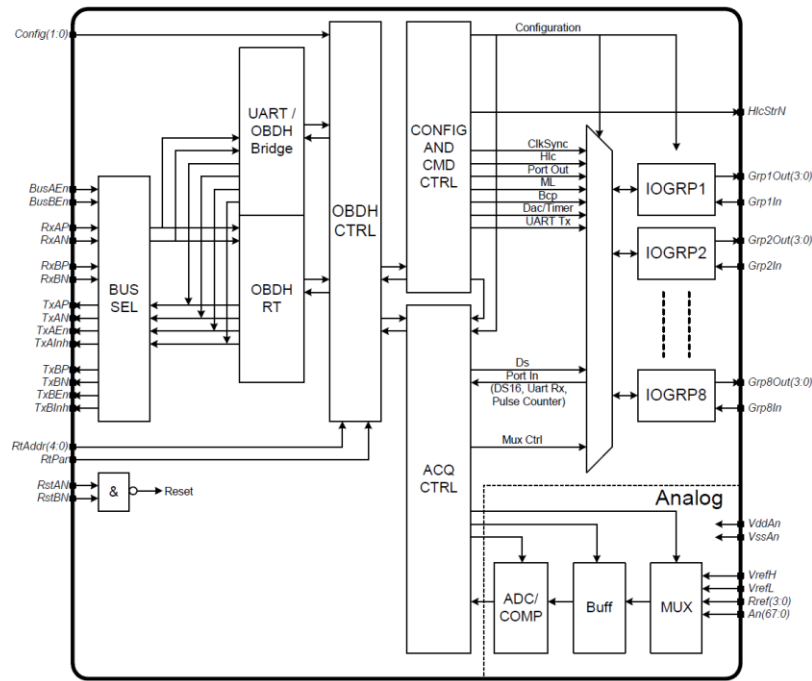


Figure 1 Functional block diagram of M2 ASIC

The hardware configuration for a typical application for the M2 ASIC on a standard IO board should look like following diagram. This is an example where it is shown how the M2 ASIC should be connected along with its support circuits on a standard IO board.

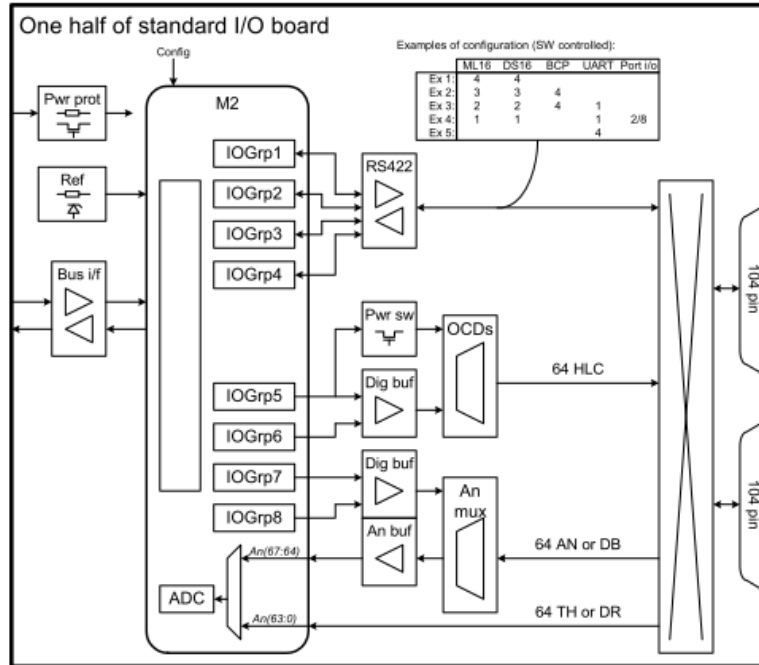


Figure 2 M2 application example

2.1 Functional Summary

The M2 ASIC hardware is capable of doing the following functions. But the software model is only capable to perform those functions which are currently used by Ruag Space. In this section all the functionalities of the M2 ASIC are briefly described with the limitations of the software model.

OBDH-RT for OBDH bus control of the M2

When M2 gets an interrogation from the OBDH-CT the OBDH-RT takes control the OBDH bus and sends response according to the interrogation. After that bus control transfers to OBDH-CT. This functionality is implemented for M2 ASIC model.

UART / OBDH bridge for UART control of the M2

In the UART mode interrogation and responses are handled by the UART / OBDH bridge block. It receives bytes from the OBDH-CT by the UART Rx which is interpreted as interrogation and then respective response is generated and converted into bytes which are transmitted over UART Tx. This functionality is not implemented as the software model does not support the UART mode of operation. The UART is not used in any application of M2 ASIC used by Ruag Space.

12 bit ADC for AN and TH channels

The real M2 ASIC hardware deals with analogue data from the analogue channels. The Analogue to Digital Converter (ADC) converts the analogue data to 12bit digital data. But in the simulator we deal with only digital data.

Comparator for DB and DR channels

The comparator is used specially to compare the values of two different source of the same type. In the software model implementation traditional comparison operator is used to compare values.

Switchable resistive conditioning for TH and DR channels

In M2 ASIC resistive conditioning of the reference resistance values is possible. The same facility is available in the software model as well.

Control of external multiplexer for AN and DB channels

The real M2 ASIC uses internal signals to control external multiplexing functionalities. The IO groups need to be configured to get desired multiplexer. The software model reads the multiplexing configuration from the IO group configuration registers and uses programming techniques instead of using internal signals.

Generation of control signals for HLC

High Level Command (HLC) is a pulse command and its width is programmable. In the software implementation the pulse width is calculated and appropriate response is sent back to the OBDH-CT.

ML/DS channel interfaces

Memory Load (ML) and Digital Serial (DS) interfaces involve IO Groups configured in ML/DS operation where data is sent or received serially through these interfaces. Serial operation is not supported by the software model as the outer interface is connected through TCP/IP. Data is sent or received through TCP/IP.

Broadcast pulse generation

A broadcast pulse is used in the UART operation to broadcast a byte to all the UART channels configured to receive broadcast messages. This function is not implemented as UART is not used.

UART as user I/O interface

As the UART is not a concern for this software model implementation the user interface for I/O in the UART is also skipped.

Support function for HLC matrix commands, HLCM

The software model supports parallel generation of HLC commands but it supports up to 64 channels where the real M2 ASIC supports up to 256 channels.

Support function for DR matrix acquisition, DRM

A Digital Relay (DR) matrix is used to acquire up to 256 channels for DR acquisition. It is supported by the software model.

General purpose port

A general purpose port is supported by the model where there are eight IO groups and each IO group contains four outputs and one input. IO groups can be configured by the IO group configuration registers.

Pulse counter, possible to latch by external signal or BCP

A pulse counter is not supported by the software but the registers for the counters can be configured for the future use.

Digital part of first order $\Sigma\Delta$ DAC

Since we are dealing with only digital data Digital to Analogue Converter (DAC) is not used.

Timer for e.g. valve control, possible to trigger by external signal or BCP

Timers are signal generators and are not implemented in the software. But the configuration registers are there for future use.

3 Implementation of M2 ASIC Model

The model for the M2 ASIC is developed based on the specification of the real hardware. The software model does not support all the features. But it is made in such a fashion that additional functionalities can be added later depending on the requirements. Since the software model does not represent the complete model of real M2, it does not support all the functionalities described above. It does not support UART, ADC or DAC, pulse counter, broadcast pulse generation and Timer.

The following environment is considered for the implemented M2 ASIC:

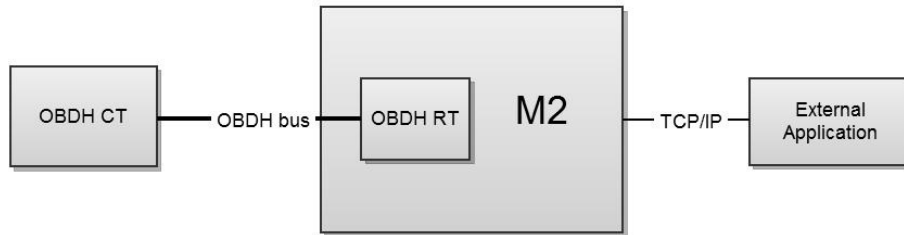


Figure 3 M2 ASIC Environment

M2 is a part of an IO board that is shown in Figure 2. M2 contains OBDH RT that is connected to the OBDH CT. M2 communicates to the outer world through TCP/IP. An application (External Application) is implemented that sends or receives data over TCP/IP. M2 does its operation in a very simple manner. It receives interrogations through OBDH bus from the OBDH CT, performs operations accordingly, generates responses and gives it back to the OBDH CT through OBDH bus.

Here is the state diagram of M2 model:

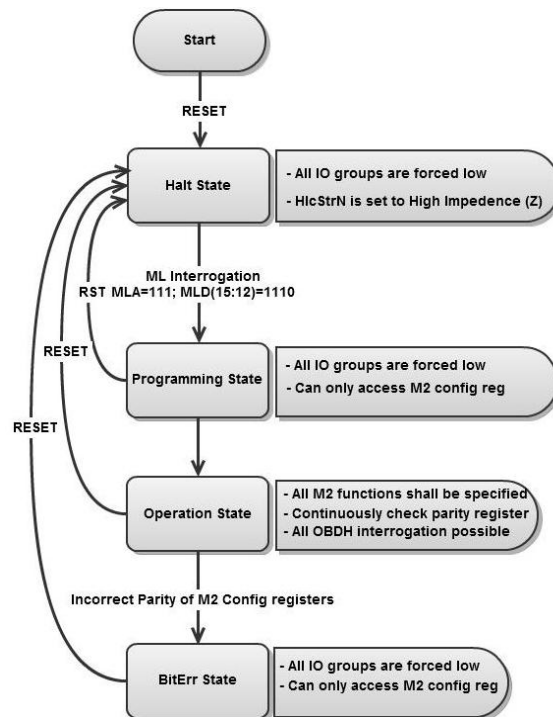


Figure 4 State diagram of M2

3.1 Modes of M2

M2 can be operated in one of the three modes: OBDH mode, UART mode and Test mode. The M2 model supports only OBDH mode since the UART and Test mode are not used by the current Ruag Simulator.

3.2 Interrogation

An interrogation is an instruction or a command from OBDH CT to one of the OBDH RTs to perform a specific operation. Interrogations are stored in a block of memory [7]. Each interrogation is a 32 bits word. The basic interrogation format is given below:

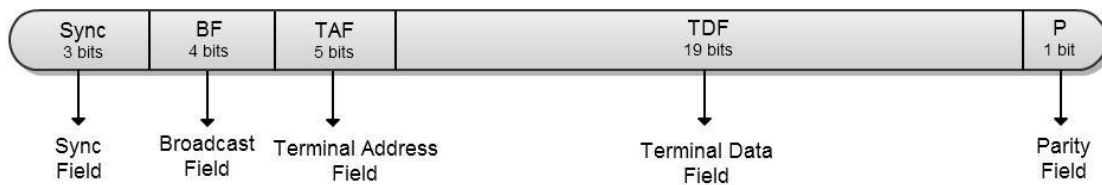


Figure 5 Basic Interrogation Format

The sync field is used for synchronization purpose of an Interrogation in the hardware implementation. Since the OBDH bus software model also supports sync field so it is kept in this implementation but not used. BF field conveys information to all terminals simultaneously. TAF field contains the address of the remote terminal. TDF contains the data required for the remote terminal to perform its given task. Even parity is used for the preceding 28 bits in the parity bit field.

An interrogation is stored in the memory location in the following manner:



Figure 6 Interrogation format in memory

3.2.1 Memory Load (ML) Interrogation

Memory load interrogation is used for write operation either in M2 registers or IO port. It has the following format:

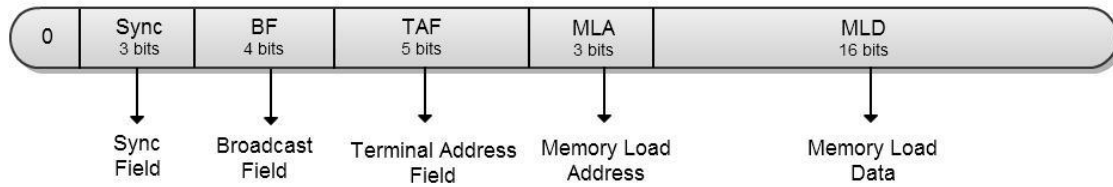


Figure 7 Memory Load Interrogation Format

The terminal data field is divided into two parts: MLA and MLD. MLA defines the address and MLD defines the data to be written.

3.2.2 Data Acquisition (DA) Interrogation

Data acquisition is a read operation. In this case the TDF is divided into few parts where the least significant 8 bits are kept for the analogue channel addresses. MOP defines the operating mode. The next bit beside MOP is 1 to identify that it is a DA interrogation. DEA is the destination address where the response should be sent and in this case it is always 0000 which is the address of the OBDH CT.

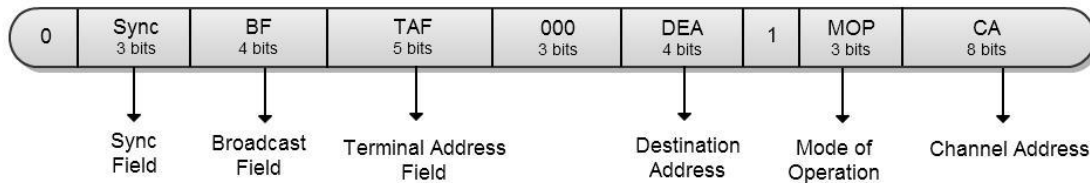


Figure 8 Data Acquisition Interrogation Format

3.2.3 On/Off Command (OO) Interrogation

OO interrogation is used to generate a special kind of command called High Level Command (HLC). It has similar instruction format like DA interrogation but the only difference is that the next bit beside MOP is 0.

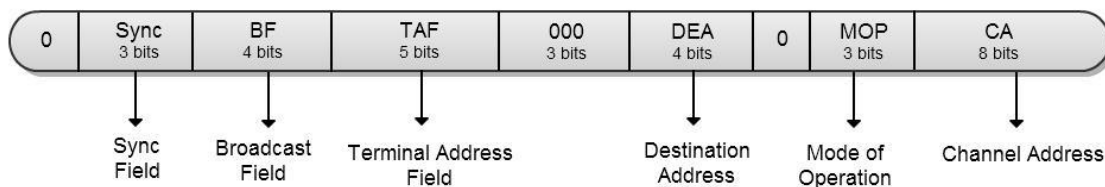


Figure 9 On/Off Interrogation Format

3.3 Response

A response is a reply from the OBDH RT to the OBDH CT for an interrogation. A response can be 13 bits or 21 bits of length depending on the kind of interrogation.

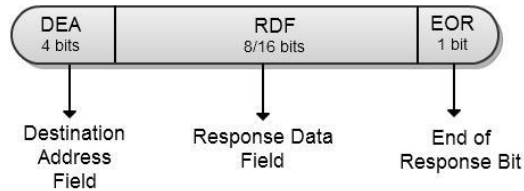


Figure 10 Response format 13/21 bits

The OBDH CT Response Register is a 32 bit register where the least significant 16 bits are used as response data field (RDF). So in the implementation only RDF is sent as response.

3.4 Interrogation Parsing

Each interrogation has a specific function. So it is important to parse an interrogation correctly and map to a correct function for that interrogation. When an interrogation is received by the OBDH RT it checks if the interrogation is 32 bits long. Then it checks for the Terminal Address and if 00000 is found then it is a dummy interrogation and no response will be generated. If the TAF matches then MLA is checked. If MLA is greater than 0 then it is an ML interrogation. But if MLA is 000 then it can be either OO or DA interrogation. Then the 11th bit from LSB is checked. If 1 is found then it is a DA interrogation otherwise it is an OO interrogation. The following diagram describes how an interrogation can be decided:

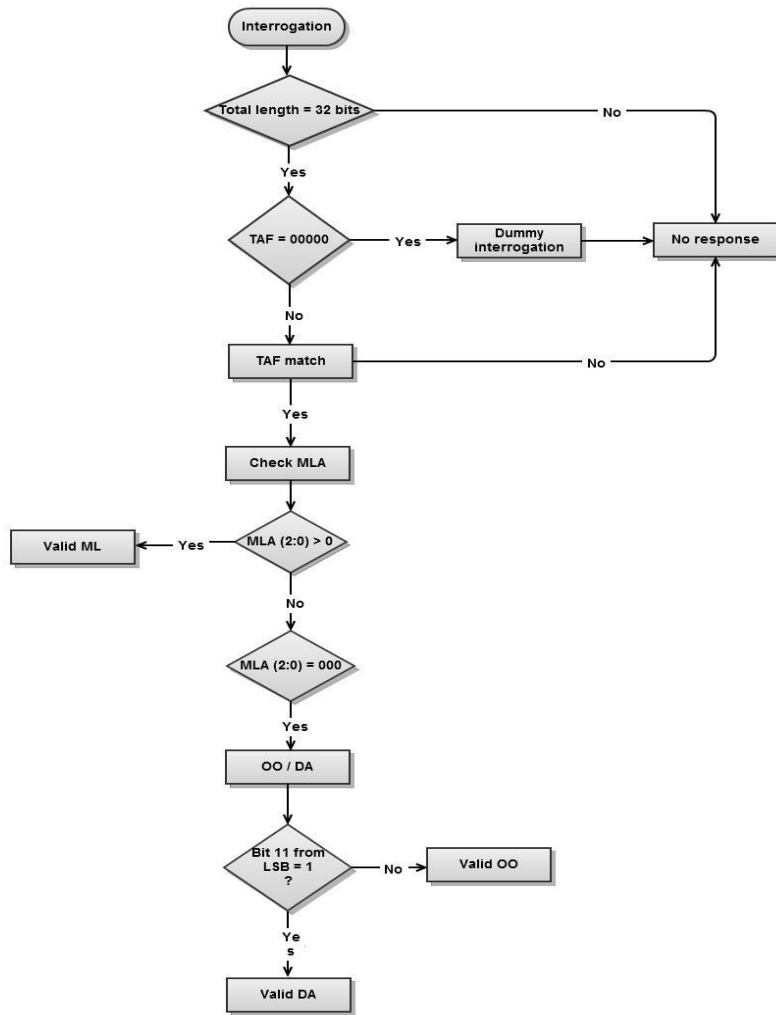


Figure 11 Interrogation Parsing

3.4.1 ML Interrogation Parsing

After it is decided as an ML interrogation then OBDH RT will check for the MLD field to map an appropriate function. According to the diagram below, MLA = 000 does not exist. But for MLA '001', '010', '011', '100', '101', '110' it can be either ML16, IO port or UART operation depending on the configuration of the intended IO group.

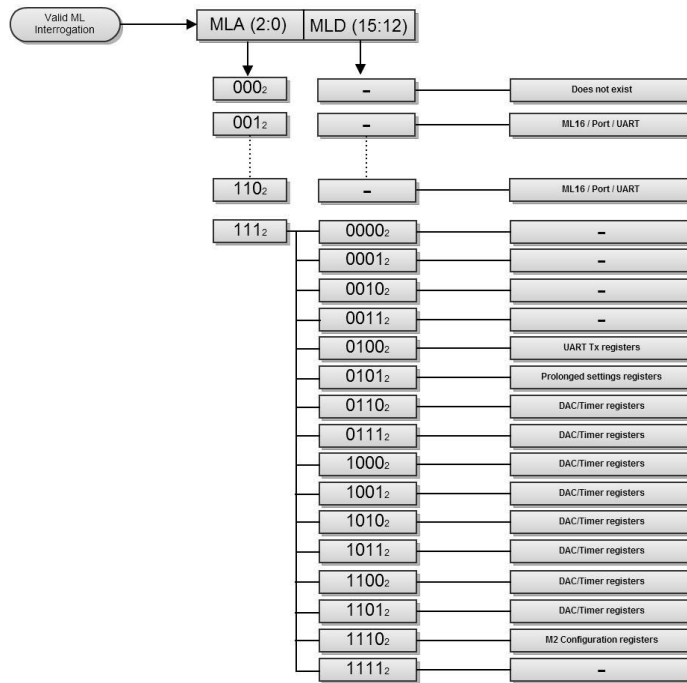


Figure 12 ML Interrogation Parsing

MLA = '111' is used to configure M2 registers. Depending on the value of MLD (15:12) different registers are selected for desired configurations.

3.4.1.1 Write Operation for M2 Configuration Registers

A valid ML interrogation with MLA '111' and MLD (15:12) '1110' can be used to configure the configuration registers of M2 ASIC.

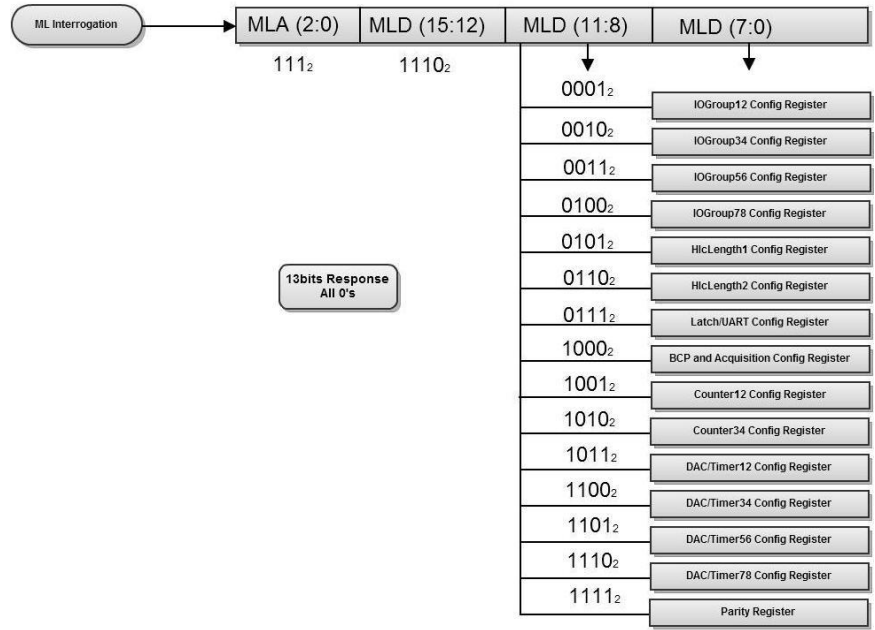


Figure 13 Write operation for M2 Configuration Registers

There are 15 configuration registers and a parity register. MLD (11:8) bits are used to select a register and MLD (7:0) contains the configuration data. Each IO group can be configured as one of the following configurations.

Configuration	IO Function
0000	Disabled
0001	Port
0010	SyncClkStr
0011	DacTimer1234
0100	ML16 ₁ DS16 ₁
0101	DacTimer5678
0110	DS16 ₃
0111	UART
1000	BCP

1001	Mx16
1010	Mx128
1011	Mx256
1100	HlcSer
1101	HlcPar4
1110	HlcPar64/ML16 ₂
1111	HlcPar256

Table 1 IO Group Configuration Table

SyncClkStr is used to control the IO synchronization. DacTimer is used for either DAC or Timer functionality. ML16_iDS16_i is used for both ML16 and DS16 operation. Only IOGroup1 to IOGroup6 can be configured in ML16_iDS16_i. ML16₂ supports only ML16 operation. DS16₃ is used for DS16 operation only. The UART and BCP are not used. Mx16, Mx128 and Mx256 are used to control external multiplexers for up to 256 channels. There are two HLC Length registers which can be configured. Other registers can be configured but are not used for operational purpose as the UART is skipped in this implementation. Hlc can be serial or parallel operation. The model can handle up to 64 channels whereas the real M2 can handle up to 256 channels. Hlcp₆₄ applies to group 4 to 7 and ML16₂ applies to group1 to group3.

3.4.1.2 ML16 Operations

An ML16 operation requires an IO Group to be configured first. An IO Group can be configured as either ML16_iDS16_i or ML16₂ depending on the operation of choice. ML16_iDS16_i supports both ML16 and DS16 operation while ML16₂ supports only ML16 operation. IO Group 1 to IO Group 6 can be configured as ML16_iDS16_i for MLA = 1 to MLA = 6 respectively but for ML16₂ configuration two channels will map to the same IO Group such as MLA = '001' and '100' map to IO Group1, for MLA = '010' and '101' IO map to Group2, MLA = '011' and '110' map to IO Group3. ML16 is a serial operation but it is implemented as a parallel operation as there is no external circuitry used in this case.

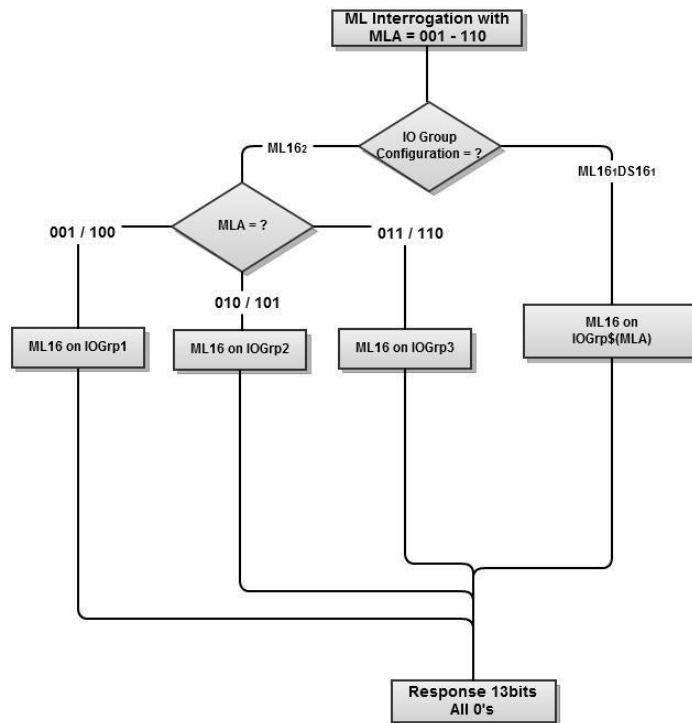


Figure 14 ML16 operation

ML16 command is sent to the TCP/IP port using the configured IO group. There should be a user defined application that receives the data and take necessary action according to the command received.

3.4.1.3 Write Operation on Output port

For an ML interrogation with MLA '001', '010', '011', '100', '101', '110' and an IO group configured other than ML16_iDS16_i, ML16₂ or UART is considered output port write operation for that IO group.

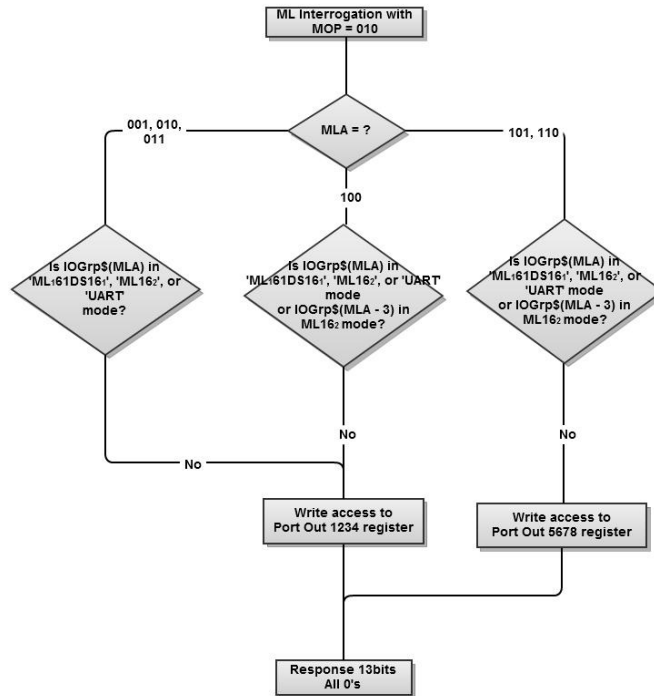


Figure 15 Write operation Output port

There are two 16 bits output port registers Port out 1234 and Port out 5678. Values are written directly to the registers.

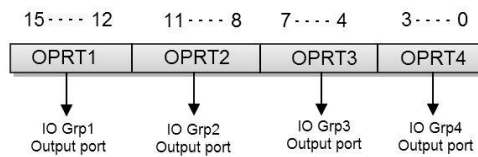


Figure 16 Port out 1234

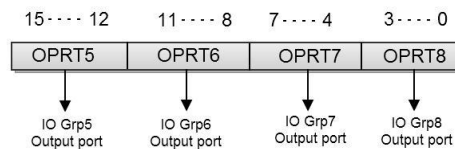


Figure 17 Port out 5678

3.4.1.4 OO Interrogation Parsing

An OO interrogation contains MLA = '000', destination address of OBDH CT which is '0000' and the 11th bit from LSB is always 0. HLC commands will be issued only for MOP '011' and '100'. CA field may contain channel addresses for getting signals from interfacing circuits.

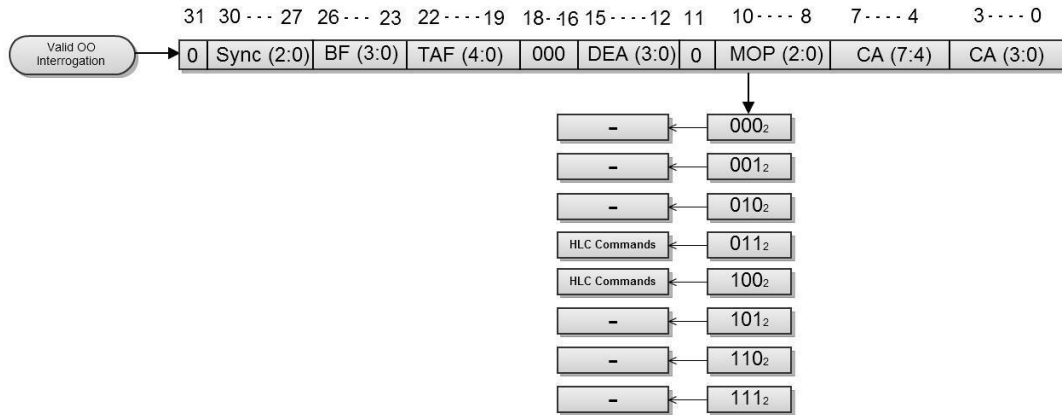


Figure 18 OO Interrogation Parsing

3.4.1.5 HLC Command

M2 can control the width of a HLC command pulse. When MOP = '011' then HlcLength1 Config register will be selected and pulse length of HLC will be calculated based on the value of that register. But if MOP = '100' then pulse length of HLC will be calculated from HlcLength2 Config register.



Figure 19 HlcLength\$ Config register

HLC command can be a serial or a parallel operation depending on the configuration of the IO groups. IO group 5 to 8 can be configured as HlcSer and HlcPar4. But IO group 4 to 7 can be configured as HlcPar64.

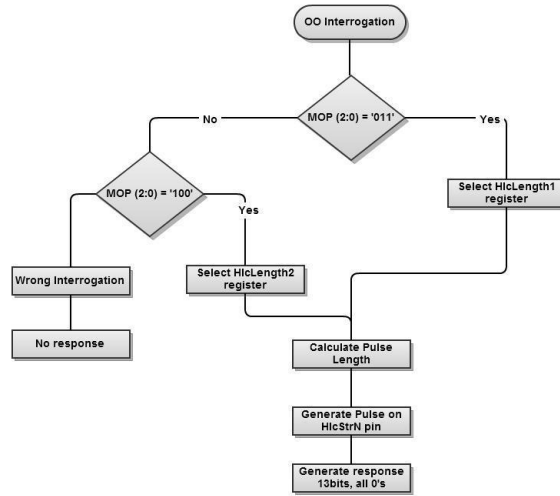


Figure 20 HLC Command operation

The HLC pulse length is calculated using the following rule:

$$T_{\text{Pulse}} = 2^{(N_1+5)}(N_2+8)T_{\text{OBDH}}, N_1 \in [0, 15], N_2 \in [0, 7]$$

Where $N_1 = \text{HicLength}(3: 0)$; $N_2 = \text{HicLength}(6: 4)$ and T_{OBDH} is the period of an IO clock. In actual hardware implementation HLC operation involves several necessary signals for synchronization purpose where in the software model they are not used as the synchronization is ensured by the OBDH CT.

3.4.2 Data handling through TCP/IP

There is an external application for handling external data to the system. It sends and receives data through TCP/IP to simulate the data for analogue channels and IO ports. Data can be for DA/DB, AN/TH, ML/DS operation. Data transfer through TCP/IP is done through messages where, Message = (Message ID + Data length + Data). Message ID is a number that represents a specific operation e.g. '01' for analogue channel (AN) acquisition. Data length is the length of data in number of bytes. Data can be different based on the type of operation. Data should be prepared in the following structure:

Operation	Data format	Description
AN acquisition	0106300500	01 = Message ID for AN 06 = Length of data in bytes 30 = Analogue channel number 0500 = value for 5.00
TH acquisition	0206300500	02 = Message ID for TH 06 = Length of data in bytes 30 = TH channel number 0500 = value for 5.00
DB acquisition	03103001001110	03 = Message ID for DB 10 = Length of data in bytes 30 = DB channel number 001001110 = digital values for 8 channels
DR acquisition	04103001110011	04 = Message ID for DR 10 = Length of data in bytes 30 = DR channel number 001110011 = digital values for 8 channels
DS16 acquisition	05161111000011110000	05 = Message ID for DS16 16 = Length of data in bytes 0xF0F0 = DS16 value
Single Channel Digital Data	0603641	06 = Message ID 03 = Length of data in bytes 64 = Channel Address 1 = value

Table 2 TCP/IP Message Data Format

3.4.3 DA Interrogation Parsing

A DA interrogation differs from an OO interrogation by 11th bit and it is 1. DA operation is mainly read operation. Depending on the mode of operation values it can request for analogue/thermistor values, DS16 acquisitions and different register values.

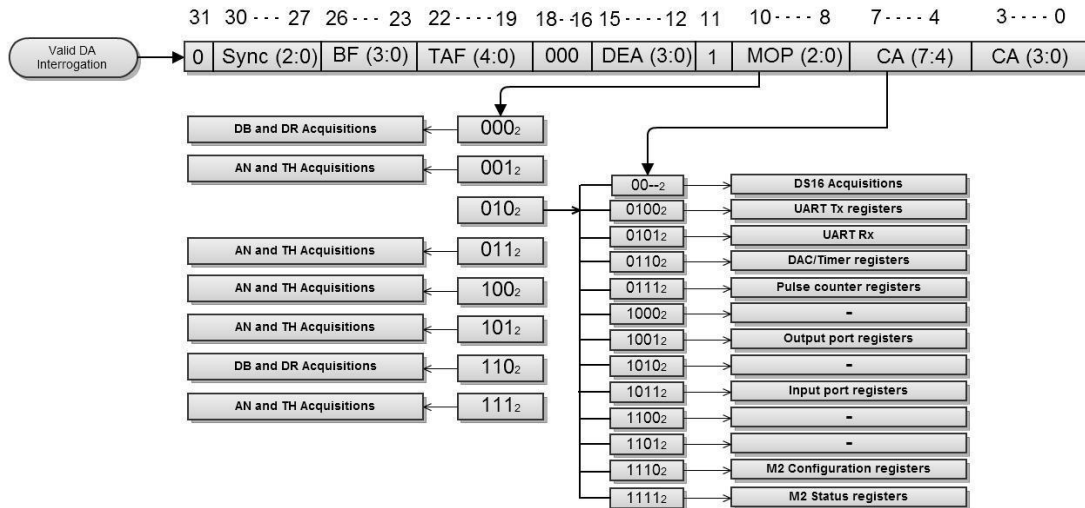


Figure 21 DA Interrogation Parsing

3.4.3.1 AN/TH Acquisition

The analogue or thermistor values are supposed to be converted into digital values. But in this implementation the ADC part is skipped since we are dealing with digital data. For the sake of this specific application AN/TH data is sent through the TCP/IP and when there is an AN/TH interrogation then the received data is analyzed and send back as response. But first when a DA interrogation is arrived it is checked against the MOP field to decide that either the interrogation is an AN or TH acquisition. Then appropriate data is expected for the channel number specified in the interrogation. Interrogation parsing is done according to the diagram below:

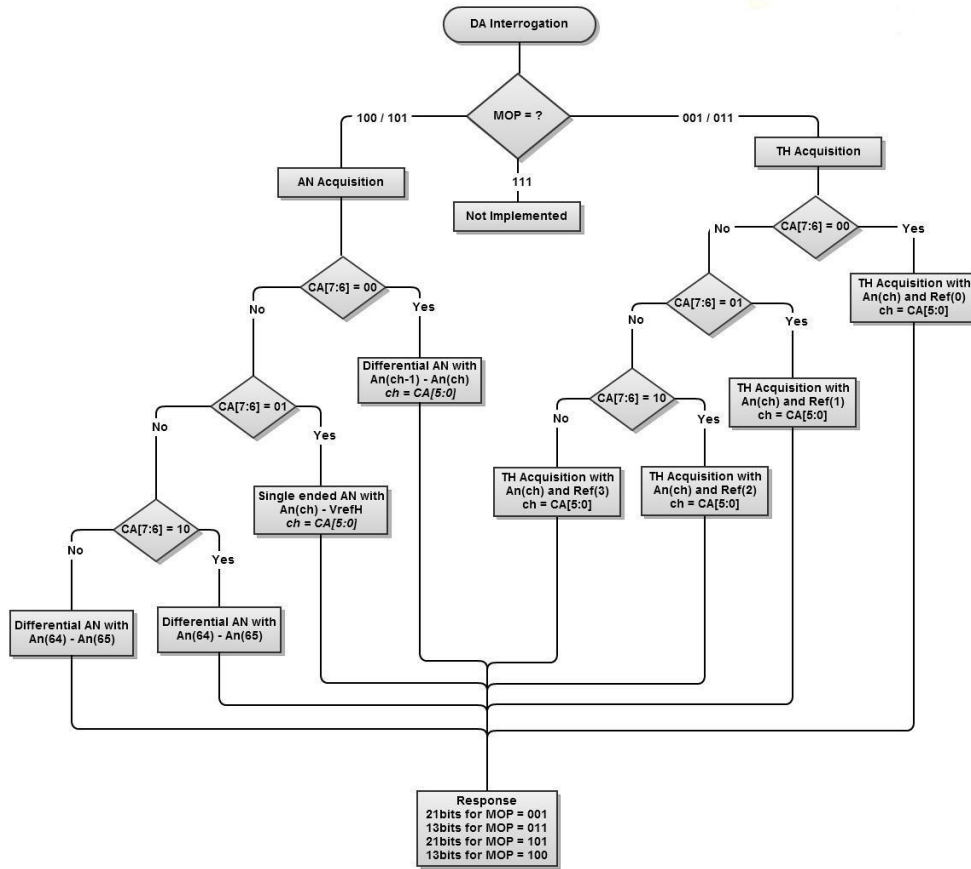


Figure 22 AN/TH Acquisition

3.4.3.2 DB/DR Acquisition

Digital Bilevel (DB) / Digital Relay(DR) acquisition is similar to AN/TH acquisition. When MOP is '110' then the DIGDB field of BCP and Acquisition Config register is used for DB acquisition. When DIGDB = 0 then DB acquisition works in Digital Relay Matrix (DRM) mode. In DRM mode only first 16 channels are used. For DIGDB = 1 CompIn input is used to receive 8 bit serial data. CompIn input is the input port of an IO group that is configured as Mx16. Only group 8 can be configured as Mx16.

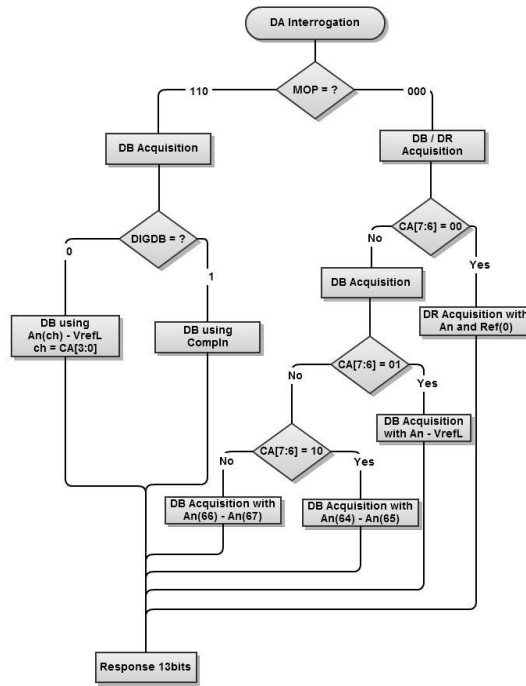


Figure 23 DB/DR Acquisition

3.4.3.3 DS16 Acquisition

DS16 is serial data transfer operation where the external circuit sends data serially to the configured IO groups of the M2 ASIC using handshaking method. But it is simulated in a different way since data is sent or received over TCP/IP according to the data format specified in table 1.

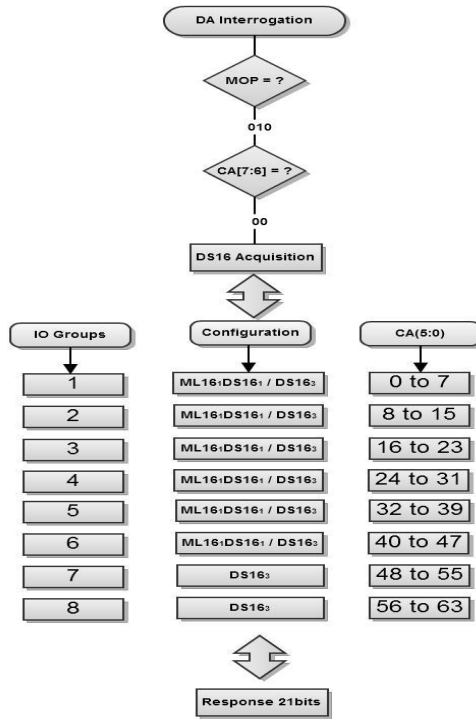


Figure 24 DS16 Acquisition

3.4.3.4 Input port registers

PortIn register is readable while the MOP field of a DA interrogation is '010' and the 8 bit channel address field contains '10110000'.

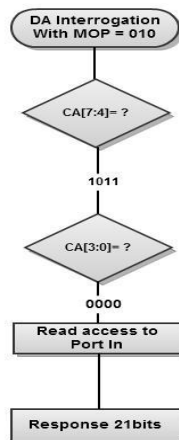


Figure 25 Input port registers

After reading the value of that register the response is generated accordingly and sent back to the OBDH bus.

3.4.3.5 Output port registers read operation

There are two output port registers: Port out 1234 and Port out 5678. Both can be read by DA interrogation with MOP > 0 and no IO groups configured to either ML16DS16 or ML16 or UART.

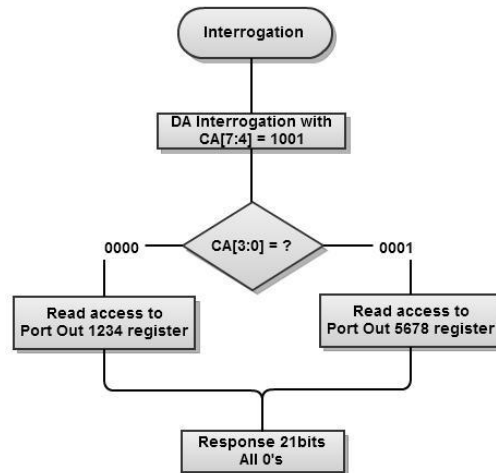


Figure 26 Output port registers Read operation

Generated response for both cases is 21bits.

4 Simulation Model Portability

4.1 Background

The European Space Agency has been working for space simulation development for a number of years. They are developing simulations for a variety of applications and this involves analysis, engineering operations preparation and training. There are different departments working on simulation and they may use different platforms and different computer languages as well. It is difficult to adapt the same model in different platforms and the communication among different platforms is also difficult. To address these issues a Simulation Model Portability study was performed and a standard was defined to ease the portability and reuse of simulation models in different environments. The SMP1 Handbook [8] was published to describe the main SMP1 scope and SMP usage. A software implementation of SMP1 was introduced which is called Simulation Model Interface (SMI).

SMP1 was achieved via four objectives:

- Minimize interaction between models and environment
- Standardization of inter model interfaces
- Simplify intra model interfaces
- Models are simple enough for other developers

SMP1 was successfully applied to several space projects. But at the same time some limitations of SMP1/SMI has been noticed:

- Inter-model communication was built on the basis of dynamic invocation. That means models could not communicate directly but by the help of the environment.
- Did not support object oriented design.
- Publication calls were done manually which is error prone.
- Scheduling mechanism was primitive.
- Did not support additional metadata for models.
- Could not properly handle initial values of models
- Does not provide access or change to simulation state
- Does not support dynamic simulation.

To overcome these limitations SMP2 was introduced and it came up with few objectives:

- Portability of models among different platforms
- Interoperability and reusability of models

SMP2 has several advantages over SMP1 which made SMP2 to be more acceptable for space project simulation:

- Developed models are platform independent
- Portability of models is now easier
- Models are more reusable as there is less dependency among models

- Model integration is simplified
- Support Object Oriented technologies
- Support for metadata
- Support dynamic simulation

4.2 SMP2 Architecture

SMP2 consists of models and a simulation environment. It defines how the models communicate with each other and how the models communicate with the simulation environment. SMP2 defines the interfaces for the inter model communication and it also defines the interfaces so that the models can communicate with the simulation environment in a controlled way. In this way models are not dependent upon each other. It ensures models are portable and can be reused in various environments.

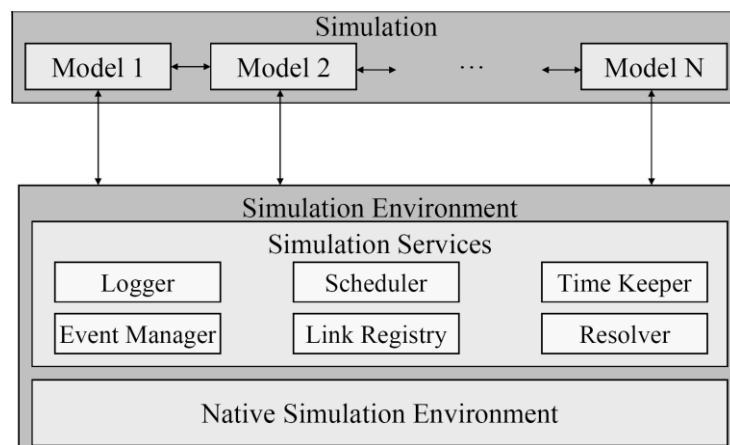


Figure 27 SMP2 architecture

Simulation environment provides services necessary for simulation. There are mandatory services such as Logger, Scheduler, Timekeeper and Event manager. On the other hand there may be optional services like Link Registry or Resolver and user defined services are also possible.

4.2.1 Simulation Environment

A simulation environment contains a native simulation environment to make it SMP2 compliant and simulation services [9]. Simulation services can be of two types, mandatory and optional services. There are four mandatory services: Logger, Scheduler, Time Keeper and

Event Manager. Optional services are resolver and link registry. User defined services can also be added.

Logger: This service is used to log event, warning and error messages. Both models and services use logger to log messages.

Scheduler: Scheduler calls the entry points based on cyclic or timed events.

Time Keeper: Time keeper service provides four types of time, a relative simulation time, an absolute epoch time, a relative mission time and Zulu time which represents current computer time.

Event manager: Event manager service provides mechanism for handling global events. Events can be registered and broadcasted. User defined event type is also supported.

Resolver: Resolver provides reference to the other models in the simulation.

Link Registry: If a model instance is deleted then Link registry service notifies other models holding reference to this model.

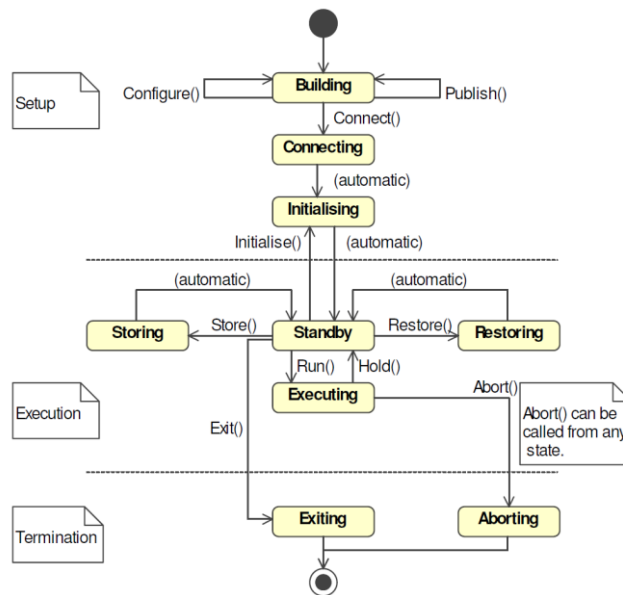


Figure 28 State Diagram of Simulation Environment [6]

After the creation of simulation services the simulation environment automatically enters into building state. In this state it creates model instances and builds model hierarchies, asks to publish their fields, operations and properties.

In the end of building state Connect() method of every model in the model hierarchy is called to enter into connecting state. ISimulator interface is passed so that every model can use the simulation services.

After connecting state the simulation environment automatically enters into initializing state but from standby state initialize() method call is required to enter in this state. In this state all the entry points are called to guarantee that all the models have their initial values and are properly linked together.

Standby state is automatically achieved after initializing state or from storing and restoring state and using a hold() method from executing state. In this state simulation time does not progress even though the Zulu time still progresses.

Executing state can be entered from the standby state using the run() method. In this state simulation time progresses as well as the other time kinds registered with entry points.

Storing state can be entered using the Store() method from Standby state. The current state of the simulation environment is saved during this state.

This state is entered from Standby state using Restore() method. The state of the simulation environment can be restored from the storage.

To properly terminate a running simulation the Exit() method is called from the Standby state.

To perform a abnormal simulation shutdown Abort() method is called from any other state. After aborting the simulation environment is in an undefined state.

4.2.2 Operational phases

There are three phases of SMP2 simulation operation. In the set up phase model instances are created and configured by the simulation environment. Models publish their states to the simulation environment and then models are connected to the simulation services and other models.

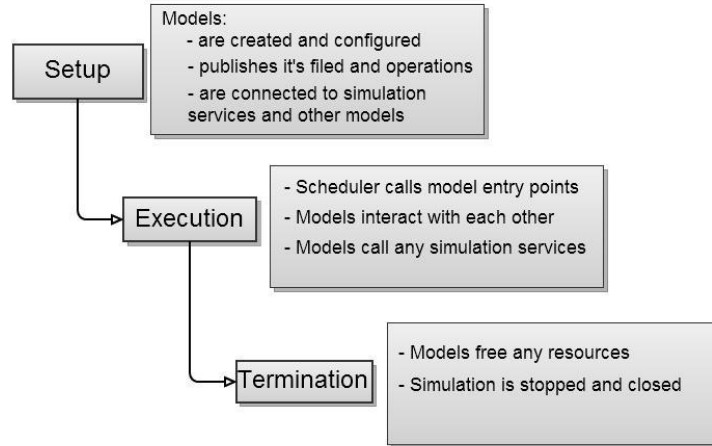


Figure 29 SMP2 operational phases

In the execution phase the models are scheduled by the scheduler and they start interacting with each other. In the termination phase models may free all the occupied resources and then simulation is stopped.

4.3 SMP2 Mechanism

The simulation environment has two containers: Model container and Service container. Model Container contains root models. A root model is a model that does not have a parent in the model hierarchy or model tree.

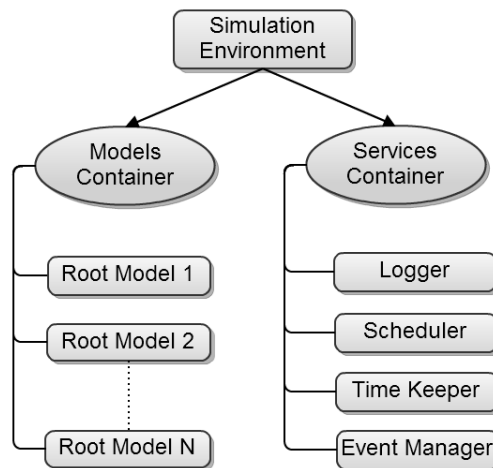


Figure 30 SMP2 Mechanism

A root model may contain other models in the model hierarchy. Service container contains the services. Services are also a kind of root models as they don't have any parent.

4.4 Inter model communication

Inter model communication depends on the design approach followed by the developer. SMP2 supports three model interaction approaches:

1. Interface based design
2. Event based design
3. Dataflow based design

4.4.1 Interface based design

An interface is a set of public features, such as fields and operations. One model provides interface so that outer world can communicate with the model through provided interfaces. The model that provides interface is called the provider.

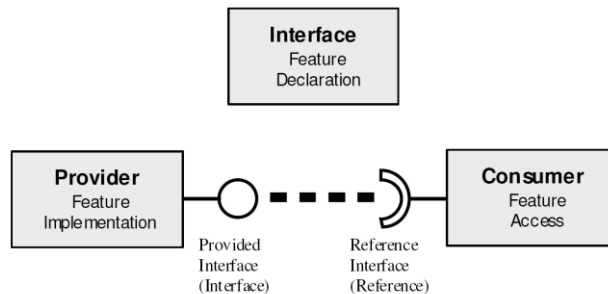


Figure 31 Interface based communication

Another model consumes the information provided by the provider. It is called the consumer. Consumer has to implement the features provided by the interface.

4.4.2 Event based design

In an event based design one model will trigger an event and other models which are dependent on that model will be notified. In this case the model (Provider) that triggers an event acts as an event source. On the other hand the models (Consumer) which consume event acts as event sink.

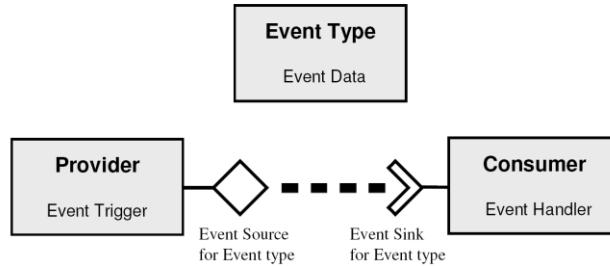


Figure 32 Event based design

Consumer defines event handler to handle events triggered by the provider. To distinguish various kinds of events, every event is assigned an event type.

4.4.3 Dataflow based design

In a data flow based design inter model communication is done based on data dependency. The model that provides data is called the source and the model that consumes data is called the target.

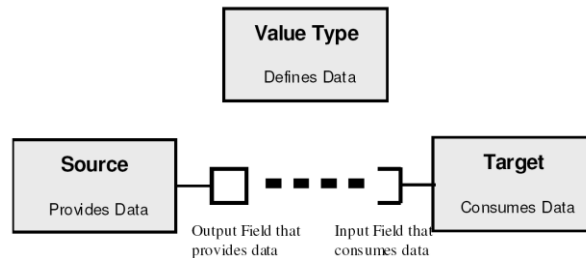


Figure 33 Dataflow based design

Data transfer is normally done by other component(s) which reads the data from the source's output field and store it into the input field of the target model.

4.5 SMP2 adaptation of M2 ASIC Model

In this section it is assumed that the models will be developed using standard C++ and it is also assumed that an SMP2 compliant tool is used for development. The first step of SMP2 development is to describe the intended models using Simulation Model Definition Language (SMDL) which is also called SMP2 Metamodel [10][11]. This is not a mandatory step. Then by using the selected tool one can generate catalogues for the models so that these catalogues can be validated against SMP2 rules. Catalogue is an xml document that contains namespaces as a primary ordering mechanism and namespace can contain types such as structure, class and interface. Catalogues can also be made by hand using a catalogue editor. The next step is to

generate assemblies and schedules. An assembly contains model instances and the links among them and schedule defines how the model instances of an assembly are scheduled. The catalogue, assembly, schedule files should be validated by appropriate tools. Then a code generator is used to make model code from catalogue, assembly and schedule [12][13]. It is important to adapt the current simulator so that it can accept SMP2 models. So despite of starting from the beginning it is advised to start the implementation directly. An SMP2 compliant Model Development Kit (MDK) [14] can be used as a starting point. The MDK contains the necessary C++ source files that describe SMP2 rules. SMP2 supports interface hierarchy and component based infrastructure [15] [16]. Structural dependency is based upon interface hierarchy.

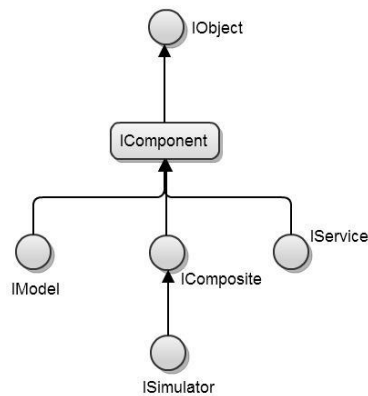


Figure 34 SMP Component

Every component in SMP is derived from IObject. An object is the base entity that must be derived by every component. IObject interface has the following structure:

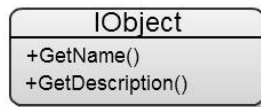


Figure 35 IObject interface

GetName() method returns the name and GetDescription() methods returns description of the object. Most of the SMP elements are components and they derive IComponent interface. Components can be a model, service or simulator.

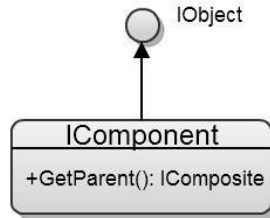


Figure 36 IComponent interface

IComponent interface defines GetParent() method that returns the parent of the component. All models implement IModel interface. Since models communicate with the simulation environment IModel interface has dependencies on IPublication and ISimulator interfaces.

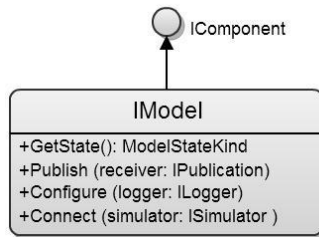


Figure 37 IModel interface

GetState() method returns current state of a model. Publish() method requests the model to publish its field, properties or operation against the publication receiver. Configure() method requests the model to perform any custom configuration. Connect() method is called to connect the model to the simulation environment.

To convert the M2 model to an SMP2 compliant model M2ASIC class (in the implementation) has to implement IModel interface. The next step is publishing model data to the simulation environment. This is done by the function Publish() described above. Publish() method takes a parameter Smp::IPublication *receiver, where IPublication interface provides a PublishField() method that is used to publish the field data to the simulation environment.

Then services need to be prepared for the models. When calling the Connect() method by the simulation environment a parameter is passed of ISimulator type. Using this parameter a model can use any service provided by the simulation environment. Then the model should be added to the scheduler. But before that an entry point should be created for that model. An entry point is a method that does not take any parameter and it does not return any value. Then this entry point should be scheduled by the scheduler. To register the model with a global event the event manager service can be used. An event can be subscribed or unsubscribed to the event manager.

4.6 SMP2 adaptation for the current SMU simulator

The SMU core simulator of Ruag space has three components: TSIM, SimKernel and SimModel [17]. TSIM is a processor emulator for ERC32/LEON processor. SimKernel contains the start up routines, simulator infrastructure and services such as logger, scheduler etc. SimModel contains the model for different IO boards, ASICs and buses.

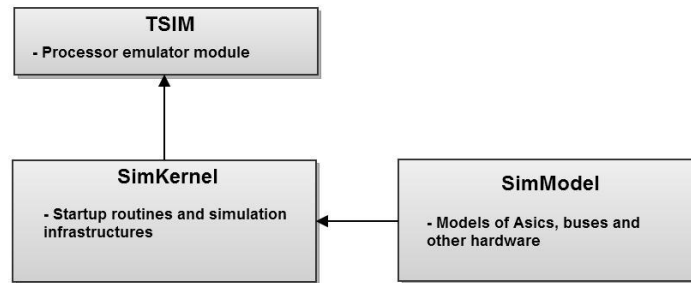


Figure 38 SMU core simulator

To make the current simulator SMP2 compliant the first step is to make the simulation environment. As the simulation environment contains services, it is necessary to take out the services described in figure 28 from SimKernel and TSIM to a new entity called simulation environment. TSIM contains the time keeper and the event manager services, where the SimKernel contains the rest. But TSIM cannot be changed as the source code for TSIM is not available. This problem can be solved by making a new component that will wrap the current TSIM and will work as an SMP2 model. The services from the SimKernel will be taken out and then put it in the simulation environment. TSIM will share its services with the simulation environment which means that calls to the services in simulation environment that belong to TSIM will be redirected to TSIM. There are some trade offs for this solution. First, the simulation environment is not independent which violates SMP2 constraints. Second, there may be unexpected delay due to service calls that belongs to TSIM. The IO boards in the SimModel can be converted separately to SMP2 models or the SimModel can be wrapped as a single model to minimize the effort.

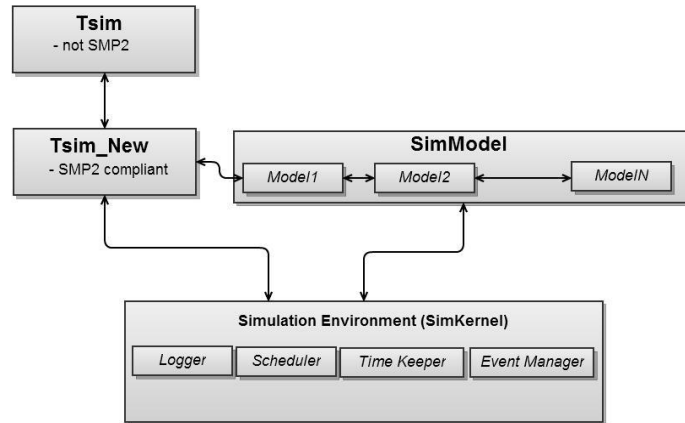


Figure 39 SMP2 compliant SMU core simulator

In both cases the simulator will be able to accept new SMP2 compliant models. There is another solution to wrap the whole simulator as one model and use a simulation environment that contains all the services. Since there will be two time keepers and two event managers, synchronization among TSIM and simulation environment can be a problem.

5 Tests and Results

For an interrogation M2 generates correct response and it does some IO operations if it requires. First a set of interrogations is written in a batch file. Then the file is loaded in the simulator. These interrogations are then sent to OBDH RT by the OBDH CT. OBDH CT sends one interrogation at a time and waits for the response. The M2 does some operations according to the interrogation and generates response that is given back to the OBDH CT.

5.1 Test Environment

The M2 ASIC model is built and tested in a small environment as described in Figure 3. After creating M2 it was connected to the OBDH CT and to TCP/IP link. First task is to start the simulator. Then the following three windows will open and among them the Sim Interface window is used to load a batch file that contains interrogation list.

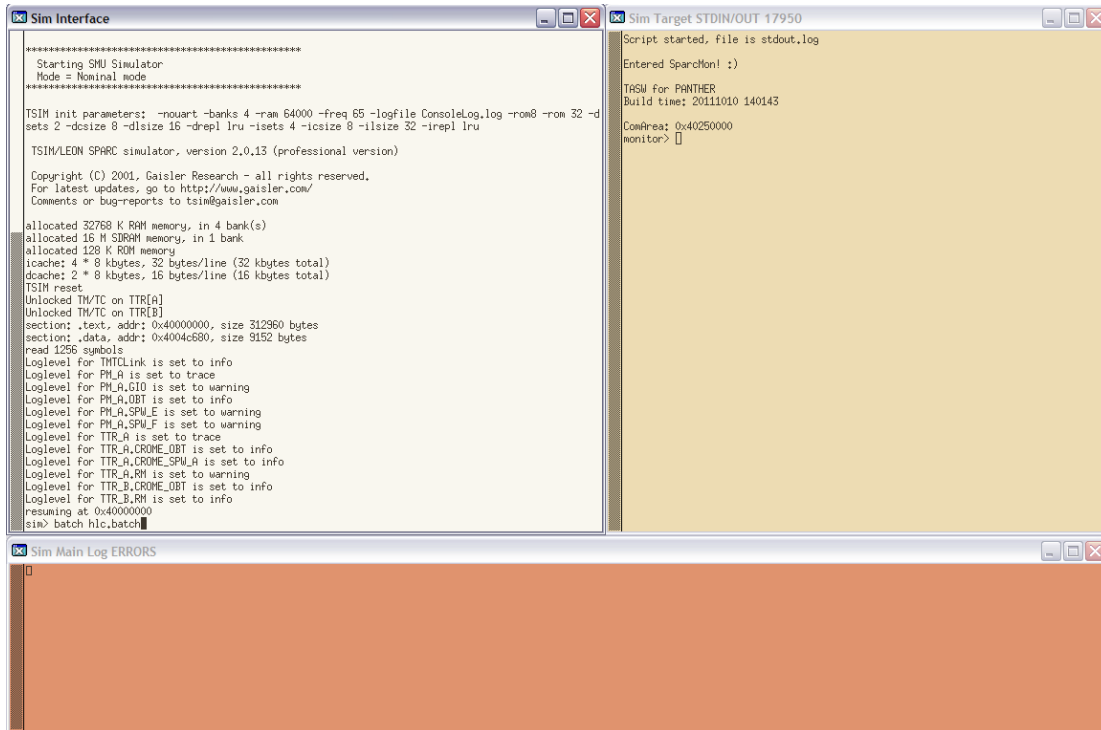


Figure 40 SMU Core Simulator

Outputs are also generated in the Sim Interface window for the purpose of the report writing though there is a log file that keeps track of operations. To start the IO operation the external application is needed that can send and receive data through TCP/IP. A simple interface is used for this application. It is written in C language. Here is a screenshot of the external application while sending DS16 data to the TCP/IP port.

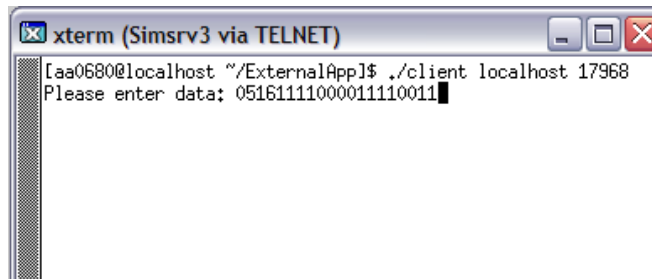


Figure 41 External Application

The interrogations are listed in appendix B and the test results are provided in appendix C. Here is an example for a list of interrogations in a batch file.

```

write 40180000 0x000FE100 // Both IOGroup1 and IOGroup2 are configured as
                          disabled
write 40180004 0x00080AE1 // Read IOGrp12 Config register

```

```

write 40180008 0x000FE110 // IOGroup1 is configured as port and IOGroup2
                           disabled

write 4018000C 0x00080AE1 // Read IOGrp12 Config register

write 40180010 0x000FE200 // Both IOGroup3 and IOGroup4 are configured as
                           disabled

write 40180014 0x00080AE2 // Read IOGrp34 Config register

```

Every line starts with the 'write' command, then the memory location where the interrogation will be written and then the interrogation is written in hexadecimal format. Interrogations are written as a block of memory and there must be even number of interrogations.

The results are shown in the same window for convenience and look like the following diagram.

```

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 0

Read Configuration registers
IOGrp12Config: 0
responseR:0

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 10

Read Configuration registers
IOGrp12Config: 10
responseR:10

Valid ML interrogation
Configure M2Config registers
--> IOGroup34 Config register: 0

Read Configuration registers
IOGrp34Config: 0
responseR:0

```

Figure 42 Result output for interrogation list

6 Discussion

The M2 ASIC software model was created and tested in the source code of Ruag's SMU simulator. It generates correct output for all the interrogations listed in Appendix A. Only the M2 ASIC is tested in this environment. There was an idea to test the M2 model in such a way that the actual hardware is tested. To create such real test environment requires a dedicated person from Ruag's side for a certain amount of time to train and help loading the Model in the simulator. So the plan was cancelled for that time being and left for future development.

The customers of Ruag space are interested to have SMP2 compliant simulator due to its advantages. So the thesis involves studies of SMP2. After the successful studies it is turned out that Ruag simulator can be made in such a way that it can accept SMP2 models. But to investigate more one should start using a tool for SMP2 development. We studied and found a tool called SimSat4 MIE and it costs €4000. But it is a preliminary cost and costs will increase much more in the future if the whole simulator is made SMP2 compliant. For the first step a small project of SMP2 development is proposed to get hands on experience on SMP2 tools and the SMP2 protocol. Then the next step is to convert the existing simulator SMP2 compliant.

7 Conclusion

7.1 M2 Simulator Model

The M2 Simulator model does not represent a complete model of the M2 ASIC as it does not support some functions described in section 2.1. In the M2 ASIC IO operation is serial i.e. data is sent or received one bit at a time so data synchronization is required. For the software model data synchronization is not required at bit level as data transfer through TCP/IP port contains 8 or 16 bit of information. Results are satisfactory for the environment it is tested in. It could be interesting to test the model in a real test environment discussed in section 6. Then it could be observed how the M2 model behaves in real environment.

7.2 Simulation Modeling Portability

SMP2 compliancy for M2 ASIC is achievable using SMP2 MDK only. The MDK comes up with the tool and it's not for free. A good tool for SMP2 model development is SIMSAT4 [18] which can generate and validate catalogues, assemblies and schedules. It has a code generator that creates wrapper code for the models. But creating an SMP2 model will not make any difference as the current system is not ready to accept SMP2 models. Now the challenge is to change the current SMU simulator such that it can accept SMP2 compliant models. But TSIM contains some services i.e. time keeper and event manager which belongs to the simulation environment. It is hard to separate these services from TSIM to simulation environment. On the other hand there is timing constraint as there are some operations which need to be done with no delays. It is unknown that how much delay will be introduced if simulation environment plays the role of TSIM. Other processor emulators should be considered for fully compliant SMP2 simulator.

8 References

- [1] TSIM ERC32/LEON simulator
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=38&Itemid=56
- [2] ERC32 <http://en.wikipedia.org/wiki/ERC32>
- [3] LEON
http://www.gaisler.com/cms/index.php?option=com_content&task=section&id=4&Itemid=33
- [4] M2 ASIC Specification, Document ID: P-ASIC-SPC-00052-SE, Issue no 12
- [5] Data bus Interface System Standard, Document ID: TTC.B.01, Issue no 1
- [6] SMP 2.0 Handbook, Document ID: EGOS-SIM-GEN-TN-0099, Issue no 1, Revision 2
- [7] OBDH User's manual, Document ID: P-ASIC-NOT-00100-SE, Issue no 2
- [8] Simulation Model Portability Handbook, EWP-2080, Issue 1, Revision 4
- [9] Simulation modelling platform - Volume 1: Principles and Requirements, ECSS-E-TM-40-07 Volume 1A
- [10] SMP 2.0 Metamodel, EGOS-SIM-GEN-TN-0100, Issue 1, Revision 2
- [11] Simulation modelling platform - Volume 2: Metamodel, ECSS-E-TM-40-07 Volume 2A
- [12] SMP 2.0 C++ Mapping, EGOS-SIM-GEN-TN-0102 Issue 1, Revision 2
- [13] Simulation modelling platform - Volume 4: C++ Mapping, ECSS-E-TM-40-07 Volume 4A
- [14] SMP 2.0 C++ Model Development Kit, EGOS-SIM-GEN-TN-1001, Issue 1, Revision 2
- [15] SMP 2.0 Component Model, EGOS-SIM-GEN-TN-0101, Issue 1, Revision 2
- [16] Simulation modelling platform - Volume 3: Component Model, ECSS-E-TM-40-07 Volume 3A, Issue no 1
- [17] Basic Software Validation Facility Systems Specification, P-CBSW-SPC-00003-SE, Issue no 3
- [18] Simulation modelling platform -Volume 5: SMP usage, ECSS-E-TM-40-07 Volume 5A, Issue no 1

Appendix A : Abbreviations

ACQ	Acquisition
ADC	Analog to Digital Converter
AN	Analog (channel)
ASIC	Application Specific Integrated Circuit
BCP	Broadcast Pulse
BF	Broadcast Field
CA	Channel Address
CMD	Command
COMP	Comparator
CONFIG	Configuration
CONV	Converter
CT	Central Terminal
Ctrl	Control
DA	Data Acquisition (type of OBDH interrogation)
DAC	Digital to Analog Converter
DB	Digital Bilevel
DEA	Destination Address
DR	Digital Relay
DRM	Digital Relay in Matrix configuration
DS	Digital Serial
DS16	Digital Serial 16 bits format
HLC	High Level Command
HLCM	High Level Command in Matrix configuration
I/F	Interface
IO, I/O	Input/output
LSB	Least Significant Bit
ML	Memory Load (type of OBDH interrogation)
ML16	Memory Load 16 bit serial interface
MLA	Memory Load Address
MLD	Memory Load Data
MOP	Mode of Operation
MSB	Most Significant Bit
MUX	Multiplexer
OBDH	On Board Data Handling
OCD	Output Command Driver

OO	On/Off (type of OBDH interrogation)
RDF	Response Data Field
Ref	Reference
Resp	Response
RF	Radio Frequency
RT	Remote Terminal
RTU	Remote Terminal Unit
Rx	Receive
SEL	Select
Sync	Synchronization
TAF	Terminal Address Field
TDF	Terminal Data Field
TH	Thermistor channel
Tx	Transmit
UART	Universal Asynchronous Receiver Transmitter
Z	High impedance
API	Application Programming Interface
ESA	European Space Agency
EuroSim	European Real-time Operations Simulator
HITL	Hardware-In-The-Loop
MDK	Model Development Kit
SIMSAT	Software Infrastructure for the Modeling of Satellites
SMDL	Simulation Model Definition Language
SMP	Simulation Model Portability
SMP1	Simulation Model Portability 1
SMP2	Simulation Model Portability 2
UML	Unified Modeling Language
UUID	Universally Unique Identifier
XML	Extensible Markup Language

Appendix B: Interrogation List

The following interrogations are used in section 5 for testing the M2 ASIC model.

DA Interrogation list

Operation	Interrogation
DR acquisition with An(ch) and Ref(0), ch = CA[5:0]	0000 0000 0000 1000 0000 1000 00 xx xxxx
DB acquisition with An - VrefL	0000 0000 0000 1000 0000 1000 01 xx xxxx
DB acquisition with An(64) – An(65)	0000 0000 0000 1000 0000 1000 10 xx xxxx
DB acquisition with An(66) – An(67)	0000 0000 0000 1000 0000 1000 11 xx xxxx
DB acquisition using An(ch) – VrefL, ch = CA[3:0]	0000 0000 0000 1000 0000 1110 0000 xxxx [DIGDB = 0]
DB using CompIn	0000 0000 0000 1000 0000 1110 xxxx xxxx [DIGDB = 1]
Differential AN with An(ch-1) – An(ch), ch = CA[5:0]	0000 0000 0000 1000 0000 110x 00 xx xxxx
Single ended AN with An(ch) – VrefH, ch = CA[5:0]	0000 0000 0000 1000 0000 110x 01 xx xxxx
Differential AN with An(64) – An(65)	0000 0000 0000 1000 0000 110x 10 xx xxxx
Differential AN with An(66) – An(67)	0000 0000 0000 1000 0000 110x 11 xx xxxx
TH acquisition with An(ch) and Ref(0), ch = CA[5:0]	0000 0000 0000 1000 0000 10x1 00 xx xxxx

TH acquisition with An(ch) and Ref(1), ch = CA[5:0]	0000 0000 0000 1000 0000 10x1 01 xx xxxx
TH acquisition with An(ch) and Ref(2), ch = CA[5:0]	0000 0000 0000 1000 0000 10x1 10 xx xxxx
TH acquisition with An(ch) and Ref(3), ch = CA[5:0]	0000 0000 0000 1000 0000 10x1 11 xx xxxx
DS16 Acquisition	0000 0000 0000 1000 0000 1010 00 xx xxxx
Read UART Tx registers	0000 0000 0000 1000 0000 1010 0100 xxxx
Read UART Rx registers	0000 0000 0000 1000 0000 1010 0101 xxxx
Read DAC/Timer registers	00 0 0000 1000 0000 1010 0110 xxxx

ML Interrogation list

Operation	Interrogation
ML16 Operation / Write Operation on PortOut1234Register	0000 0000 0000 1001 xxxx xxxx xxxx xxxx
ML16 Operation / Write Operation on PortOut1234Register	0000 0000 0000 1010 xxxx xxxx xxxx xxxx
ML16 Operation / Write Operation on PortOut1234Register	0000 0000 0000 1011 xxxx xxxx xxxx xxxx
ML16 Operation / Write Operation on PortOut1234Register	0000 0000 0000 1100 xxxx xxxx xxxx xxxx
ML16 Operation / Write Operation on	0000 0000 0000 1101 xxxx xxxx xxxx xxxx

PortOut5678Register	
ML16 Operation / Write Operation on PortOut5678Register	0000 0000 0000 1110 xxxx xxxx xxxx xxxx
Write Operation on UART Tx Registers	0000 0000 0000 1111 0100 xxxx xxxx xxxx
Write operation on Prolonged Settings register	0000 0000 0000 1111 0101 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 0110 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 0111 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1000 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1001 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1010 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1011 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1100 xxxx xxxx xxxx
Write Operation on DAC/Timer Registers	0000 0000 0000 1111 1101 xxxx xxxx xxxx
Write Operation on M2 configuration registers	0000 0000 0000 1111 1110 xxxx xxxx xxxx

Write Operation on M2 Configuration Registers

Operation	Interrogation
IOGrp12 Config	0000 0000 0000 1111 1110 0001 xxxx xxxx
IOGrp34 Config	0000 0000 0000 1111 1110 0010 xxxx xxxx
IOGrp56 Config	0000 0000 0000 1111 1110 0011 xxxx xxxx
IOGrp78 Config	0000 0000 0000 1111 1110 0100 xxxx xxxx
HlcLength1 Config	0000 0000 0000 1111 1110 0101 xxxx xxxx
HlcLength2 Config	0000 0000 0000 1111 1110 0110 xxxx xxxx
Latch/UART Config	0000 0000 0000 1111 1110 0111 xxxx xxxx
BCP and Acquisition Config	0000 0000 0000 1111 1110 1000 xxxx xxxx
Counter12 Config	0000 0000 0000 1111 1110 1001 xxxx xxxx
Counter34 Config	0000 0000 0000 1111 1110 1010 xxxx xxxx
DAC/Timer12 Config	0000 0000 0000 1111 1110 1011 xxxx xxxx
DAC/Timer34 Config	0000 0000 0000 1111 1110 1100 xxxx xxxx
DAC/Timer56 Config	0000 0000 0000 1111 1110 1101 xxxx xxxx
DAC/Timer78 Config	0000 0000 0000 1111 1110 1110 xxxx xxxx
Parity	0000 0000 0000 1111 1110 1111 xxxx xxxx

Read Operation on M2 Configuration Registers

Operation	Interrogation
IOGrp12 Config	0000 0000 0000 1000 0000 1010 1110 0001
IOGrp34 Config	0000 0000 0000 1000 0000 1010 1110 0010

IOGrp56 Config	0000 0000 0000 1000 0000 1010 1110 0011
IOGrp78 Config	0000 0000 0000 1000 0000 1010 1110 0100
HlcLength1 Config	0000 0000 0000 1000 0000 1010 1110 0101
HlcLength2 Config	0000 0000 0000 1000 0000 1010 1110 0110
Latch/UART Config	0000 0000 0000 1000 0000 1010 1110 0111
BCP and Acquisition Config	0000 0000 0000 1000 0000 1010 1110 1000
Counter12 Config	0000 0000 0000 1000 0000 1010 1110 1001
Counter34 Config	0000 0000 0000 1000 0000 1010 1110 1010
DAC/Timer12 Config	0000 0000 0000 1000 0000 1010 1110 1011
DAC/Timer34 Config	0000 0000 0000 1000 0000 1010 1110 1100
DAC/Timer56 Config	0000 0000 0000 1000 0000 1010 1110 1101
DAC/Timer78 Config	0000 0000 0000 1000 0000 1010 1110 1110
Parity	0000 0000 0000 1000 0000 1010 1110 1111

OO Interrogation list

Operation	Interrogation
HLC command using HlcLength1 register	0000 0000 0000 1000 0000 0011 xxxx xxxx
HLC command using HlcLength2 register	0000 0000 0000 1000 0000 0100 xxxx xxxx

Appendix C: Test results

Testing ML interrogations

Interrogations are written as a block of memory and there must be even number of interrogations. Below there is a list of ML interrogations related to register configuration. Each interrogation is commented with brief description.

```
write 40180000 0x000FE100 // Both IOGroup1 and IOGroup2 are configured as
disabled
write 40180004 0x00080AE1 // Read IOGrp12 Config register
write 40180008 0x000FE110 // IOGroup1 is configured as port and IOGroup2
disabled
write 4018000C 0x00080AE1 // Read IOGrp12 Config register
write 40180010 0x000FE200 // Both IOGroup3 and IOGroup4 are configured as
disabled
write 40180014 0x00080AE2 // Read IOGrp34 Config register
write 40180018 0x000FE210 // IOGroup3 is configured as port and IOGroup4
disabled
write 4018001C 0x00080AE2 // Read IOGrp34 Config register
write 40180020 0x000FE300 // Both IOGroup5 and IOGroup6 are configured as
disabled
write 40180024 0x00080AE3 // Read IOGrp56 Config register
write 40180028 0x000FE310 // IOGroup5 is configured as port and IOGroup6
disabled
write 4018002C 0x00080AE3 // Read IOGrp56 Config register
write 40180030 0x000FE400 // Both IOGroup7 and IOGroup8 are configured as
disabled
write 40180034 0x00080AE4 // Read IOGrp78 Config register
write 40180038 0x000FE410 // IOGroup7 is configured as port and IOGroup8
disabled
write 4018003C 0x00080AE4 // Read IOGrp78 Config register
write 40180040 0x000FE554 // HlcLength1 Configuration, N2 = 5 and N1 = 4
```

```

write 40180044 0x00080AE5 // Read HlcLength1 register
write 40180048 0x000FE60A // HlcLength2 Configuration, N2 = 0 and N1 = A
write 4018004C 0x00080AE6 // Read HlcLength2 register
write 40180050 0x000FE808 // BCP and Acquisition Config register

write 40180054 0x00080AE8 // Read BCP and Acquisition Config register

write 40180058 0x000FE701 // Latch/UART Config register

write 4018005C 0x00080AE7 // Read Latch/UART Config register
write 40180060 0x000FE990 // Counter12 Config
write 40180064 0x00080AE9 // Read Counter12 Config register
write 40180068 0x000FEA09 // Counter34 Config
write 4018006C 0x00080AEA // Read Counter34 Config register
write 40180070 0x000FEB11 // DAC/Timer12 Config

write 40180074 0x00080AEB // Read DAC/Timer12 Config register

write 40180078 0x000FEC11 // DAC/Timer34 Config
write 4018007C 0x00080AEC // Read DAC/Timer34 Config register
write 40180080 0x000FED11 // DAC/Timer56 Config
write 40180084 0x00080AED // Read DAC/Timer56 Config register
write 40180088 0x000FEE11 // DAC/Timer78 Config
write 4018008C 0x00080AEE // Read DAC/Timer78 Config register
write 40180090 0x000FEF11 // Parity register configuration

```

The output for the given script looks like this:

```

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 0

Read Configuration registers
IOGrp12Config: 0
responseR:0

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 10

Read Configuration registers
IOGrp12Config: 10
responseR:10

Valid ML interrogation
Configure M2Config registers
--> IOGroup34 Config register: 0

Read Configuration registers
IOGrp34Config: 0
responseR:0

Valid ML interrogation
Configure M2Config registers
--> IOGroup34 Config register: 10

Read Configuration registers
IOGrp34Config: 10
responseR:10

Valid ML interrogation
Configure M2Config registers
--> IOGroup56 Config register: 0

Read Configuration registers
IOGrp56Config: 0
responseR:0

Valid ML interrogation
Configure M2Config registers
--> IOGroup56 Config register: 10

Read Configuration registers
IOGrp56Config: 10
responseR:10

Valid ML interrogation
Configure M2Config registers
--> IOGroup78 Config register: 0

Read Configuration registers
IOGrp78Config: 0
responseR:0

Valid ML interrogation
Configure M2Config registers
--> IOGroup78 Config register: 10

Read Configuration registers
IOGrp78Config: 10
responseR:10

```

```

Valid ML interrogation
Configure M2Config registers
--> HLC Length 1 config register: 54

Read Configuration registers
HlcLength1Config: 54
responseR:54

Valid ML interrogation
Configure M2Config registers
--> HLC Length 2 config register: a

Read Configuration registers
HlcLength2Config: a
responseR:a

Valid ML interrogation
Configure M2Config registers
--> BCP and Acquisition config register: 8

Read Configuration registers
BCPAndAcqConfig: 8
responseR:8

Valid ML interrogation
Configure M2Config registers
--> Latch/UART config register: 1

Read Configuration registers
LatchOrUARTConfig: 1
responseR:1

Valid ML interrogation
Configure M2Config registers
--> Counter12 Config Register: 90

Read Configuration registers
Counter12Config: 90
responseR:90

Valid ML interrogation
Configure M2Config registers
--> Counter34 Config Register: 9

Read Configuration registers
Counter34Config: 9
responseR:9

Valid ML interrogation
Configure M2Config registers
--> DAC/Timer12 Config Register: 11

Read Configuration registers
DACOrTimer12Config: 11
responseR:11

```

```

Valid ML interrogation
Configure M2Config registers
--> DAC/Timer34 Config Register: 11

Read Configuration registers
DACOrTimer34Config: 11
responseR:11

Valid ML interrogation
Configure M2Config registers
--> DAC/Timer56 Config Register: 11

Read Configuration registers
DACOrTimer56Config: 11
responseR:11

Valid ML interrogation
Configure M2Config registers
--> DAC/Timer78 Config Register: 11

Read Configuration registers
DACOrTimer78Config: 11
responseR:11

Valid ML interrogation
Configure M2Config registers
--> Config Parity Register: 11

Read Configuration registers
Parity: 11
responseR:11

```

Figure 43 Ouput for ML interrogations

Testing DA interrogations

Tests for different types of DA interrogations are given below.

Testing AN acquisition

The following test contains analogue channel (AN) acquisitions. In this case the channel values are set by the external application. The external application sends data to the corresponding channel and these are user defined. In this case channel 29, 30, 64, 65, 66, 67 gets 1, 1, 1, 0, 0, 0 respectively.

```
write 40180000 0x00080C1E // Differential AN with An(29)-An(30)
write 40180004 0x00080D1E // Differential AN with An(29)-An(30)
write 40180008 0x00080C5E // Single Ended AN with An(30)-VrefH
write 4018000C 0x00080D5E // Single Ended AN with An(30)-VrefH
write 40180010 0x00080C9E // Differential An with An(64)-An(65)
write 40180014 0x00080D9E // Differential An with An(64)-An(65)
write 40180018 0x00080CDE // Differential An with An(66)-An(67)
write 4018001C 0x00080DDE // Differential An with An(66)-An(67)
```

Test results for the AN acquisition are given below:

```
Differential AN with [An(29) - An(30)]
An[29]: 1
An[30]: 1
responseR: 0

Differential AN with [An(29) - An(30)]
An[29]: 1
An[30]: 1
responseR: 0

Single ended AN with [An(30) - VrefH]
responseR: 1

Single ended AN with [An(30) - VrefH]
responseR: 1

Differential AN with [An(64) - An(65)]
responseR: 1

Differential AN with [An(64) - An(65)]
responseR: 1

Differential AN with [An(66) - An(67)]
responseR: 0

Differential AN with [An(66) - An(67)]
responseR: 0
```

Figure 44 AN interrogation

Testing TH acquisition

```
write 40180020 0x0008091E    // TH Acquisition with An(30) and Rref(0)
write 40180024 0x00080B1E    // TH Acquisition with An(30) and Rref(0)
write 40180028 0x0008095E    // TH Acquisition with An(30) and Rref(1)
write 4018002C 0x00080B5E    // TH Acquisition with An(30) and Rref(1)
write 40180030 0x0008099E    // TH Acquisition with An(30) and Rref(2)
write 40180034 0x00080B9E    // TH Acquisition with An(30) and Rref(2)
write 40180038 0x000809DE    // TH Acquisition with An(30) and Rref(3)
write 4018003C 0x00080BDE    // TH Acquisition with An(30) and Rref(3)
```

Results of TH acquisition look like the following:

```
TH acquisition with An(30) and Rref(0) where ch = CA[5:0]
An[30]: 1
Rref[0]: 0
responseR: 1

TH acquisition with An(30) and Rref(0) where ch = CA[5:0]
An[30]: 1
Rref[0]: 0
responseR: 1

TH acquisition with An(30) and Rref(1) where ch = CA[5:0]
An[30]: 1
Rref[1]: 0
responseR: 1

TH acquisition with An(30) and Rref(1) where ch = CA[5:0]
An[30]: 1
Rref[1]: 0
responseR: 1

TH acquisition with An(30) and Rref(2) where ch = CA[5:0]
An[30]: 1
Rref[2]: 0
responseR: 1

TH acquisition with An(30) and Rref(2) where ch = CA[5:0]
An[30]: 1
Rref[2]: 0
responseR: 1

TH acquisition with An(30) and Rref(3) where ch = CA[5:0]
An[30]: 1
Rref[3]: 0
responseR: 1

TH acquisition with An(30) and Rref(3) where ch = CA[5:0]
An[30]: 1
Rref[3]: 0
responseR: 1
```

Figure 45 TH interrogation Results

Testing DB/DR acquisitions

Here is a set of interrogations for DB/DR acquisition and the test results as well.

```
write 40180040 0x0008081E    // DR acquisition with An(30) and Ref(0)
```

```

write 40180044 0x0008085E // DB with An(30)-VrefL
write 40180048 0x0008089E // DB with An(64)-An(65)
write 4018004C 0x000808DE // DB with An(66)-An(67)
write 40180050 0x000FE800 // Set DIGDB = 0
write 40180054 0x00080E0D // DB with DIGDB = 0
write 40180058 0x00080E0E // DB with DIGDB = 0
write 4018005C 0x000FE808 // set DIGDB = 1
write 40180060 0x000FE409 // Configure IO Group8 as Mx16
write 40180064 0x00080EDE // DB with DIGDB = 1

```

<pre> DR with An(30) and Ref(0) --> AN[30] = 1 --> AN[31] = 0 --> AN[16] = 0 --> AN[17] = 0 --> AN[18] = 1 --> AN[19] = 1 --> AN[20] = 0 --> AN[21] = 0 Response DR : 00110001 DB with An(30) - VrefL --> AN[30] = 1 --> AN[31] = 0 --> AN[16] = 0 --> AN[17] = 0 --> AN[18] = 1 --> AN[19] = 1 --> AN[20] = 0 --> AN[21] = 0 Response DR : 00110001 DB with An(64) - AN(65) response : 00000001 DB with An(66) - AN(67) responseR:0 </pre>	<pre> Valid ML interrogation Configure M2Config registers --> BCP and Acquisition config register: 0 DB on channel :d --> AN[d] = 1 --> AN[e] = 1 --> AN[f] = 0 --> AN[0] = 1 --> AN[1] = 0 --> AN[2] = 1 --> AN[3] = 1 --> AN[4] = 1 DIGDB=0 responseR:eb DB on channel :e --> AN[e] = 1 --> AN[f] = 0 --> AN[0] = 1 --> AN[1] = 0 --> AN[2] = 1 --> AN[3] = 1 --> AN[4] = 1 --> AN[5] = 1 DIGDB=0 responseR:f5 Valid ML interrogation Configure M2Config registers --> BCP and Acquisition config register: 8 Valid ML interrogation Configure M2Config registers --> IOGroup78 Config register: 9 CompIn DIGDB=1 responseR:ab </pre>
--	---

Figure 46 DB/DR Interrogation Results

DS16 acquisition testing

Interrogations and results for DS16 testing are given below:

```

write 40180068 0x000FE140 // Configure IO Group1 as ML161DS161
write 4018006C 0x00080A03 // DS16 Acquisition on IO Group1
write 40180070 0x000FE160 // Configure IO Group1 as DS16
write 40180074 0x00080A03 // DS16 Acquisition on IO Group1

```

```

write 40180078 0x000FE460 // Configure IO Group7 as DS16
write 4018007C 0x00080A30 // DS16 Acquisition on IO Group7

```

```

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 40

DS 16 acquisition
IOGrp1 ... DS16: f0f3

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 60

DS 16 acquisition
IOGrp1 ... DS16: f0f3

Valid ML interrogation
Configure M2Config registers
--> IOGroup78 Config register: 60

DS 16 acquisition
IOGrp7 ... DS16: f0f3

```

Figure 47 DS16 acquisition

Testing OO interrogations

OO interrogations are listed below with brief description.

```

write 40180000 0x000FE554 // HlcLength1 Configuration, N2 = 5 and N1 = 4
write 40180004 0x000FE645 // HlcLength2 Configuration, N2 = 5 and N1 = 4
write 40180008 0x000FE3C0 // Set IO Group5 as HlcSer
write 4018000C 0x000803FF // HLC command using HlcLength1 register
write 40180010 0x000FE30C // Set IO Group5 as HlcSer
write 40180014 0x000803FF // HLC command using HlcLength1 register
write 40180018 0x000FE4C0 // Set IO Group7 as HlcSer
write 4018001C 0x000804FF // HLC command using HlcLength2 register
write 40180020 0x000FE40C // Set IO Group8 as HlcSer
write 40180024 0x000804FF // HLC command using HlcLength2 register
write 40180028 0x000D2EBC // Write 2EBC to PortOut5678 Register
write 4018002C 0x000FE3E0 // Set IO Group5 as HlcSer4
write 40180030 0x000FE4D0 // Set IO Group7 as HlcSer64

```

```

write 40180034 0x000803FF // HLC command using HlcLength1 register
write 40180038 0x000804FF // HLC command using HlcLength2 register
write 4018003C 0x000FE40D // Set IO Group8 as HlcSer64
write 40180040 0x000804FF // HLC command using HlcLength2 register
write 40180044 0x000804FF // To make even number of interrogation

```

Here is the output for the above interrogations:

<pre> Valid ML interrogation Configure M2Config registers --> HLC Length 1 config register: 54 Valid ML interrogation Configure M2Config registers --> HLC Length 2 config register: 45 Valid ML interrogation Configure M2Config registers --> IOGroup56 Config register: c0 OO Interrogation HlcLength1Config >> n1: 4 n2: 5 hlcStr length: 6656 using hlcSerial Valid ML interrogation Configure M2Config registers --> IOGroup56 Config register: c OO Interrogation HlcLength1Config >> n1: 4 n2: 5 hlcStr length: 6656 using hlcSerial Valid ML interrogation Configure M2Config registers --> IOGroup78 Config register: c0 </pre>	<pre> OO Interrogation HlcLength2Config >> n1: 5 n2: 4 hlcStr length: 12288 using hlcSerial Valid ML interrogation Configure M2Config registers --> IOGroup78 Config register: c OO Interrogation HlcLength2Config >> n1: 5 n2: 4 hlcStr length: 12288 using hlcSerial Valid ML interrogation Write access to output port register 5678 ---- MLA Valid ML interrogation Configure M2Config registers --> IOGroup56 Config register: e0 Valid ML interrogation Configure M2Config registers --> IOGroup78 Config register: d0 OO Interrogation HlcLength1Config >> n1: 4 n2: 5 hlcStr length: 6656 using hlcPar64 on channel 2e OO Interrogation HlcLength2Config >> n1: 5 n2: 4 hlcStr length: 12288 using hlcPar64 on channel 2e </pre>
--	--

Figure 48 OO Interrogations and results

Testing ML16 interrogations

ML16 interrogations are special kind of ML interrogations where IO operations are also involved. Here are some ml16 interrogations:

```

write 40180000 0x000FE140 // IOGroup1 configured as ML161DS161
write 40180004 0x00091234 // ML16 operation on IO Group1
write 40180008 0x000FE1D0 // IOGroup1 configured as ML16
write 4018000C 0x000FE20D // IOGroup4 configured as ML16
write 40180010 0x00091234 // ML16 operation on IO Group1 and IO
Group4

```



```
write 40180014 0x00091234 // ML16 operation on IO Group1 and IO
Group4
```

Outputs after executing the instructions:

```
Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: 40

Valid ML interrogation
IOGRP1 = 0100 -- ML16DS16 and ML Data: 1234

Valid ML interrogation
Configure M2Config registers
--> IOGroup12 Config register: d0

Valid ML interrogation
Configure M2Config registers
--> IOGroup34 Config register: d

Valid ML interrogation
IOGRP1 and IOGRP4 = ML16 and ML Data: 1234

Valid ML interrogation
IOGRP1 and IOGRP4 = ML16 and ML Data: 1234
```

Figure 49 ML16 interrogations and results