

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

**DESIGN OF A MICROPROCESSOR-BASED EMBEDDED FAULT
DIAGNOSTIC SYSTEM AND AN FPGA-BASED IMPROVEMENT PROPOSAL**



M.Sc. THESIS

Onur BEKAR

Department of Electronics and Communication Engineering

Electronics Engineering Programme

JUNE 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

**DESIGN OF A MICROPROCESSOR-BASED EMBEDDED FAULT
DIAGNOSTIC SYSTEM AND AN FPGA-BASED IMPROVEMENT PROPOSAL**



M.Sc. THESIS

**Onur BEKAR
(504181292)**

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Ece Olcay GÜNEŞ

JUNE 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

**MİKROİŞLEMCİ TABANLI BİR GÖMÜLÜ TANI SİSTEMİ TASARIMI VE
FPGA TABANLI BİR OPTİMİZASYON ÖNERİSİ**

YÜKSEK LİSANS TEZİ

**Onur BEKAR
(504181292)**

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Ece Olcay GÜNEŞ

HAZİRAN 2022

Onur BEKAR, a M.Sc. student of İTÜ Graduate School student ID 504181292 successfully defended the thesis/dissertation entitled “DESIGN OF A MICROPROCESSOR-BASED EMBEDDED FAULT DIAGNOSTIC SYSTEM AND AN FPGA-BASED IMPROVEMENT PROPOSAL”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Ece Olcay GÜNEŞ**
Istanbul Technical University

Jury Members : **Prof. Dr. Müştak Erhan YALÇIN**
Istanbul Technical University

Assoc. Prof. Dr. Revna ACAR VURAL
Yıldız Technical University

Date of Submission : 16 May 2022
Date of Defense : 06 June 2022



FOREWORD

First of all, I would like to express my gratitude and serve my respects to my thesis advisor, Prof. Dr. Ece Olcay GÜNEŞ, for her guidance, interest and support. It was an honour for me to have the opportunity to work with her.

Furthermore, I would like to thank my family, however especially my wife Gizem BEKAR, from the bottom of my heart for their understanding and limitless support.

In addition, I serve my loads of appreciation to ASELSAN, where I work as a design engineer, for the opportunity provided. I would also like to thank my workmate, Ahmet ÇAKALOĞLU, for his helps for test activities in the microprocessor-based design verification phase.

May 2022

Onur BEKAR
(Electronics and Communication Engineer)



TABLE OF CONTENTS

	<u>Page</u>
FOREWORD	vii
TABLE OF CONTENTS.....	ix
ABBREVIATIONS	xi
SYMBOLS	xiii
LIST OF TABLES	xv
LIST OF FIGURES	xvii
SUMMARY	xxi
ÖZET.....	xxv
1. INTRODUCTION.....	1
1.1 Purpose of Thesis	3
1.2 Literature Review	4
2. A MICROPROCESSOR-BASED FAULT DIAGNOSTIC SYSTEM DESIGN	
.....	7
2.1 Hardware Development Process	14
2.2 Software Development Process.....	17
2.2.1 MRAM write & read functions	17
2.2.2 MRAM continuous write functions	21
2.2.3 EEPROM write & read functions	23
2.2.4 EEPROM erase functions	30
2.2.5 EEPROM – MRAM communication function.....	37
2.2.6 Final system architecture & design specification	40
3. TESTS FOR THE MICROPROCESSOR-BASED SYSTEM.....	43
3.1 MRAM Write & Read Functions Test Results	45
3.2 MRAM Continuous Write Functions Test Results	58
3.3 EEPROM Write & Read Functions Test Results	71
3.4 EEPROM Erase Functions Test Results	78
3.5 The Case of Error Final Tests (The Final Error Scenario Tests)	88
3.5.1 Accuracy of the diagnostic software	90
3.5.2 Performance of the diagnostic software.....	91
4. AN FPGA-BASED PROPOSAL FOR THE FDS DESIGN.....	97
4.1 SPI Word Transfer Function	97
4.2 Fault Diagnostic System Design	101
4.3 Multitasking Fault Diagnostic System Design.....	105
4.3.1 Error recording decision.....	106
4.3.2 Final system architecture & design specification	108
5. TESTS FOR THE FPGA-BASED SYSTEM	111
5.1 Tests for SPI Word Transfer Function	111
5.2 Final Error Scenario Tests for FDS	116
5.2.1 The test environment.....	117
5.2.2 The test results	119
5.3 The Final Error Scenario Tests for MFDS	131
5.3.1 The test results of error recording decision module.....	131

5.3.2 The test environment.....	135
5.3.3 The test results.....	136
6. CONCLUSIONS AND RECOMMENDATIONS	155
6.1 Contributions of This Study	155
6.2 Comparisons.....	158
6.3 Suggestions for Future Work.....	161
7. REFERENCES.....	163
8. CURRICULUM VITAE	165



ABBREVIATIONS

CCS	: Code Composer Studio
CoTS	: Commercial Off-The-Shelf
CPHA	: Clock Phase
CPOL	: Clock Polarity
CPU	: Central Processing Unit
CS	: Chip Select
DTR	: Double Transfer Rate for EEPROM IC
ECC	: Error Correcting Code
EEPROM	: Electronically Erasable Programmable Read-Only Memory
EPROM	: Erasable Programmable Read-Only Memory
FDS	: Fault Diagnostic System
FPGA	: Field Programmable Gate Array
HDL	: Hardware Description Language
I / O	: Input / Output
IC	: Integrated Circuit
ICU	: Inverter Control Unit
IEEE	: Institute of Electrical and Electronic Engineers
LCD	: Liquid-crystal Display
LED	: Light-emitting Diode
LSB	: Least Significant Bit
LUT	: Look-up Table
MFDS	: Multitasking Fault Diagnostic System
MISO	: Master In Slave Out
mod	: Modulo Operator
MOSI	: Master Out Slave In
MRAM	: Magnetoresistive Random Access Memory
MSB	: Most Significant Bit
PCB	: Printed Circuit Board
RAM	: Random Access Memory
READ	: Read Data Bytes

ROM	: Read-Only Memory
RTR	: Real-Time Recording
RX	: Receive
SCLK	: Serial Clock
SI	: Serial Data In
SIMO	: Slave In Master Out
SMART	: Self-Monitoring Analysis and Reporting Technology
SO	: Serial Data Out
SOMI	: Slave Out Master In
SPI	: Serial Peripheral Interface
SS / CSS	: Slave Enabled Signal
TI	: Texas Instruments
TX	: Transmit
USB	: Universal Serial Bus
VHDL	: VHSIC Hardware Description Language
VHSIC	: Very High Speed Integrated Circuit

SYMBOLS

BP0, BP1	: MRAM Bits
DQ [3:0]	: Serial I/O of EEPROM IC
GNDD	: Digital Ground pin
Hz	: Hertz
Kb	: Kilobit
KB	: Kilobyte
kHz	: Kilohertz
MB	: Megabyte
READ	: The Read Data Bytes Command
s	: Second
S#	: Chip Select Pin of EEPROM IC
SRWD	: Status Register Write Disable bit
^tBE	: EEPROM IC Bulk Erase Time
^tSE	: EEPROM IC Sector Erase Time
^tSSE	: EEPROM IC Subsector Erase Time
WEL	: Write Enable Latch Bit
WRITE	: The Write Data Bytes Command
n_{error}	: Total Size of an Error Message
n_{packet}	: Size of a Packet
f_s	: Sampling Frequency
V_{CC}	: Common Collector Voltage (The positive supply voltage for an IC)
Δt_{recording}	: Recording Period
#_{sensors}	: Number of Sensors
$\overline{\text{CS}}$	: Chip Select Pin of MRAM IC
$\overline{\text{WP}}$	: Write Protection Pin of MRAM IC



LIST OF TABLES

	<u>Page</u>
Table 2.1 : SPI modes [15].....	13
Table 2.2 : Status register bit assignments [17].	19
Table 2.3 : Memory protection modes [17].....	19
Table 2.4 : FLASH memory PROGRAM operations [16].	24
Table 2.5 : WRITE ENABLE/DISABLE operations [16].....	27
Table 2.6 : READ MEMORY operations [16].	28
Table 2.7 : FLASH memory ERASE operations [16].....	31
Table 2.8 : FLASH erase functions created within the scope of the thesis.....	37
Table 3.1 : MRAM write & read functions test details.	46
Table 3.2 : MRAM continuous write function test details.....	58
Table 3.3 : EEPROM write & read functions test details.	72
Table 3.4 : Initial test details.	79
Table 3.5 : Expected results after bulk erasing.	79
Table 3.6 : Expected results after sector erasing.	83
Table 3.7 : Initial test details.	85
Table 3.8 : Expected results after slot erasing.....	85
Table 3.9 : Test details.	90
Table 5.1 : Test details.	112
Table 6.1 : Comparison table.	160
Table 6.2 : Systems' progress table.....	161



LIST OF FIGURES

	<u>Page</u>
Figure 2.1 : Operating logic of the diagnostic system.	9
Figure 2.2 : Structure of an error message.	11
Figure 2.3 : SPI regular independent slave configuration.	13
Figure 2.4 : Block diagram of the diagnostic PCB.	16
Figure 2.5 : Write data bytes (WRITE) timing [17].	18
Figure 2.6 : MRAM write function flowchart.	18
Figure 2.7 : Read data bytes (READ) timing [17].	20
Figure 2.8 : MRAM read function flowchart.	21
Figure 2.9 : MRAM continuous write function flowchart.	22
Figure 2.10 : FLASH IC page program operation timing [16].	25
Figure 2.11 : FLASH write function flowchart.	26
Figure 2.12 : WRITE ENABLE and WRITE DISABLE timing [16].	27
Figure 2.13 : READ timing [16].	29
Figure 2.14 : FLASH read function flowchart.	29
Figure 2.15 : SUBSECTOR and SECTOR ERASE timing [16].	31
Figure 2.16 : BULK ERASE timing [16].	32
Figure 2.17 : FLASH sector erase function flowchart.	33
Figure 2.18 : FLASH bulk erase function flowchart.	34
Figure 2.19 : FLASH slot erase function flowchart.	35
Figure 2.20 : 2 nd parallel block of the FLASH slot erase function flowchart.	36
Figure 2.21 : EEPROM – MRAM communication function flowchart.	39
Figure 2.22 : The main function flowchart.	41
Figure 3.1 : The microprocessor-based ICU setup view-1.	43
Figure 3.2 : The microprocessor-based ICU setup view-2.	44
Figure 3.3 : The microprocessor-based ICU setup view-3.	44
Figure 3.4 : MRAM read and write test procedure.	46
Figure 3.5 : READ1 oscilloscope view.	47
Figure 3.6 : READ1 oscilloscope view interpreted.	48
Figure 3.7 : READ1 CCS expression view.	48
Figure 3.8 : READ2 oscilloscope view interpreted.	49
Figure 3.9 : READ2 CCS expression view.	49
Figure 3.10 : READ3 oscilloscope view interpreted.	50
Figure 3.11 : READ3 CCS expression view.	50
Figure 3.12 : READ4 oscilloscope view interpreted.	51
Figure 3.13 : READ4 CCS expression view.	51
Figure 3.14 : READ5 oscilloscope view interpreted.	52
Figure 3.15 : READ5 CCS expression view.	52
Figure 3.16 : READ6 oscilloscope view interpreted.	53
Figure 3.17 : READ6 CCS expression view.	53
Figure 3.18 : READ7 oscilloscope view interpreted.	54
Figure 3.19 : READ7 CCS expression view.	54
Figure 3.20 : READ8 oscilloscope view interpreted.	55

Figure 3.21 : READ8 CCS expression view.	55
Figure 3.22 : READ9 oscilloscope view interpreted.	56
Figure 3.23 : READ9 CCS expression view.	56
Figure 3.24 : READ10 oscilloscope view interpreted.	57
Figure 3.25 : READ10 CCS expression view.	57
Figure 3.26 : READ1 oscilloscope view interpreted.	59
Figure 3.27 : READ1 CCS expression view.	60
Figure 3.28 : READ2 oscilloscope view interpreted.	60
Figure 3.29 : READ2 CCS expression view.	61
Figure 3.30 : READ3 oscilloscope view interpreted.	61
Figure 3.31 : READ3 CCS expression view.	62
Figure 3.32 : READ4 oscilloscope view interpreted.	62
Figure 3.33 : READ4 CCS expression view.	63
Figure 3.34 : READ5 oscilloscope view interpreted.	63
Figure 3.35 : READ5 CCS expression view.	64
Figure 3.36 : READ6 oscilloscope view interpreted.	64
Figure 3.37 : READ6 CCS expression view.	65
Figure 3.38 : READ7 oscilloscope view interpreted.	65
Figure 3.39 : READ7 CCS expression view.	66
Figure 3.40 : READ8 oscilloscope view interpreted.	66
Figure 3.41 : READ8 CCS expression view.	67
Figure 3.42 : READ9 oscilloscope view interpreted.	67
Figure 3.43 : READ9 CCS expression view.	68
Figure 3.44 : READ10 oscilloscope view interpreted.	68
Figure 3.45 : READ10 CCS expression view.	69
Figure 3.46 : MRAMs continuous write function oscilloscope view.	70
Figure 3.47 : SPI bus zoomed oscilloscope view-1.	70
Figure 3.48 : SPI bus zoomed oscilloscope view-2.	71
Figure 3.49 : SPI bus zoomed oscilloscope view-3.	71
Figure 3.50 : FLASH read and write test procedure.	72
Figure 3.51 : READ1 oscilloscope view interpreted.	73
Figure 3.52 : READ2 oscilloscope view interpreted.	73
Figure 3.53 : READ3 oscilloscope view interpreted.	74
Figure 3.54 : READ4 oscilloscope view interpreted.	74
Figure 3.55 : READ5 oscilloscope view interpreted.	75
Figure 3.56 : READ6 oscilloscope view interpreted.	75
Figure 3.57 : READ7 oscilloscope view interpreted.	76
Figure 3.58 : READ8 oscilloscope view interpreted.	76
Figure 3.59 : READ9 oscilloscope view interpreted.	77
Figure 3.60 : READ10 oscilloscope view interpreted.	77
Figure 3.61 : All reading results on CCS expression view.	78
Figure 3.62 : All reading results on CCS expression view.	80
Figure 3.63 : All reading results on oscilloscope screen.	80
Figure 3.64 : All reading results on CCS expression view after bulk erasing.	81
Figure 3.65 : All reading results on oscilloscope screen after bulk erasing.	81
Figure 3.66 : All reading results on CCS expression view.	82
Figure 3.67 : All reading results on oscilloscope screen.	82
Figure 3.68 : All reading results on CCS expression view after sector erasing.	83
Figure 3.69 : All reading results on oscilloscope screen after sector erasing.	84
Figure 3.70 : All reading results on CCS expression view.	85

Figure 3.71 : All reading results on oscilloscope screen.....	86
Figure 3.72 : All reading results on CCS expression view after slot erasing.	86
Figure 3.73 : All reading results on oscilloscope screen after slot erasing.....	87
Figure 3.74 : The error recording function final test scenario flowchart.....	89
Figure 3.75 : Accuracy tests CCS expression view.	91
Figure 3.76 : Error recording process oscilloscope view.....	92
Figure 3.77 : The error recording period.....	93
Figure 3.78 : The error recording period after optimizations.	95
Figure 4.1 : Block diagram of SPI word transfer function.....	98
Figure 4.2 : SPI word transfer function flowchart (part-1).	100
Figure 4.3 : SPI word transfer function flowchart (part-2).	101
Figure 4.4 : Block diagram of the diagnostic system (FPGA).	103
Figure 4.5 : Error recording decision flowchart.....	107
Figure 4.6 : Block diagram of the multitasking diagnostic system.....	109
Figure 5.1 : Block diagram of the test structure.....	112
Figure 5.2 : General view of the test.	113
Figure 5.3 : Period before the transmission.	113
Figure 5.4 : First word transfer.	114
Figure 5.5 : Second word transfer.	114
Figure 5.6 : Third word transfer.....	115
Figure 5.7 : Fourth word transfer.	115
Figure 5.8 : Last word transfer and end of the program.	115
Figure 5.9 : Data of transmission and receive buffers.	116
Figure 5.10 : Block diagram of the final test environment.	118
Figure 5.11 : General view of the test result (diagnostic system).....	120
Figure 5.12 : Ready state.....	121
Figure 5.13 : Beginning of real time recording state.	123
Figure 5.14 : End of real time recording state.....	124
Figure 5.15 : Beginning of error state.	126
Figure 5.16 : End of error state.	127
Figure 5.17 : Beginning of error transfer state.....	129
Figure 5.18 : End of error transfer state.	130
Figure 5.19 : Block diagram of the error recording decider test environment.....	132
Figure 5.20 : No error scenario test result.....	133
Figure 5.21 : Single error scenario test result.	133
Figure 5.22 : Worst-case scenario test result.	134
Figure 5.23 : Block diagram of the final test environment.	136
Figure 5.24 : No error scenario test result part-1.	138
Figure 5.25 : No error scenario test result part-2.	139
Figure 5.26 : Single error scenario test result part-1.....	140
Figure 5.27 : Single error scenario test result part-2.....	141
Figure 5.28 : The changeover moment of active diagnostic system.....	142
Figure 5.29 : Worst case scenario test result part-1.	144
Figure 5.30 : Worst case scenario test result part-2.	145
Figure 5.31 : Emphasized parts of the worst case scenario test result.....	146
Figure 5.32 : The first changeover of active diagnostic system.	149
Figure 5.33 : The second changeover of active diagnostic system.....	150
Figure 5.34 : The third changeover of active diagnostic system.	151
Figure 5.35 : The fourth changeover of active diagnostic system.	153
Figure 5.36 : The fifth changeover of active diagnostic system.....	154

Figure 6.1 : The time interval new sensor measurements cannot be recorded.	156
Figure 6.2 : Uninterrupted logging (MFDS).	157



DESIGN OF A MICROPROCESSOR-BASED EMBEDDED FAULT DIAGNOSTIC SYSTEM AND AN FPGA-BASED IMPROVEMENT PROPOSAL

SUMMARY

In the past, the use of proprietary hardware, software and protocols in systems and the lack of continuous interaction between the systems /subsystems were a natural obstacle to prevent critical errors from occurring for systems. Therefore, troubleshooting, which has been important in engineering from the past to the present, was not as challenging as in current applications, and the detection of errors was not seen as important part of as it is at present in systems' lifecycle.

However, today's applications are complex systems consisting of embedded systems, units, modules etc. that continuously interact with each other, with interfaced, chip to chip, wired and/or wireless communication between. In addition, the widespread use of new technologies, protocols, commercial off-the-shelf (CoTS) products and operating systems has made a remarkable effect to increasing the error's frequency. The increasing complexity of engineering applications has brought challenges associated with the exposure to multifarious failures or errors, affecting the systems' reliability, safety, availability and performability. In this sense, new concepts such as error detection, debugging, error recovery, correction, maintenance and repair over have become indispensable elements of engineering applications.

In particular, the detection of errors has become even more important in order to correct the errors, ensure maximum efficiency, and provide optimized and sustainable systems. In this context, the systems, named as Fault Diagnostic Systems; designed specifically for the purpose of detecting and recording the error conditions on the software, hardware or equipment etc. have become widespread in different fields such as, defence industry, industrial control and automation systems, aerospace industry, railway transportation sector, smart factories, electric and hybrid vehicles, power plants, network and information technologies, automotive, medical/healthcare systems and have become an important part of the system design phase.

In this thesis, a microprocessor-based embedded fault diagnostic system, which will operate in conformity with or integrated into main units such as control, automation, power units to be used in numerous fields is designed. This diagnostic system is fundamentally a logger system that continuously records sensor data to be sent from the main system and creates the "Error Messages" formed from the waveforms measured by the sensors. The related system is responsible for and able to record the measurements real-timely, operate in different modes according to the present scenarios, detect the data for a certain second before the error occurs in case of a situation that is considered as an "Error" by the main system, and store the error messages permanently in protected memory structures.

In the the first chapter of the study, a comprehensive literature review on fault diagnostic systems and embedded systems is made. In relevant parts of the chapter, the related issues are researched in details such as their history, purposes, structures, operating logics, architectures, hardware and software components etc.

In the second chapter, a microprocessor-based diagnostic system is designed by starting from the requirements and conceptual system design. Then, it is advanced to Detailed System Design Phase which the system architecture and operating scenario for the system are built and needed specifications such as sampling frequency, data sizes, required memory etc. are calculated. Afterwards, the hardware and software development processes are started respectively by taking the system level designs as reference. In hardware design studies, components used in the Diagnostic PCB such as memory units are selected according to the results of analysis on memory types. In the final part of the hardware design phase, a printed circuit board named Diagnostic PCB is designed in Altium PCB Design Software & Tools by using the recommended information in datasheets, PCB design rules and lessons learnt from past experiences. Then, the PCB is manufactured and verifications tests are performed. Thus, the hardware is verified and software development processes in C programming language are started. In software development process, software architecture is built and the architectural design is made based on the scenario determined in the system design section. The Diagnostic software is designed and verified one by one.

In the third chapter, accuracy and performance tests are conducted for the Diagnostic System in the microprocessor-based inverter control unit (ICU) setup, and the results are given. Within the scope of the tests, a dummy traffic is generated and the external systems that Diagnostic System will work with or operate into are simulated. By doing so, real operation scenarios run, and essential optimizations for software are made. Then, the accuracy and performance of the system are retested again and again by simulating the real environment, and the system is verified and validated.

The last development within the scope of the thesis is an FPGA-based improvement which is named Multitasking Diagnostic System proposed in the fourth chapter. The related system is developed starting from zero by using VHDL in the VIVADO Design Suite environment as high-level. However, it should be emphasized that no hardware design or tests on physical setup have been made, as in the system developed based on microprocessor. The new system model providing uninterrupted operation is created by considering the advantages of using FPGA for embedded applications such as parallel processing, speed, flexibility, extendibility.

In the fifth chapter, accuracy and performance tests are performed also for the Multitasking Diagnostic System through behavioural simulation on the VIVADO Design Suite environment. For all tests, a dummy sensor traffic is generated and a similar scenario is built by taking the scenario followed in the microprocessor-based design processes as reference. Scenarios are run repeatedly by simulating the real environment, and essential optimizations for firmware are made also for. And consequently, the system is also verified and validated.

In the last chapter, the microprocessor-based and FPGA-based system designs with their results are summarized, especially the worth-emphasizing parts are given by restating the dissertation main motivation which is the need of an Embedded Diagnostic System. The systems are examined, and a detailed comparison containing their advantages and disadvantages according to the requirements and circumstances is given.

Lastly, the contributions to the literature, and the future considerations related to the dissertation is introduced. In conclude, both of the microprocessor-based and FPGA-based systems are successfully designed, verified and validated.





MİKROİŞLEMCİ TABANLI BİR GÖMÜLÜ TANI SİSTEMİ TASARIMI VE FPGA TABANLI BİR OPTİMİZASYON ÖNERİSİ

ÖZET

Geçmişte sistemlerde tescilli donanım, yazılım ve protokollerin kullanılması ve sistemler/alt sistemler arasında sürekli etkileşimin olmaması, sistemlerde kritik hataların oluşmasının önünde doğal bir engeldi. Bu nedenle, geçmişten günümüze mühendislikte önemli olan arıza giderme, mevcut uygulamalardaki kadar zorlayıcı değildi ve hataların tespiti, sistem yaşam döngüsünün günümüzdeki kadar önemli bir parçası olarak görülmemekteydi.

Ancak günümüz uygulamaları; arayüzler üzerinden, çipten çipe, kablolu ve/veya kablosuz haberleşme ile birbirleriyle sürekli etkileşim halinde olan, birimler, gömülü sistemler, modüller vb.' den oluşan karmaşık sistemlerdir. Ayrıca yeni teknolojilerin, protokollerin, ticari kullanıma hazır (CoTS) ürünlerin ve işletim sistemlerinin yaygın olarak kullanılması, hata sıklığının artmasında dikkate değer bir etki yaratmıştır. Mühendislik uygulamalarının artan karmaşıklığı, sistemlerin güvenilirliğini, güvenliğini, kullanılabilirliğini ve sürdürülebilirliğini etkileyen çok çeşitli arızaları ve/veya hatalara maruz kalma sıklığı konusunda zorlukları da beraberinde getirmiştir. Bu anlamda hata tespiti, hata ayıklama, hata kurtarma, düzeltme, bakım ve onarım gibi yeni kavramlar mühendislik uygulamalarının vazgeçilmez unsurları haline gelmiştir.

Özellikle günümüz uygulamalarında bu hataları düzeltmek, optimize edilmiş ve sürdürülebilir sistemler sağlayabilmek adına hataların tespiti daha da önemli hale gelmiştir. Bu kapsamda özel olarak hataları tespit etmek ve kayıt altına almak amacıyla tasarlanan Tanı Sistemleri; savunma sanayi, endüstriyel kontrol ve otomasyon sistemleri, havacılık sanayi, demiryolu ulaşım sektörü, akıllı fabrikalar, elektrikli ve hibrit araçlar, enerji santralleri, ağ ve bilgi teknolojileri, otomotiv, medikal/sağlık sistemleri gibi farklı alanlarda yaygınlaşmış, sistem tasarım aşamasının önemli bir parçası haline gelmiştir.

Arızaların tespiti, izolasyonu, loglanması ve tanımlanmasına odaklanan ilgili sistemler, tasarım mühendisleri, test mühendisleri, kullanıcılar ve bakım personeli vb. ilgili kişilere sorunlar konusunda rehberlik sağlamıştır. Ayrıca, üretim sürecinin, operasyonların ve ilgili sistemlerin sistem yaşam döngüsü boyunca maksimum verimliliğini sağlamak için zaman ve maliyetleri kontrol ederken şirketlerin kaynaklarını korumalarına yardımcı olurlar. Tanı sistemi, güvenilir bir çözüm sağlamaya, hata ayıklama sürecini hızlandırmaya ve kök neden analizini basitleştirmeye yardımcı olan önemli bir araçtır. Mühendislik problemlerine sistematik olarak, bir algoritma uygulanarak veya uygulanabilir bir çözüme ulaşılması için adım adım çözümleme yapılarak yaklaşılmalıdır. Bu yönüyle mühendislik problemlerinin etkin bir şekilde çözülmesi için mühendislik disiplinlerinde hataların tanı sistemleri ile teşhis edilmesi çok önemli bir rol oynamaya başlamıştır.

Tanı Sisteminin bu hayati görevlerinden bazıları, izlenen sistemde bir şeylerin ters gittiğinin belirlenmesi, hangi hata adayının gözlenen anormal davranışı açıklayabileceğinin belirlenmesi, yazılım, donanım veya ekipman arızalarının nedeni olan hata durumlarının tespit edilmesi, verilerin kaydedilmesi ve tespit edilen arızanın büyüklüğünün ve türünün belirlenmesidir.

Bu tezde, birçok alanda kullanılabilecek kontrol, otomasyon, güç üniteleri gibi ana birimlerle uyumlu veya entegre çalışacak mikroişlemci tabanlı gömülü bir tanı sistemi tasarlanmıştır. Bu tanı sistemi temel olarak, ana sistemden gönderilecek sensör verilerini sürekli olarak kaydeden ve sensörlerin ölçtüğü dalga formlarından oluşan “Hata Mesajlarını” oluşturan bir logger sistemidir. İlgili sistem, elektriksel ölçümleri gerçek zamanlı olarak kaydetmekten, mevcut senaryolara göre farklı modlarda çalışmaktan, “Hata” olarak kabul edilen durumlarda hata oluşmadan önce ve sonra belirli bir saniye boyunca kaydedilen ilgili sensör verileri kullanılarak hata mesajlarını ve hataları korumalı bellek yapılarında kalıcı olarak saklamaktan sorumludur. Arıza durumları elektrik kesintisi de dahil olmak üzere elektriksel anormallikler içerebileceğinden, sistem tasarlanırken elektrik kesintisi durumunda bile bu hata mesajlarının silinmemesi gerektiği göz önünde bulundurulmuştur.

Çalışmanın birinci bölümünde tanı sistemleri, çeşitli alanlardaki uygulamaları, gömülü sistemler ve yapıları hakkında kapsamlı bir literatür taraması yapılmıştır. Bölümün ilgili kısımlarında incelenen sistemleri tarihçeleri, amaçları, çalışma mantıkları, donanım ve yazılım bileşenleri vb. ilgili konular detaylı olarak araştırılmıştır.

İkinci bölümde, gereksinimler ve kavramsal sistem tasarımından yola çıkılarak mikroişlemci tabanlı bir tanı sistemi tasarlanmıştır. Ardından sistem mimarisinin ve sistem çalışma senaryosunun oluşturulduğu, örnekleme frekansı, veri boyutları, gerekli bellek alanı, sinyal sayısı vb. gibi ihtiyaç duyulan özelliklerin hesaplandığı Ayrıntılı Sistem Tasarım Aşamasına geçilmiştir. Daha sonra sistem düzeyindeki tasarımlar referans alınarak sırasıyla donanım ve yazılım geliştirme süreçlerine geçilmiştir. Donanım tasarımı çalışmalarında, bellek tipleri üzerinde yapılan analiz sonucuna göre Diagnostik PCB'de kullanılan bellek birimleri gibi bileşenler seçilmiştir. Donanım tasarım aşamasının son bölümünde, Altium PCB Design Software & Tools programında bileşen veri sayfalarında önerilen bilgiler, PCB tasarım kuralları ve geçmiş deneyimlerden alınan dersler kullanılarak Diagnostik PCB adlı bir baskılı devre kartı tasarlanmıştır. PCB üretildikten ve tüm bileşenler lehimlendikten sonra donanım doğrulama testleri yapılmış, böylece donanım doğrulanmış ve C programlama dilinde yazılım geliştirme süreçlerine başlanmıştır. Yazılım geliştirme sürecinde ana fonksiyon için yazılım mimarisi oluşturulmuş, sistem tasarımı bölümünde belirlenen senaryoya göre mimari tasarım yapılmıştır. Daha sonra ilgili tasarım alt fonksiyonlara ayrılmış ve tez kapsamında oluşturulan tanı yazılımının bu alt fonksiyonları tek tek tasarlanmış ve doğrulanmıştır. Bu alt işlevler, doğrudan MRAM ve EEPROM entegrelerinin işlevleridir. İlgili bölümün sonunda tasarlanan alt fonksiyonların yazılımları öncesinde oluşturulan sistem işleyiş senaryosu referans alınarak birbirleriyle entegre edilmiş, ana yazılım yazılmıştır. İlgili yazılım gerçek zamanlı kayıt, hata durumu, hata mesajının geçici hafıza birimleri olan MRAM’lerde oluşturulması ve hata mesajının kaydının tamamlanması durumunda ilgili verinin geçici hafıza birimlerinden okunarak, EEPROM kalıcı hafıza birimlerine aktarımı gibi tanılama görevleri yerine getirmektedir.

Üçüncü bölümde, mikroişlemci tabanlı evirici kontrol ünitesi (ICU) test platformu oluşturulmuş, tasarlanan Tanı Sistemi için doğruluk ve performans testleri yapılmıştır.

Sonuçlar incelenerek, detaylarıyla verilmiştir. Testler kapsamında yapay bir trafik oluşturularak, Tanı Sisteminin birlikte çalışacağı veya entegre edileceği harici sistemlerin davranışları yazılımsal olarak modellenmiştir. Böylece gerçek operasyon senaryoları çalıştırılmış ve sonuçlara istinaden yazılım için gerekli optimizasyonlar yapılmıştır. Ardından sistemin doğruluğu ve performansı gerçek ortam simüle edilerek tekrar tekrar test edilmiştir. Böylece, sistem doğrulanmış ve tasarım dondurulmuştur.

Tez kapsamındaki son geliştirme ise dördüncü bölümde önerilen Paralel İşletimli / Çoklugoğrevli Tanı Sistemi (Multitasking Diagnostic System) adlı FPGA tabanlı bir iyileştirmedir. İlgili sistem VIVADO Design Suite ortamında yüksek-seviyeli olarak VHDL dilinde sıfırdan geliştirilmiştir. Ancak FPGA tabanlı olarak tasarlanan sistemin sadece bir teknik öneri olduğunu vurgulamak gerekir. Dolayısıyla mikroişlemciye dayalı olarak geliştirilen sistemde olduğu gibi herhangi bir donanım tasarımı veya fiziksel kurulum (ekipman) üzerinde testler yapılmamıştır. Sistem temel olarak, paralel çalışan dört tanı sistemi, bu sistemlerin yönetimi ve senkronizasyonunun sağlaması adına tasarlanmış bir yönetim ve karar yazılımından oluşmaktadır. İlgili sistem, gömülü uygulamalar için FPGA kullanımının paralel işlem yeteneği, hız, esneklik, genişletilebilirlik gibi avantajları göz önünde bulundurularak önerilen sistem, kesintisiz çalışma ve kayıpsız veri kaydı sağlayan yeni bir sistem modelidir.

Beşinci bölümde, VIVADO Design Suite ortamında davranışsal benzetim yoluyla Çoklugoğrevli Tanı Sistemi için de doğruluk ve performans testleri yapılmıştır. Benzer şekilde, tüm testler için yapay bir sensör trafiği oluşturulmuş, mikroişlemci tabanlı tasarım süreçlerinde izlenen senaryo referans alınarak benzer bir senaryo oluşturulmuştur. Ancak, tasarım bölümünde de belirtildiği gibi FPGA tabanlı geliştirme süreci geleceğe yönelik teknik bir öneri olduğundan, mikroişlemci tabanlı sistemin aksine, tanı sisteminin birlikte çalışacağı veya entegre olacağı dış birimlerin davranışlarının modellenmesi, FPGA-tabanlı sistem testleri için yeni bir ihtiyaçtır. Bu bağlamda, önerilen sistemin birlikte çalışacağı tüm birimler de yine VHDL donanım tasarım dili kullanarak tasarlanmıştır. VIVADO test ortamının kurulması adına yapılan son çalışmada yüksek hiyerarşi davranışsal bir test senaryosu oluşturulmuş, olası tüm senaryolar tekrar tekrar çalıştırılmıştır. Sistemin hatasızlığı ve performansı gerçek ortam simüle edilerek tekrar tekrar test edilmiş, sistem doğrulanmıştır.

Son bölümde ise mikroişlemci tabanlı ve FPGA tabanlı sistem tasarımları sonuçları ile özetlenmiş, özellikle tezin ana motivasyonu olan Gömülü Tanı Sistemi ihtiyacı yeniden ifade edilerek vurgulanmaya değer kısımları verilmiştir. Sistemler incelenmiş, ihtiyaç ve şartlara göre avantaj ve dezavantajlarını içeren detaylı bir karşılaştırma yapılmıştır. Katkılar ve tez kapsamında yapılan çalışmalar ile ilgili geleceğe yönelik değerlendirmeler verilmiştir.

Sonuç olarak, hem mikroişlemci tabanlı hem de FPGA tabanlı olarak geliştirilen sistemler tasarım hedeflerine uygun şekilde başarıyla tasarlanmış, doğrulanmış ve onaylanmıştır.



1. INTRODUCTION

A diagnostic program (or system) is an automatic computer program sequence that determines the operational status within the software, hardware, or any combination thereof in a component, a system, or a network of systems. Diagnostic programs ideally provide the user with guidance regarding any issues or problems found during its operation [1].

The increasing complexity of engineered systems, such as industrial processes, aircraft, road vehicles, manufacturing systems, electrical and electronic equipment, brings challenges associated with the exposure to diverse failure modes, affecting the systems' reliability, safety, and performability. Hence, there is an essential need to develop diagnostics and prognostics methods to implement for complex systems, operating in real-world conditions.

The diagnostics tasks are focused on fault detection, fault isolation, and fault identification. Fault detection is the task of identifying that something is going wrong in the monitored system as soon as a fault occurs, fault isolation determines which fault candidate can explain the observed abnormal behaviour, and fault identification determines the magnitude and type of the fault when it is detected.

The benefits of diagnostics for a complex system are significant; the biggest being that they are used to increase system safety and operational reliability and also to improve and optimize system design and development. It also has a powerful impact on the life-cycle cost, which reduce operating cost and increase revenue [2].

With the above-mentioned needs and the advantages, diagnostic systems, designed specifically for the purpose of detecting and recording the errors, they have become widespread in different fields such as defense industry, industrial control and automation systems, aerospace industry, railway transportation sector, smart factories, electric and hybrid vehicles, power plants, network and information technologies, automotive, medical/healthcare systems etc. and have become an important part of the system design phase [3], [4], [5], [6], [7], [8], [9], [10], [11].

The diagnostics system for a device or system may be independently located or integrated within. These Methods of Operation are arranged, more-or-less, in order of increasing complexity and their increasing value of diagnostics information. Some functionality examples for diagnostic systems that serve different purposes are given below:

1. Background monitoring of system indicators, for statistical analysis of trends, and for recording abnormal events.
2. Solution-based diagnostics, that test for known failure modes by identifying if their known symptoms are detected.
3. Black box, which is testing of a mechanism without knowing how it works, and merely focusing on the accuracy of output data based on a known input.
4. White box, which uses knowledge of a mechanism's inner functions for direct testing.
5. Operation-oriented, a combination of both black and white box, with one or more black-box operations interleaved with one or more white-box operations. This mode of testing is not preferred, however, some complex systems do not have the necessary interfaces to perform one or the other type independently.
6. Integrated background diagnostics, that perform testing of system components during idle-time of a system.
7. Operation-interleaved diagnostics, that incorporate diagnostics into the normal operation of a system component, thus any marginal operating mode is immediately diagnosed. Examples of hardware components with features that assist a diagnostic program are:
 1. Modern hard drives have Self-Monitoring, Analysis and Reporting Technology (SMART) commands that provide information about internal error conditions, e.g., spin retry counts, bad sector counts, etc.
 2. Some systems may use Error Correcting Code (ECC) memory that logs memory failure events that were automatically corrected [1].

1.1 Purpose of Thesis

Having a logger becomes crucial when debugging is not possible or convenient. Under some cases, debugging may not be possible—for example, when the application runs on specific hardware or when the problem is completely unclear [12]. In those cases, collection and analysis of the logs is crucial to determine the root of a problem. In this sense, having an internal recording unit has become an important feature desired and rising in importance day by day for electronic systems. In this sense, diagnostic systems which are responsible for detecting and recording the error conditions on the software or equipment malfunctions in many industries such as defense, railway transportation, automotive, information technologies, aerospace, biomedical etc. has emerged as an important technical need.

In this thesis, it is aimed to develop a microprocessor-based embedded fault diagnostic system, that is responsible of recording the sensor data measure from a main unit fulfilling the need of an internal logger unit.

This Diagnostic System is briefly a modular recording and playback system that is able to perform the operations such as continuously recording sensor data to be sent from the Main System, creating the Error Messages consisted of the waveforms for 4 seconds of 24 different sensor data. The primary purpose of the thesis is to design an embedded system that will operate in conformity with the main unit by analysing the main unit that the Diagnostic System will integrated into and determining the most suitable architecture, memory structure, protocol etc. solutions. For this purpose, conceptual and detailed system designs are carried out in the light of relevant needs, studies and research. Both software and hardware designs are made within the scope of the relevant system. For the hardware studies, a modular and generic PCB is designed, while the algorithm of the whole system software will be defined and developed in modules or functions.

Then, the accuracy and performance tests will be performed by simulating possible cases on a real hardware for the microprocessor-based Diagnostic System. According to the results, the system will be refocused and work on optimization studies.

The last study, proposes an FPGA-based improvement for the microprocessor-based Diagnostic System based on the results of the accuracy and performance tests performed for the finalized microprocessor-based system. It should be emphasized that

an FPGA-based system shall be only proposed for future work. So, the related system was designed as high-level, developed by using VHDL and tested in the VIVADO Design Suite environment. The design of the FPGA-based system has been created by considering the advantages of FPGA for embedded applications such as parallel processing, speed, flexibility, extendibility.

In conclusion, this study aims to provide a comprehensive embedded system that is capable of needed diagnosis operations profiting by the advantages of several architectures, software, hardware, protocols etc. in maximum and contributing to the literature.

1.2 Literature Review

In this study, it is aimed to design an embedded FDS, which operates in conformity with or integrated into main systems such as control, automation, power distribution units to be used in numerous fields. Hence, especially the engineering journals and conferences were reviewed in detail. In this section, previous academic works on fault diagnostic systems in various fields are gathered by citing them.

A battery module diagnostic system with low-cost and compact configuration was proposed by C. Hua Lin, K. Chen and Y. Lee. The proposed diagnostic system consists of a microcontroller, a signal capturing circuit, a human machine interface and a sinusoidal load [3]. Another article was devoted to the development of a multi-agent architecture of a medical system for automated diagnosis of human health. To implement the diagnostic technique, a multi-agent architecture was used, which allowed for each stage to create a flexible autonomous structure (module). The basic twolevel multi-agent architecture of a general-purpose medical system has five modules: an information collection module, an information storage module, a module for diagnosing the functional state of the human body, an information output module, and a user query module [4]. A. M. Boronakhin, D. Y. Larionov, L. N. Podgornaya and R. V. Shalymov published another article which is devoted to the possibilities of using such built-in diagnostic systems to solve the problems of ensuring safe and energy-efficient driving of locomotives, by taking the state of the rolling stock and rail track into account. Criteria for choosing the speed mode, as well as determining the composition, location and basic principles of information processing in an embedded diagnostic system, are considered [5]. As another example of FDS in railway industry,

authors set the problem of developing of the automated current collection failure diagnostic system based on receiving and processing the electromagnetic emissions in the radio frequency band from arc discharges. In the paper, two possible types of system structure based on a digital controller with RAM block and a radio modem were presented [6]. Another monitoring and diagnostic system was designed and developed by O. Mareček to facilitate the testing process for power plant electrical equipment. New monitoring system enables to record analogue and binary data acquired both from directly measured data (voltage, current, switch condition, relay, technological automatics) and data coming via data communication (communication protocols with protections, automatics, control systems) [7].

An important example from the aspect of the architecture and communication protocol, is the voltage measuring system, which based on the AVR microcontroller AT90CAN32 and multicell battery monitoring integrate circuits. The system includes the hardware design and the software design for Serial Peripheral Interface (SPI) communication [8]. The FDSs are also frequently used in defence industry. For instance, in the paper submitted by S. Kim, D. Kim and D. Kim an event log analysis software for naval combat systems using smart platform is proposed. The event log analysis software provides gathering node information and log analyzing function to support development of naval combat system [9]. The FDS examples for systems having large structures where a higher number of units are in communication with each other also was examined. To illustrate, the authors introduced an expert system, called AILAN, which is an intelligent diagnostic system for local area networks (LAN) in relevant study. This system is unique in that it entirely relies only on data acquired from the broadcast communication channel of the LAN [10].

Lastly, the paper proposes the development of On-Board Diagnostic (OBD) system. embedded on-board diagnostics system consists of master slave communication implemented through CAN network, is reviewed. In this system, sensors are interfaced to slave microcontroller for monitoring various parameters of system. Master microcontroller produces diagnostic trouble codes (DTC) to indicate the causes of the fault produced in parameters of system [11].



2. A MICROPROCESSOR-BASED FAULT DIAGNOSTIC SYSTEM DESIGN

Within the scope of the thesis, a microprocessor-based fault diagnostic system (FDS), which will operate with or integrated into a Main Unit, was designed. While the relevant FDS will be used in an already active power unit, the points to be considered while designing and the requirements determined according to the researches are given below:

- The FDS to be designed shall be integrated into an existing Power Unit and shall be responsible for continuously recording sensor data to be sent from the Main System (Power Unit).
- The anomalies considered as “An Error” shall be specified by the Main Unit, and the FDS shall be responsible for creating the “Error Message” (After this point, this data, which includes the 4-second sensor data, will be named as "Error Message").
- Error Message shall be formed from 2 signals. While the first signal consisting of 24 different sensor data comprises of 2-second waveforms before occurrence of the error, the second signal consists of the 2 seconds after the moment error occurred.
- Error messages shall not be erased even in the case of a power cut, as they may contain electrical anomalies, including also power failure.
- The moment that error occurred shall be detected by the main unit and an indication shall be sent by using the error flag.
- The number of sensors to be stored is 24.
- The sampling frequency for the sensor values is 5 kHz.
- Sensor messages shall be sent as 16-bit words from the Main Unit.

In the light of the requirements given above and as a result of the examinations made on the Main System, the concept system design of the FDS was made and the details are given below.

The system will basically operate in 3 different modes.

1. Real-Time Recording mode which live sensor data is recorded real-timely.
2. Error Mode, which the relevant sensor data is recorded for 2 seconds more after the error moment.
3. Error Transfer Mode, the error message shall be transferred in to the permanent memory structure.

By command of start the real-time recording process of the main system which the diagnostic system will be integrated into, the system starts operating. The sensor data transmitted from 24 sensors are recorded in the temporary memory structures, and new values shall be overwritten the old values in case of no errors. This recording continues until the real-time error record request is recalled or an error occurs. System transitions to the Error State with the indication of an error.

The behaviour of the FDS in the Error State is technically the same as it is in the Real Time Recording state. The only difference between the Real Time Recording and Error states, always a data with a certain size or length is recorded in the Error State. While the communication in the Real Time Recording State continues unless an opposite request from the main system or an error are occurred, in the Error State always a 2-second error message is recorded. In other words, while there is no definite end point for Serial Peripheral Interface (SPI) communication in the Real Time Recording, always a fixed-size message will be written in the Error State. In the Error State, the system automatically transitions to the Error Transfer State after all operations are executed properly.

The last state of the FDS is the Error Transfer State in which the 4-second error message, consisting of the sensor measurements sampled at 5 kHz, is transferred from temporary memory structures to the corresponding permanent error recording structure. If any abnormality regarded as an error by the Main System occurs, that data shall be recorded to permanent error recording structure. So, permanent error recording ICs are used in the Error Transfer State as distinct from the other modes.

The operating logic of the Diagnostic System is also shown in the Figure 2.1.

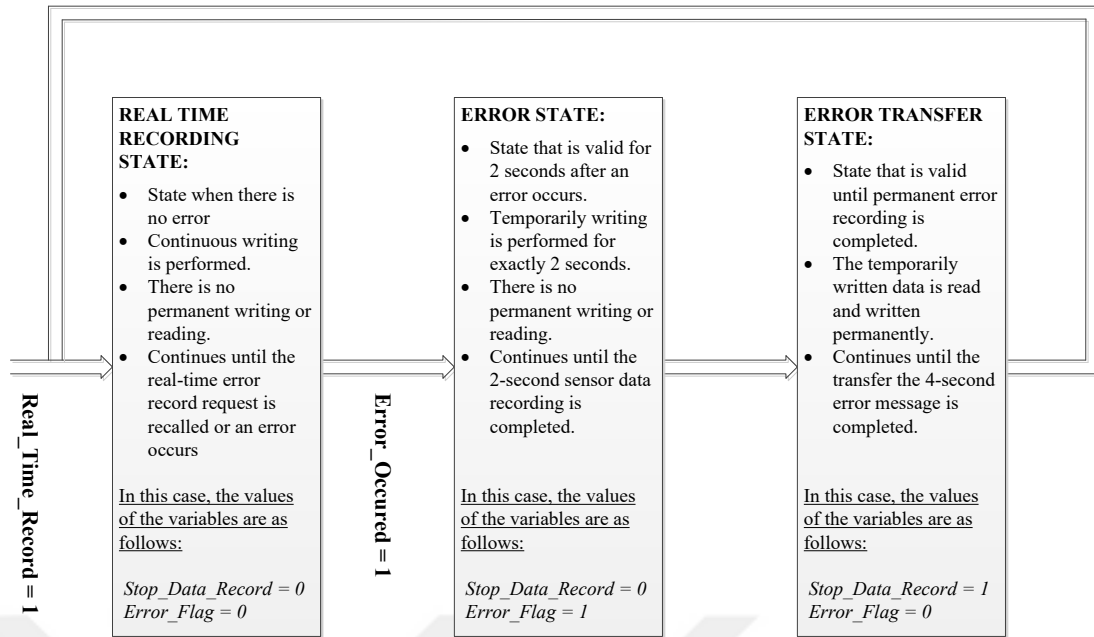


Figure 2.1 : Operating logic of the diagnostic system.

The recorded Error Messages must not be deleted even in the case of a power cut, or accidentally overwritings must be blocked since it also could result in the loss of an error message. Additionally, more capacity is needed in Error Transfer State, since the all stored data is transferred in that state. In the light of this information, the main required specifications of the memory that will be used for storing the error messages are listed as:

- Permanent medium of storage
- Being non-volatile storage that retains its data when power is removed
- Rewriteable but requires erase operation before writing
- Efficient use
- Higher capacity at lower costs

So, using a memory structure in rewriteable type of ROM has been determined as the most suitable solution taking into account all of these reasons.

However, even it is rewriteable type, ROMs can withstand only a limited number of write and erase cycles before the insulation is permanently damaged [13]. In addition, new data should not be stored in permanent memory unless there is an error, and the new data shall be overwritten with old ones. Therefore, using ROMs for real time recording causes shortening the system life cycle and increasing the cost.

In this sense, it has been determined to use a RAM type memory structure whose data can be read, written, and erased any number of times. The other main specifications of the memory that is used for real time recording required are given below:

- The data can be altered any number of times.
- High-speed memory.
- Less power compared to the other disks
- Increases the computer speed [14].

So, using a memory structure in non-volatile of RAM for real time recording has been determined as the most suitable solution taking into account all of these reasons.

Then, communication specifications (sampling frequency, data sizes, required memory, number of signals etc.) were specified and necessary calculations were made by taking the system operating scenario as a reference.

As emphasized before, the number of 24 sensors' data is recorded in scope of the thesis. The sensor values sampled at 5 kHz frequency will be transmitted in the digital format of 16-bit words and there will be a continuous and real-time communication between the main unit and the slaves. The data size for an error message (n_{error}) is calculated by using the size of a packet (n_{packet}), sampling frequency (f_s), recording period ($\Delta t_{\text{recording}}$) and number of the sensors ($\#_{\text{sensors}}$) as follows:

$$n_{\text{error}} = n_{\text{packet}} \times f_s \times \Delta t_{\text{recording}} \times \#_{\text{sensors}} \quad (2.1)$$

The total size of an error message is evaluated as 960.000 bytes. However, it should be emphasized that these 960.000 bytes refers to the whole error message including the sensor data transactions sampled at 5 kHz frequency for 4 seconds. While the first 480.000-byte part contains 2-second before the error, the second 480.000 byte is formed from the electrical measurements read in the 2-second period after the moment error occurred. The structure of an error message is also shown in the Figure 2.2 for the sake of clarity.

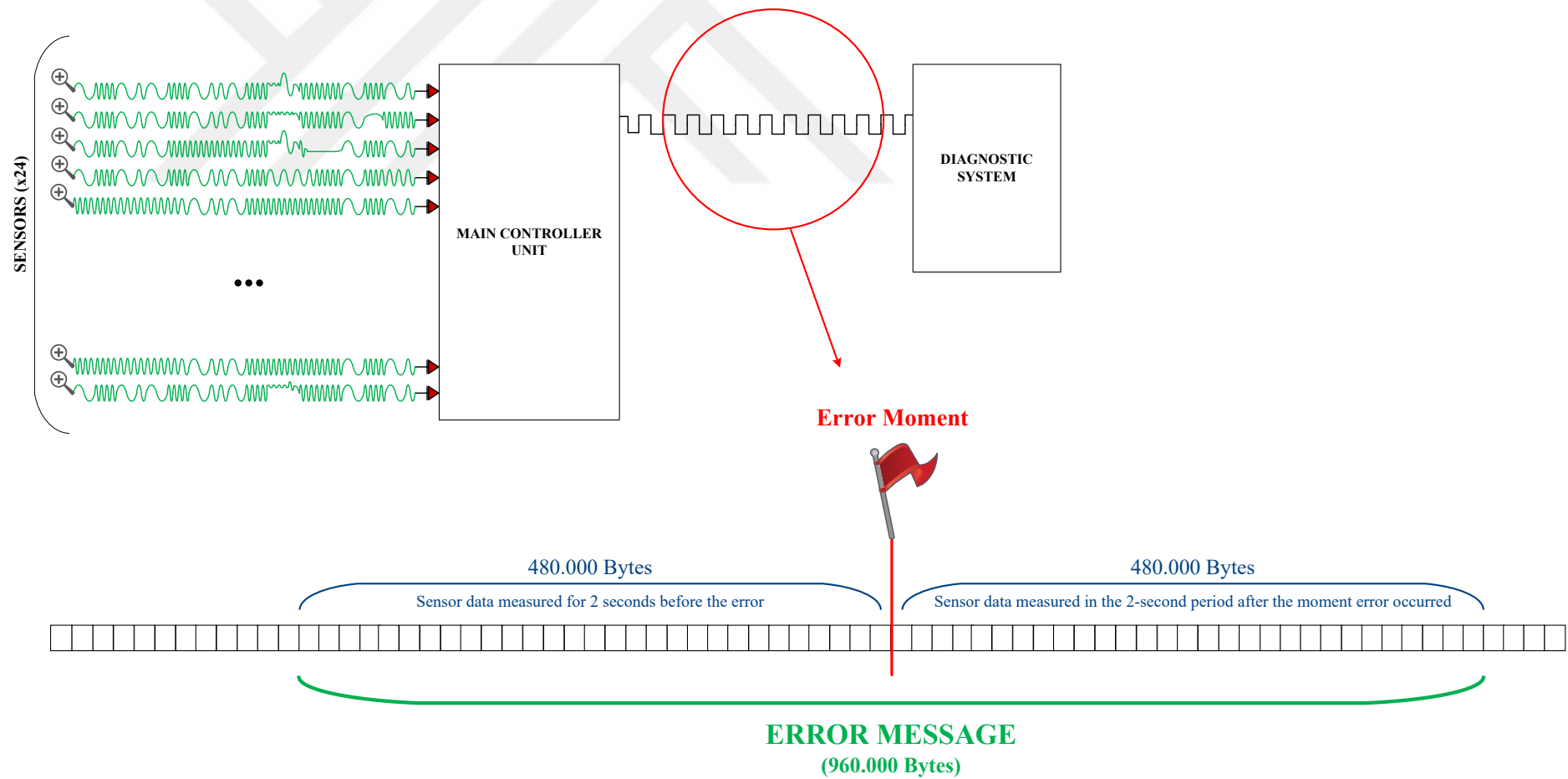


Figure 2.2 : Structure of an error message.

Lastly, the protocol to be used for communication within the FDS has been determined, since it is an important issue also for determination of the slave units to be used in the system. For this purpose, a synchronous serial data protocol that acts as an interface bus will be used. This protocol should also support high-speed applications and full-duplex communication, which means data can be sent and received simultaneously. As a result of the studies, it has been decided to use the SPI protocol for communication between the microprocessor and the slave units. The general outline of the SPI protocol is given below.

SPI is a synchronous data bus, which means that it uses separate lines for receiving and transferring data and a clock to keep both sides in perfect synchronization and also a line which for addressing functions. In total, the SPI bus will have a total of 4 lines which they use to communicate between the master and peripheral device which are:

- Master Data Output Slave Data Input (MOSI): is responsible for transmitting data from the master to the slave.
- Master Data Input Slave Data Output (MISO): is responsible to transmit data from the slave to the master.
- Clock Signal, generated by the master device (SCLK): is the clock signal is generated by the master to the slave to signal it to look at the data line to read data bits incoming.
- Chip Select, slave enabled signal controlled by the master device (SS / CSS / CS): is used as an addressing function. This signal is normally an active-low signal which is then pulled high to disconnect a slave from the SPI bus.

The SPI protocol first starts with the Master sending a clock signal and selecting a slave through the CS line. On the CS line, there are two ways in which the master can use to select a slave to receive or send data when there are multiple slaves involved. The way named Regular Independent Slave Configuration that will be used in the system designed in scope of the thesis is demonstrated in Figure 2.3.

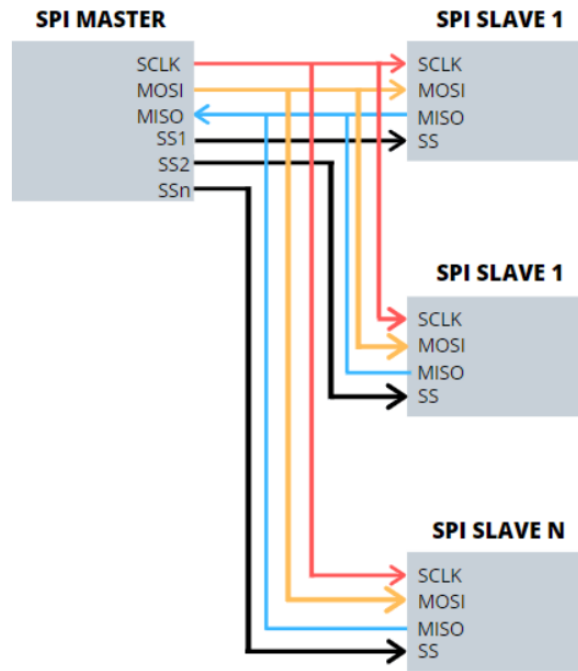


Figure 2.3 : SPI regular independent slave configuration.

As mentioned above, the SPI protocol first starts with the Master sending a clock signal and selecting a slave through the SS line. The master can select the clock polarity and clock phase using a specific SPI mode where each mode control whether data is shifted in and out on the rising or falling edge of the data clock signal (known as clock phase) and when the clock will be idle at either high or low. (known as clock polarity). By switching the clock polarity (CPOL) and clock phase (CPHA), it can form 4 unique modes to provide flexibility in communication between master and slave. Here is a table listing the four modes of transmission [15]:

Table 2.1 : SPI modes [15].

SPI Mode	CPOL	CPHA	Output Edge	Data Capture
0	0	0	Falling	Rising
1	0	1	Rising	Falling
2	1	0	Rising	Falling
3	1	1	Falling	Rising

Thus, the system design for the FDS was completed with the main lines, and detailed design studies were started. Both software and hardware designs were carried out within the scope of the related system. Hardware and software designs are explained in details in the relevant following sections.

2.1 Hardware Development Process

As explained in Chapter 2, all necessary theoretical calculations were made, scenarios were created, and the system level design was completed based on the works done. In this section, the components to be used in the FDS were selected based on the requirements emphasized and the decisions taken during the system design phase. Then, a Printed Circuit Board (PCB) was designed, which contains the recording units such as memories, super capacitors, and real-time clock/calendar unit of the system.

First of all, the units to be used in the Diagnostic PCB were selected. It was started with selecting the memory structure to be used for permanent error recording by taking the principles highlighted in system design phase as reference. Using a memory structure in rewriteable type of ROM for permanent recording was determined as the most suitable solution. At this stage, the memory architectures in ROM type were studied and using a NOR Flash memory which is actually one type of EEPROM has been decided. The corresponding integrated circuit is MT25QL128ABA Serial NOR Flash Memory manufactured by Micron Technology. The technical specifications that featured in choosing the relevant component are listed below:

- SPI-compatible serial bus interface.
- Non-volatile storage that retains its data when power is removed.
- Data retention: 20 years (TYP).
- Rewriteable but requires erase operation before writing (With several erase capability).
- 128Mb density.
- Single and double transfer rate (STR/DTR).
- Conformity with industrial conditions such as operating temperature range, humidity.
- 133 MHz Maximum Clock Frequency [16].

Secondly, the memory structure to be placed for real time recording was chosen according to the requirements and decisions emphasized in high-level system design phase. It was determined to use a RAM type memory structure whose data can be read and written any number of times as the most suitable solution before. In the scope

of the studies performed in this section, the memory architectures in RAM type were researched and using an MRAM memory has been decided. The corresponding integrated circuit is MR25H40 Magnetoresistive RAM (MRAM) memory manufactured by Everspin Technologies. The technical specifications having the impact on selecting the relevant component are given below:

- Unlimited write endurance.
- Fast, simple SPI interface, up to 50 MHz clock rate with MR20H40.
- No write delays.
- Conformity with industrial conditions (Industrial (-40 to 85°C), Extended (-40 to 105°C), and AEC-Q100).
- Data retention greater than 20 years.
- Direct replacement for serial EEPROM, Flash, and FeRAM.
- 4Mb (512K x 8) Memory Size [17].

Lastly, the information about the microcontroller that the software developed in the thesis study will run on, is given. The related microcontroller is TMS320F28377D IC which is in the family of TMS320F2837xD Dual-Core Delfino Microcontrollers of Texas Instruments. The overview of the microprocessor is as follows:

The CPU is a 32-bit fixed-point processor. This device draws from the best features of digital signal processing; Reduced Instruction Set Computing (RISC); and microcontroller architectures, firmware, and tool sets. The CPU features include a modified Harvard architecture and circular addressing. The RISC features are single-cycle instruction execution, register-to-register operations, and modified Harvard architecture. The microcontroller features include ease of use through an intuitive instruction set, byte packing and unpacking, and bit manipulation. The modified Harvard architecture of the CPU enables instruction and data fetches to be performed in parallel. The CPU can read instructions and data while it writes data simultaneously to maintain the single-cycle instruction operation across the pipeline. The CPU does this over six separate address/data buses [18].

There are also various components used on the PCB designed such as real-time clock/calendar for providing the date and time information of error messages, super capacitors to supply power in case of a power failure to the components that can be

affected, connectors full-compatible for the physical connection to the main system, passive elements for power distribution etc. However, detailed information for these components are not given, owing to the fact that principal focus of the thesis is the main functions of FDS such as real-time recording, error recording and error transfer. The block diagram of the Diagnostic PCB is given in Figure 2.4:

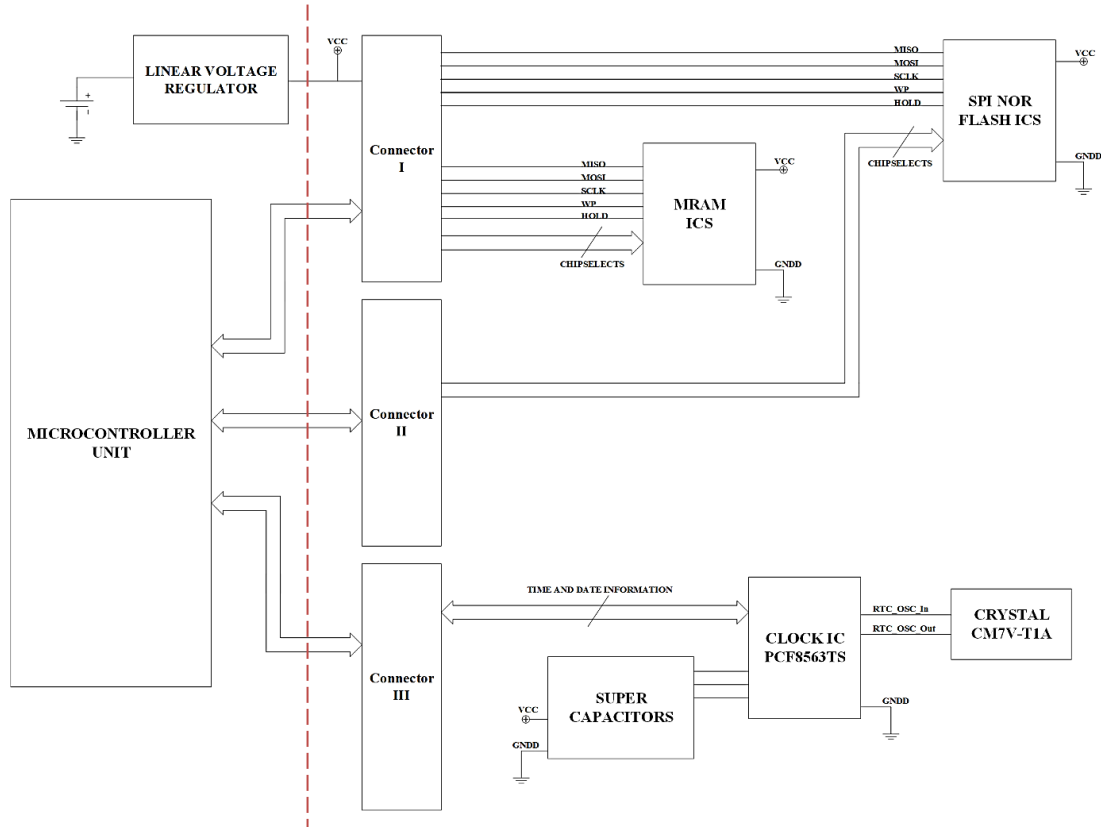


Figure 2.4 : Block diagram of the diagnostic PCB.

In the final part of the hardware design phase, a PCB was designed in Altium PCB Design Software & Tools.

After the chosen components were defined, the schematic and layout were designed respectively. The other components such as decoupling capacitors, pull-up resistors were also chosen according to the numerical calculations and recommended information given in datasheets. After PCB was manufactured and all components were soldered on the PCB, the hardware verification tests were performed. Herewith, the system was made ready for software development processes and software studies in C programming language were started.

2.2 Software Development Process

Firstly, software architecture was built for the main function and the architectural design was made based on the scenario determined in the design section. Then, the related design was divided into sub-functions and these sub-functions of the FDS created within the scope of the thesis were designed and verified one by one. These sub-functions are directly functions of MRAM and EEPROM ICs. In the end of this section, the main software was written by integrating the software of these sub-functions designed.

2.2.1 MRAM write & read functions

As mentioned in previous chapters, real time error recording processes are realized by using MRAM ICs. First step to do this is realizing write and read functions for MRAMs accurately. In this section, write and read processes were investigated for MRAM ICs and write and read functions were created respectively.

The Write Data Bytes (WRITE) command allows data bytes to be written starting at an address specified by the 24-bit address. Only address bits 0-18 are decoded by the memory. The data bytes are written sequentially in memory until the write operation is terminated by bringing $\overline{CS}$ high. The entire memory can be written in a single command. The address counter will roll over to 0000H when the address reaches the top of memory.

Unlike EEPROM or Flash Memory, MRAM can write data bytes continuously at its maximum rated clock speed without write delays or data polling. Back to back WRITE commands to any random location in memory can be executed without write delay. MRAM is a random access memory rather than a page, sector, or block organized memory so it is ideal for both program and data storage.

The WRITE command is entered by driving $\overline{CS}$ low, sending the command code, and then sequential write data bytes. Writes continue as long as the memory is clocked. The command is terminated by bringing $\overline{CS}$ high [17].

It should be emphasized that MRAM executes the writing process as words referring to fixed 16-bit data size. The writing data bytes procedure of the MRAM is shown in Figure 2.5 below.

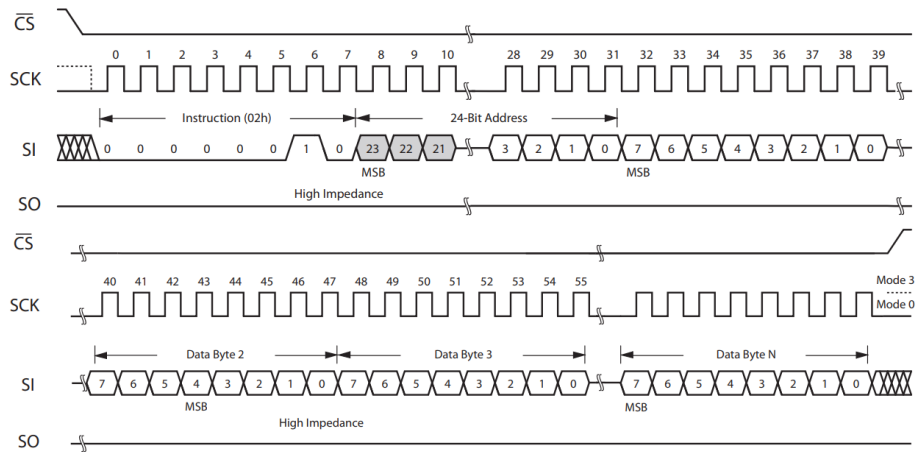


Figure 2.5 : Write data bytes (WRITE) timing [17].

According to the information emphasized in datasheet of MRAM ICs and explained above, C codes for MRAM word writing on SPI protocol were written. The base algorithm is demonstrated by using a flowchart and given in Figure 2.6.

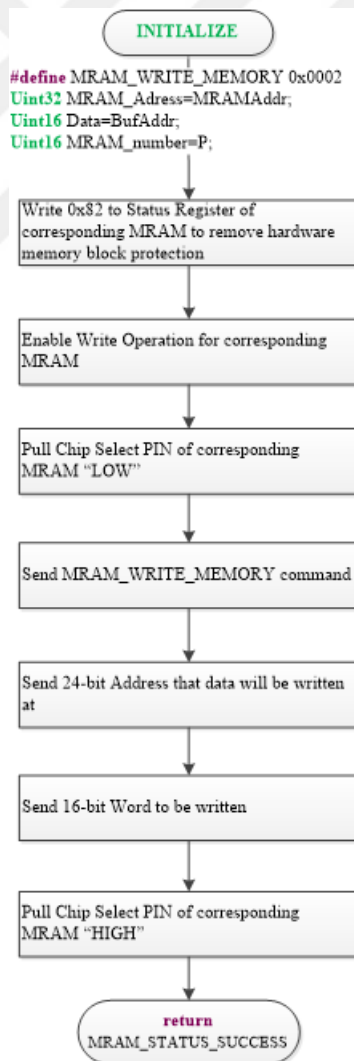


Figure 2.6 : MRAM write function flowchart.

In the first step, necessary initializations are made for the expressions that will be used in SPIMRAM write function. MRAMAddr to specify the addresses to be written, BufAddr for the data value to be written, and P to determine which MRAM IC will be active slave that writing will be executed, are used. In initialization step, that variables and WRITE command which is equal to 0x002 are assigned.

Even if master sends WRITE command to the slave, write cannot be executed because of memory protection property of the MRAM IC. Therefore, this protection must be removed before starting the write data procedure. The status register of the MRAMs is used to determine the modes. The status register consists of the 8 bits listed in Table 2.2. The Status Register Write Disable bit (SRWD) is used in conjunction with bit 1 (WEL) and the Write Protection pin ($\overline{WP}$) to provide hardware memory block protection as seen in Table 2.3. Bits BP0 and BP1 define the memory block arrays that are protected as described in Table 2.2. The fast writing speed of the MR2xH40 does not require write status bits. The state of bits 6, 5, 4, and 0 can be user modified and do not affect memory operation. All bits in the status register are pre-set from the factory in the “0” state [17].

Table 2.2 : Status register bit assignments [17].

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SRWD	Don't Care	Don't Care	Don't Care	BP1	BP0	WEL	Don't Care

When WEL is reset to 0, writes to all blocks and the status register are protected. When WEL is set to 1, BP0 and BP1 determine which memory blocks are protected. While SRWD is reset to 0 and WEL is set to 1, status register bits BP0 and BP1 can be modified. Once SRWD is set to 1, $\overline{WP}$ must be high to modify BP0 and BP1.

Table 2.3 : Memory protection modes [17].

WEL	SRWD	$\overline{WP}$	Protected Blocks	Unprotected Blocks	Status Register
0	X	X	Protected	Protected	Protected
1	0	X	Protected	Writable	Writable
1	1	Low	Protected	Writable	Protected
1	1	High	Protected	Writable	Writable

As it is seen in Table 2.3 SRWD and WEL bits must be “1” to make corresponding address writable. So, the code that must be send according to the Status Register Bit

Assignments “10000010” which is equal to 0x82 in hexadecimal notation. The other steps are created in conformity with writing data bytes procedure of the MRAM shown in Figure 2.5 and explained in software flowchart given in Figure 2.6. The only important matter here that should be emphasized that writing and reading processes are executed as bytes for MRAM normally. However, this operations should be executed as words in scope of the study. So, to accomplish this function in words format, microprocessor buffers are used and data is split into two bytes in writing operation, and concatenated two bytes into a word in reading function. In both splitting and concatenation, the most significant byte is processed first and the least significant byte second. Afterward write function was completed, read function for MRAM was created in the same way. First of all, the information emphasized for read process of MRAM ICs was followed to obtain the C code.

The Read Data Bytes (READ) command allows data bytes to be read starting at an address specified by the 24-bit address. Only address bits 0-18 are decoded by the memory. The data bytes are read out sequentially from memory until the read operation is terminated by bringing $\overline{CS}$ high. The entire memory can be read in a single command. The address counter will roll over to 0000H when the address reaches the top of memory.

The READ command is entered by driving $\overline{CS}$ low and sending the command code. The memory drives the read data bytes on the SO pin. Reads continue as long as the memory is clocked. The command is terminated by bringing $\overline{CS}$ high [17].

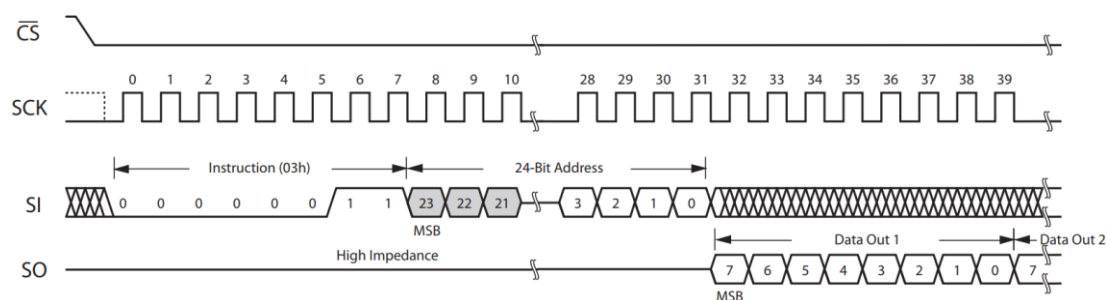


Figure 2.7 : Read data bytes (READ) timing [17].

The algorithm that was constituted by taking Figure 2.7 as reference is demonstrated by using a flowchart and given in Figure 2.8.

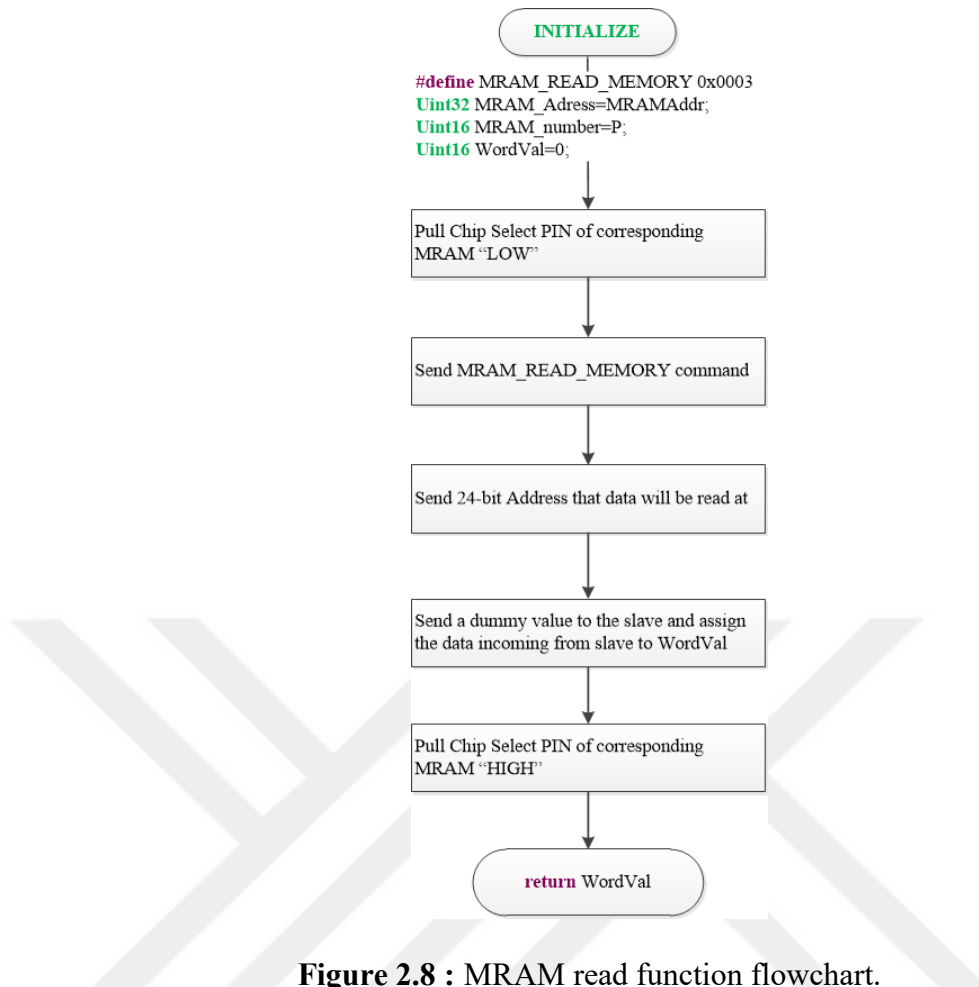


Figure 2.8 : MRAM read function flowchart.

The most important difference in MRAM read function by comparison with WRITE is that there is no need for removing memory protection property of the MRAM IC for reading. So, memory protection steps are deleted from the process. In the second place, MRAM read memory command is sent instead of write command. After read command is sent, dummy bytes are sent to trigger reading. The dummy bytes are required for clocking the reply back in, since the device that provides the clock is always master. In other words, dummy bytes are used to generate the clock signal needed for reading.

2.2.2 MRAM continuous write functions

As calculated in the system design phase, an error signal size of 980.000 byte consisting of 24 different sensor data will be recorded within the scope of the thesis. While the first 480.000-byte signal comprises of 2-second data before occurrence of the error, the second 480.000 byte consists of the 2 seconds after occurrence of the error. The moment that error occurred will be detected by using the error flag.

In this context, it has been decided to create separated functions that automatically generate these data and save to the corresponding MRAM. In this section, these functions named MRAM_Continuous_Write_Functions are explained. The base algorithm is given by using a flowchart given in Figure 2.9.

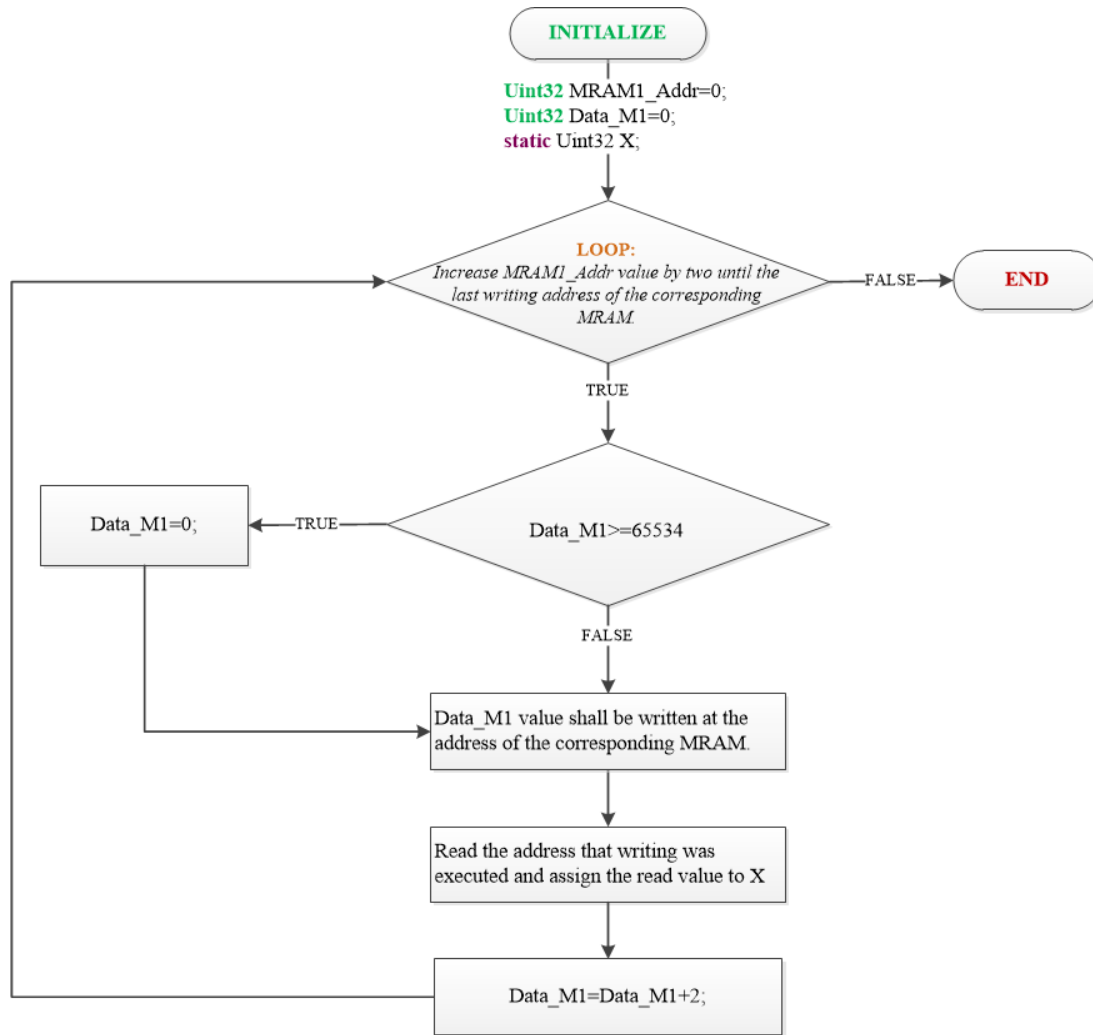


Figure 2.9 : MRAM continuous write function flowchart.

In the first step, necessary initializations are made for the integers that would be used in MRAM continuous write function. MRAM1_Addr to determine the 1st MRAM's addresses to be written, and Data_M1 for the data value to be written at MRAM1 are used. Since the first signal part, 2-second data before the error occurs, is 480.000 byte, MRAM1_Addr value is increased two by two until 480000. The increment for Data_M1 is also the same. However, because READ and WRITE processes are executed as words and a word could be maximally 1111111111111111 which is equal to 65535 in decimal, Data_M1 value is reset at 65534 for sake of the ease of trace. In

addition, a static integer named X is created and also read function for every address that are written, run after writing process is completed. The read values are assigned to X, thus the writing success is checked. When an error occurs, the code flow exits out of the loop and function is completed.

In total 2 MRAM continuous write functions are used, and the 2nd function which is for MRAM2 is also the same functionally. The only differences are that; the commands and integers are updated to specify the 2nd MRAM. The code flow always exits out of the loop, when MRAM2_Addr value reaches 480000. In other words, the function is completed after the 480000th byte is written, for the reason that data size to be written after the error moment is always fixed.

Since there is no functional difference, MRAM2_Continuous_Write function is not explained in this section. However, both of them were tested and the results are included for both functions in the corresponding section, because two functions were used separately in final scenario and run by using different hardware.

2.2.3 EEPROM write & read functions

As stated in previous sections, the error data which consist of the real-time recorded sensor data by the MRAM integrated circuits will be transferred to the FLASH integrated circuits on error condition. Thus, high storage capacity (density) and automatic data protection during a power failure properties of FLASH ICs are utilized. In this regard, realizing the write and read functions for FLASHs accurately is another milestone of the thesis. Write and read processes are investigated for FLASH ICs, and their functions are created respectively in this section. The “program” word is used instead of write, since this term is more commonly used for FLASH memories.

Before a PROGRAM command is initiated, the WRITE ENABLE command must be executed to set the write enable latch bit to 1. To initiate a command, S# is driven LOW and held LOW until the eighth bit of the last data byte has been latched in, after which it must be driven HIGH. If S# is not driven HIGH, the command is not executed, flag status register error bits are not set, and the write enable latch remains set to 1. Each address bit is latched in during the rising edge of the clock. When a command is applied to a protected sector, the command is not executed, the write enable latch bit remains set to 1, and flag status register bits 1 and 4 are set. If the operation times out, the write enable latch bit is reset and the program fail bit is set to 1 [16].

Note: The manner of latching data shown and explained in the timing diagrams ensures that the number of clock pulses is a multiple of one byte before command execution, helping reduce the effects of noisy or undesirable signals and enhancing device data protection. The PROGRAM operations are given in Table 2.4 [16].

Table 2.4 : FLASH memory PROGRAM operations [16].

Operation Name	Description/Conditions
PAGE PROGRAM (02h)	A PROGRAM operation changes a bit from 1 to 0.
DUAL INPUT FAST PROGRAM (A2h)	When the operation is in progress, the write in progress bit is set to 1. The write enable latch bit is cleared to 0, whether the operation is successful or not. The status register and flag status register can be polled for the operation status. When the operation completes, the write in progress bit is cleared to 0. An operation can be paused or resumed by the PROGRAM/ERASE SUSPEND or PROGRAM / ERASE RESUME command, respectively.
EXTENDED DUAL INPUT FAST PROGRAM (D2h)	
QUAD INPUT FAST PROGRAM (32h)	
EXTENDED QUAD INPUT FAST PROGRAM (38h)	
	If the bits of the least significant address, which is the starting address, are not all zero, all data transmitted beyond the end of the current page is programmed from the starting address of the same page. If the number of bytes sent to the device exceed the maximum page size, previously latched data is discarded and only the last maximum page-size number of data bytes are guaranteed to be programmed correctly within the same page. If the number of bytes sent to the device is less than the maximum page size, they are correctly programmed at the specified addresses without any effect on the other bytes of the same page.

As seen in Table 2.4, there are several programming methods for the FLASH ICs. When the related types of programming are analyzed, it is preferred to use the PAGE PROGRAM operation in order to obtain a compatible writing process with the processes made for MRAM and to use the simplest method that satisfies the needs. Operation timing of the PAGE PROGRAM command is given in Figure 2.10.

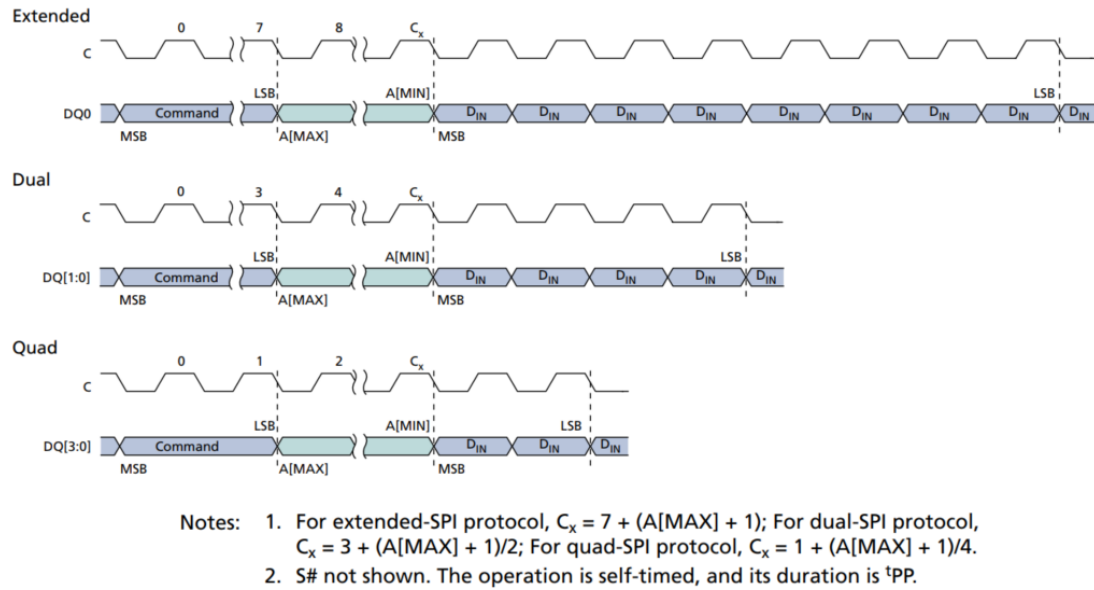


Figure 2.10 : FLASH IC page program operation timing [16].

Nonetheless, the WRITE ENABLE command must be executed to set the write enable latch bit to 1 before a PROGRAM command is initiated. That is to say, writing will not be performed, if the WRITE ENABLE command is not executed. Moreover, to execute the PROGRAM command, chip select pin which is labelled as S# in the component datasheet should be driven LOW and held LOW until the eighth bit of the last data byte has been latched in, after which it must be driven HIGH.

According to the information emphasized in datasheet of FLASH ICs and explained above, C codes for FLASH word writing on SPI protocol were written. The base algorithm is demonstrated by using a flowchart and given in Figure 2.11.

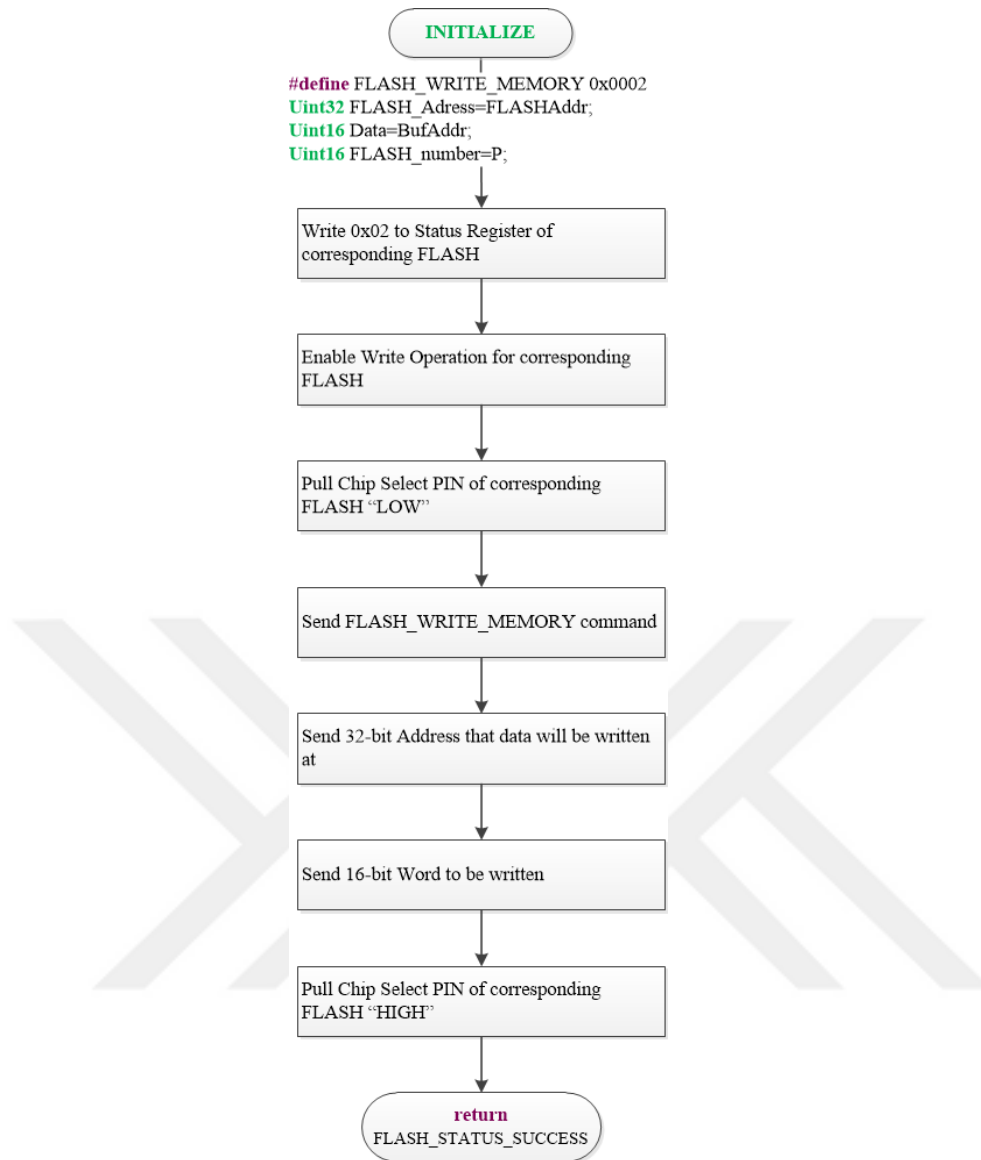


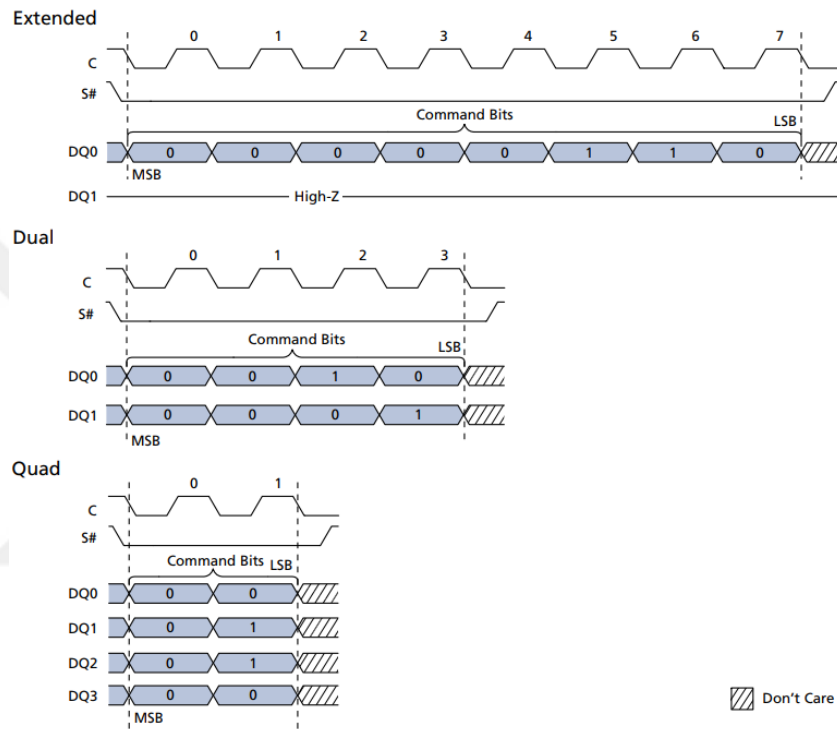
Figure 2.11 : FLASH write function flowchart.

In the first step, necessary initializations are made for the expressions that will be used in SPIFLASH write function. FLASHAddr to specify the addresses to be written, BufAddr for the data value to be written, and P to determine which FLASH IC will be active slave that writing will be executed to, are used.

To initiate a command, S# is driven LOW and held LOW until the eighth bit of the command code has been latched in, after which it must be driven HIGH. For extended-, dual-, and quad-SPI protocols respectively, the command code is input on DQ0, DQ [1:0], and DQ [3:0]. If S# is not driven HIGH after the command code has been latched in, the command is not executed, flag status register error bits are not set, and the write enable latch remains cleared to its default setting of 0, providing protection against errant data modification [16].

Table 2.5 : WRITE ENABLE/DISABLE operations [16].

Operation Name	Description/Conditions
WRITE ENABLE (06h)	Sets the write enable latch bit before each PROGRAM, ERASE, and WRITE command.
WRITE DISABLE (04h)	Clears the write enable latch bit. In case of a protection error, WRITE DISABLE will not clear the bit. Instead, a CLEAR FLAG STATUS REGISTER command must be issued to clear both flags.



Note: 1. WRITE ENABLE command sequence and code, shown here, is 06h (0000 0110 binary); WRITE DISABLE is identical, but its command code is 04h (0000 0100 binary).

Figure 2.12 : WRITE ENABLE and WRITE DISABLE timing [16].

Secondly, the write enable latch must be set before program the FLASH. Otherwise, operation cannot be performed because of the WRITE DISABLE attribute even if master sends PROGRAM command to the slave correctly. For this reason, the write enable operation are executed by considering the guidelines in Table 2.5 and Figure 2.12 before starting the programming.

The other steps were created in conformity with programming data bytes procedure of the FLASH IC explained before. The one significant point is that writing and reading processes are identified as bytes for FLASH also. However, this operations are executed as words also for FLASH in scope of the study. So, the same method, which

is that data is split into two bytes in writing operation and concatenated two bytes into a word in reading function with microprocessor buffers, is used. In both splitting and concatenation, the most significant byte is processed first and the least significant byte as second.

Afterward write function was completed, read function for FLASH was created in the same way. First of all, the information emphasized for read process of MRAM ICs is examined to obtain the C code:

To initiate a command, S# is driven LOW and the command code is input on DQn, followed by input of the address bytes on DQn. The operation is terminated by driving S# HIGH at any time during data output [16].

Table 2.6 : READ MEMORY operations [16].

Operation Name	Description/Conditions
READ (03h)	The device supports 3-byte addressing (default), with A[23:0] input during address cycle. After any READ command is executed, the device will output data from the selected address. After the boundary is reached, the device will start reading again from the beginning.
FAST READ (0Bh)	
DUAL OUTPUT FAST READ (3Bh)	
DUAL INPUT/OUTPUT FAST READ (BBh)	
QUAD OUTPUT FAST READ (6Bh)	Each address bit is latched in during the rising edge of the clock. The addressed byte can be at any location, and the address automatically increments to the next address after each byte of data is shifted out; therefore, a die can be read with a single command.
QUAD INPUT/OUTPUT FAST READ (EBh)	
DTR FAST READ (0Dh)	FAST READ can operate at a higher frequency (fC).
DTR DUAL OUTPUT FAST READ (3Dh)	
DTR DUAL INPUT/OUTPUT FAST READ (BDh)	DTR commands function in DTR protocol regardless of settings in the nonvolatile configuration register or enhanced volatile configuration register; other commands function in DTR protocol only after DTR protocol is enabled by the register settings.
DTR QUAD OUTPUT FAST READ (6Dh)	
DTR QUAD INPUT/OUTPUT FAST READ (EDh)	E7h is similar to the QUAD I/O FAST READ command except that the lowest address bit (A0) must equal 0 and only four dummy clocks are required prior to the data output. This command is supported in extended SPI and quad-SPI protocols, but not in the DTR protocol; it is ignored in dual-SPI protocol.
QUAD INPUT/OUTPUT WORD READ (E7h)	

As seen in Table 2.6, there are also several reading operations of the FLASH ICs. When the related types are analyzed, it is preferred to use the READ operation with 03h in hexadecimal command in order to obtain a compatible reading process with MRAM and to use the simplest method that satisfies the needs. Operation timing of the READ command is given in Figure 2.13.

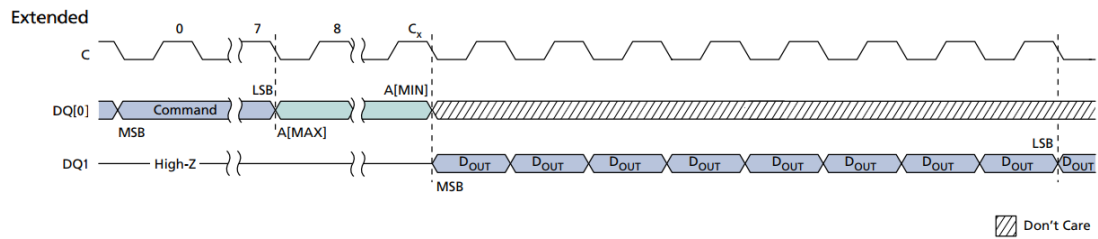


Figure 2.13 : READ timing [16].

The algorithm that was obtained by taking Figure 2.13 as reference is demonstrated by using a flowchart and given in Figure 2.14.

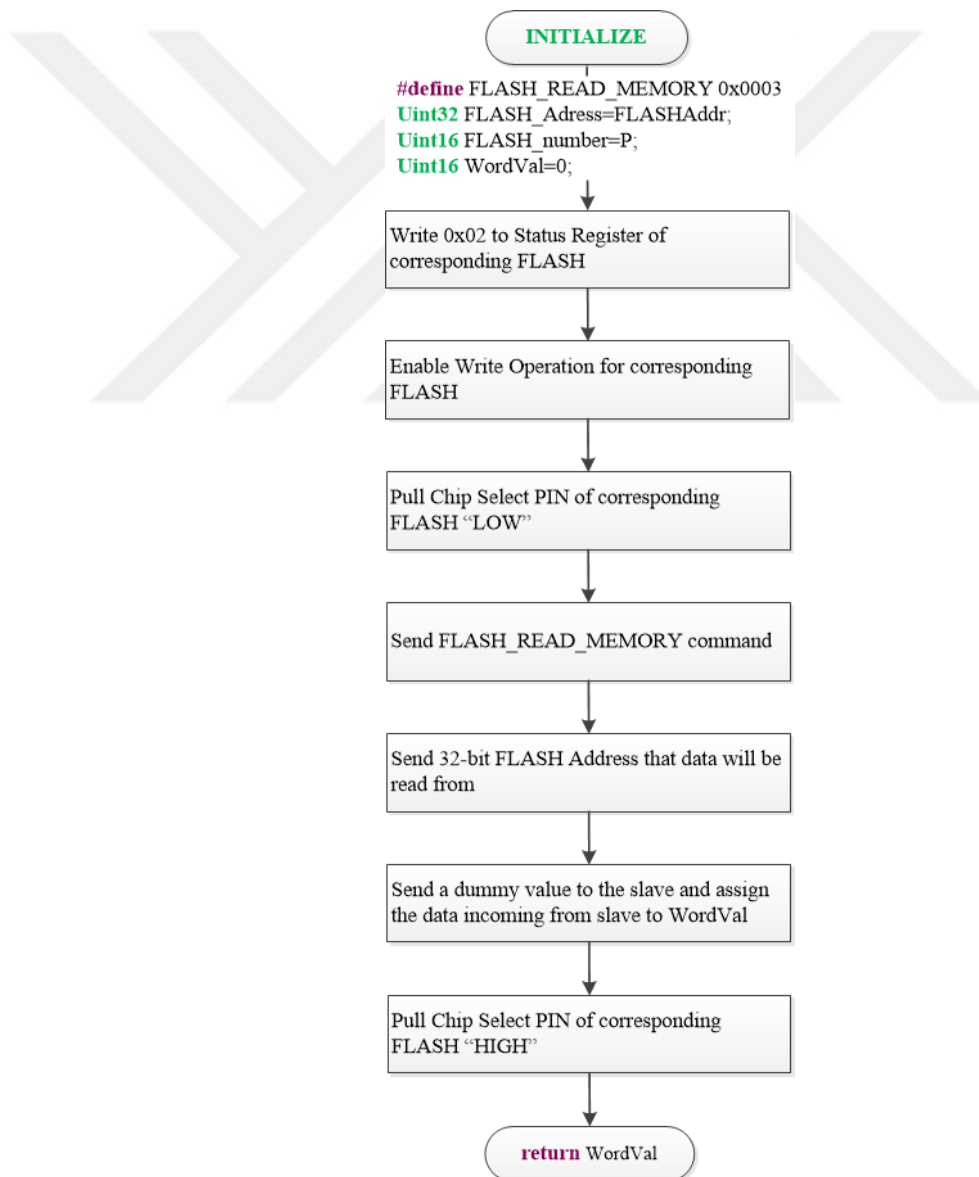


Figure 2.14 : FLASH read function flowchart.

In the read function, FLASH read memory command is sent instead of write command. After read command is sent, dummy bytes (0x0000) which are required to generate the clock signal are sent by triggering the master.

In addition, the read value is assigned to the return value of the function. Thus, the values are read and checked through the CCS interface.

2.2.4 EEPROM erase functions

Unlike MRAM devices, it is not possible to overwrite or rewrite on addresses at FLASH memories. In other words, if a write operation has been performed to an address on the FLASH memory, the corresponding address must be deleted before a new write operation. At this point, there is also a need to create erase functions for FLASH devices differently from MRAM devices. In this context, erase operations for FLASH memories are examined, and different types of erasing functions are created in line with the device features and the project requirements. The information specified in the product datasheet of the FLASH devices is as follows:

An ERASE operation changes a bit from 0 to 1. Before any ERASE command is initiated, the WRITE ENABLE command must be executed to set the write enable latch bit to 1; if not, the device ignores the command and no error bits are set to indicate operation failure. S# is driven LOW and held LOW until the eighth bit of the last data byte has been latched in, after which it must be driven HIGH. The operations are self-timed, and duration is tSSE, tSE, or tBE according to command [16].

If S# is not driven HIGH, the command is not executed, flag status register error bits are not set, and the write enable latch remains set to 1. A command applied to a protected subsector is not executed. Instead, the write enable latch bit remains set to 1, and flag status register bits 1 and 5 are set [16].

When the operation is in progress, the program or erase controller bit of the flag status register is set to 0. In addition, the write in progress bit is set to 1. When the operation completes, the write in progress bit is cleared to 0. The write enable latch bit is cleared to 0, whether the operation is successful or not. If the operation times out, the write enable latch bit is reset and the erase error bit is set to 1. The status register and flag status register can be polled for the operation status. When the operation completes, these register bits are cleared to 1.

Note: For all ERASE operations, noisy or undesirable signal effects can be reduced and device data protection enhanced by holding S# LOW until the eighth bit of the last data byte has been latched in; this ensures that the number of clock pulses is a multiple of one byte before command execution [16].

Table 2.7 : FLASH memory ERASE operations [16].

Operation Name	Description/Conditions
SUBSECTOR ERASE (52h/20h)	Sets the selected subsector or sector bits to FFh. Any address within the subsector is valid for entry. Each address bit is latched in during the rising edge of the clock. The operation can be suspended and resumed by the PROGRAM/ERASE SUSPEND and PROGRAM/ERASE RESUME commands, respectively.
SECTOR ERASE (D8h)	
BULK ERASE (C7h/60h)	Sets the device bits to FFh. The command is not executed if any sector is locked. Instead, the write enable latch bit remains set to 1, and flag status register bits 1 and 5 are set.

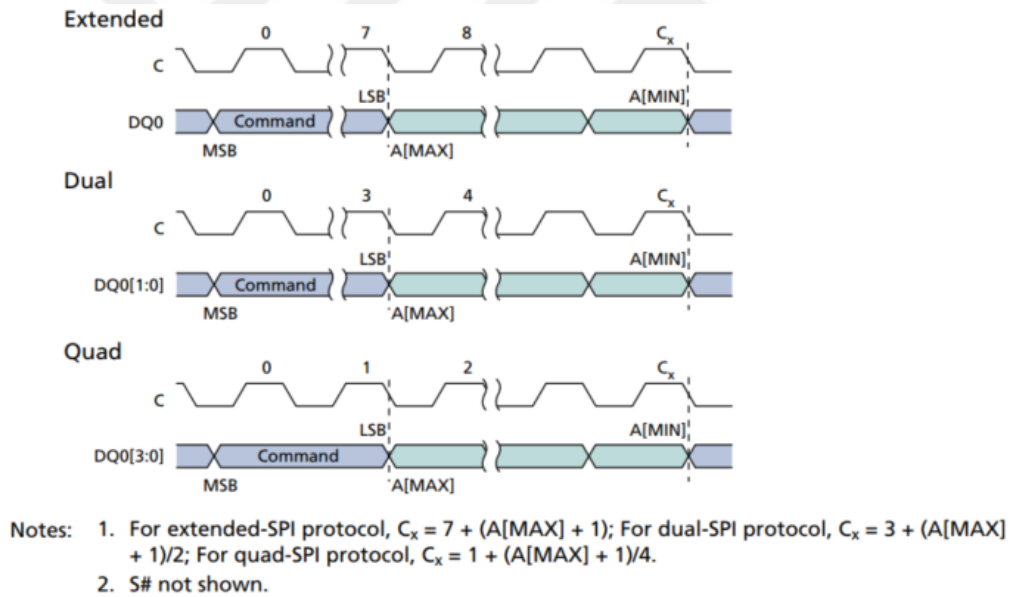
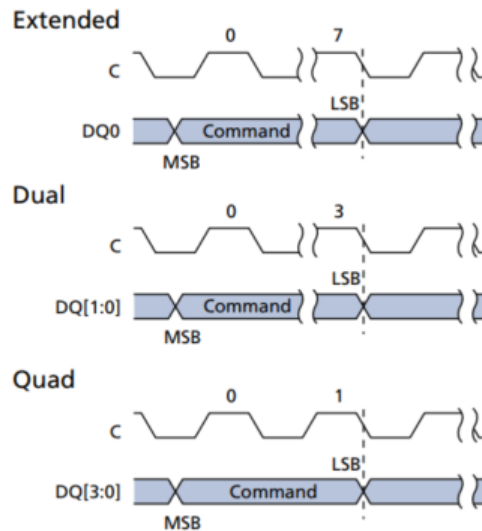


Figure 2.15 : SUBSECTOR and SECTOR ERASE timing [16].



Note: 1. S# not shown.

Figure 2.16 : BULK ERASE timing [16].

With the Bulk Erase function, data in all addresses of the corresponding EEPROM are deleted. This means that all 16MByte data is deleted.

Sector is 64KByte data fields (actually 65536 Byte, however it is considered 64Kbyte for safety reasons) customized in EEPROM. While it is enough to specify the relevant integrated in order to execute the Bulk Erase function, the EEPROM number and any address number within the sector to be deleted are required for erasing a sector. Hence, an address data within that sector need to be determined and sent in order to operate the Sector Erase Function.

According to the information given in datasheet of the corresponding FLASH memory, C codes for FLASH Erase Functions on SPI protocol were written.

Three different erasing functions were created for the FLASH memory device. Two of these are Sector Erase and Bulk Erase operations that are defined as provided erase features of the FLASH device. While creating these functions, the information given in Table 2.7, timings given in Figure 2.15 and 2.16 have been taken as the reference. The flowcharts of the functions are given below:

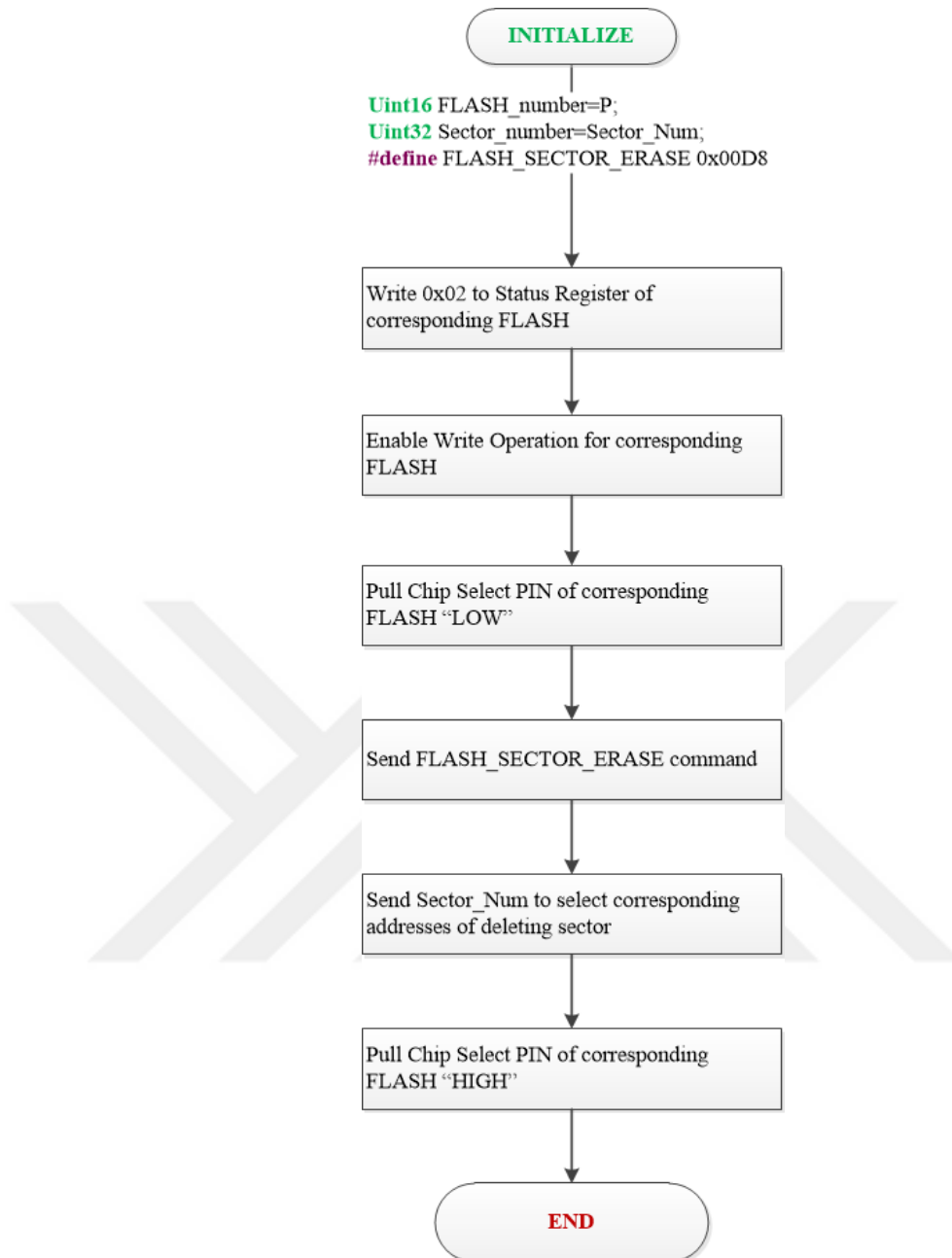


Figure 2.17 : FLASH sector erase function flowchart.

In the first step, necessary initializations are made for the expressions that will be used in SPICFLASH Sector Erase function. Sector_Num to specify the sector number to be erased, FLASH_SECTOR_ERASE for the command to be run, and P to determine which FLASH IC will be active slave that erasing will be executed to, are used.

The Bulk Erase function was also created with the same logic as sector erase. The only remarkable difference that should be specified is that there is no need to specify any memory block such as address, subsector or sector, since bulk erase means that all addresses of the corresponding FLASH memory are deleted.

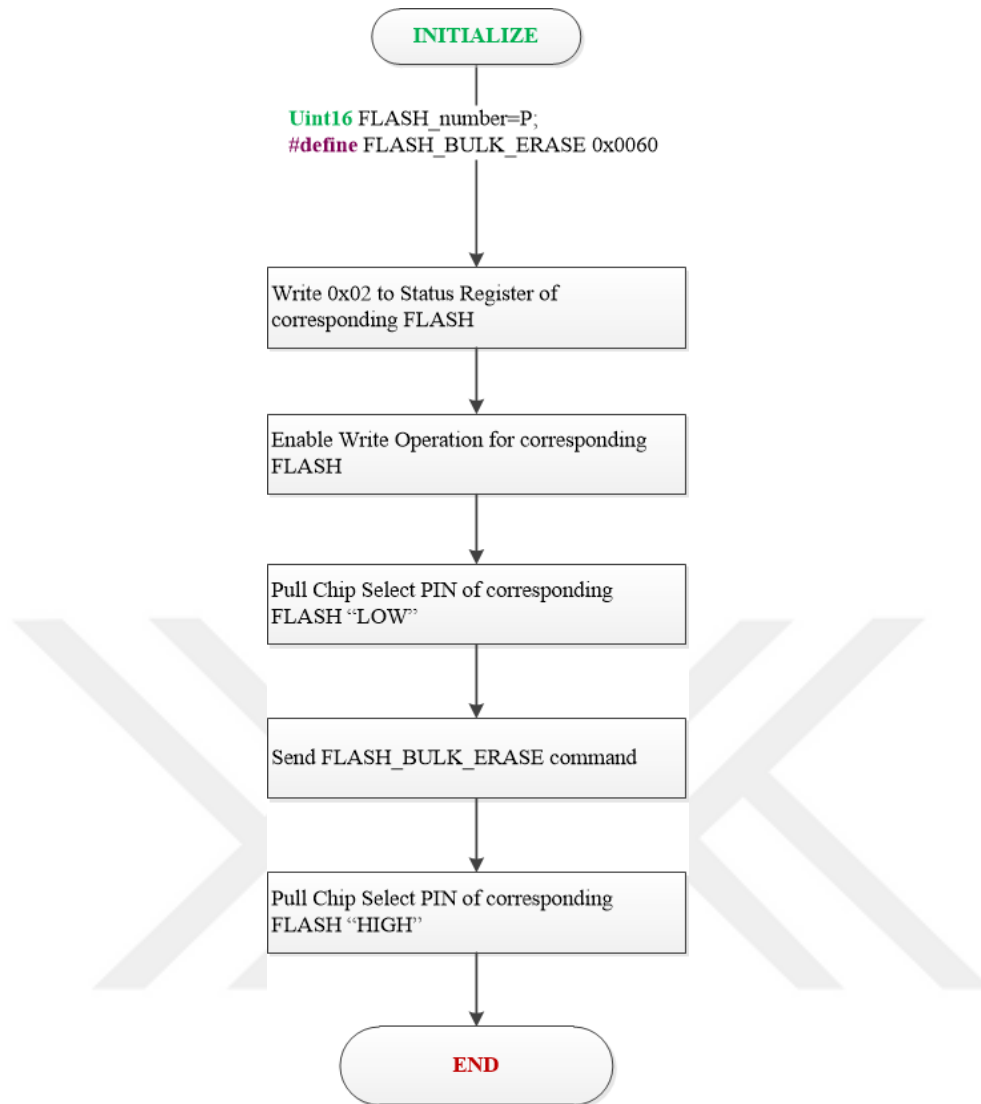


Figure 2.18 : FLASH bulk erase function flowchart.

As stated in the previous chapters, size of an error message is 960.000 bytes in total. The memory size needed for writing an error message will be named "Slot" after this point as well. This memory block allocated for writing an error message (a slot) has been selected as 1048576 byte in design stage for leaving a safety margin. Thus and so, the total 75 error messages can be recorded without any deletion in the FDS. Therefore, necessary memory size (next slot) must be erased before next write in case that 75 error messages are written. However, only the addresses within the corresponding slot must be erased not to cause a mistakenly deletion.

Due to these reasons, the need to create a new erase function (or operation) arose, and "Slot Erase" function was created for permanent error recording structure. The basic algorithm of slot erase function is demonstrated by using a flowchart and given in Figure 2.19.

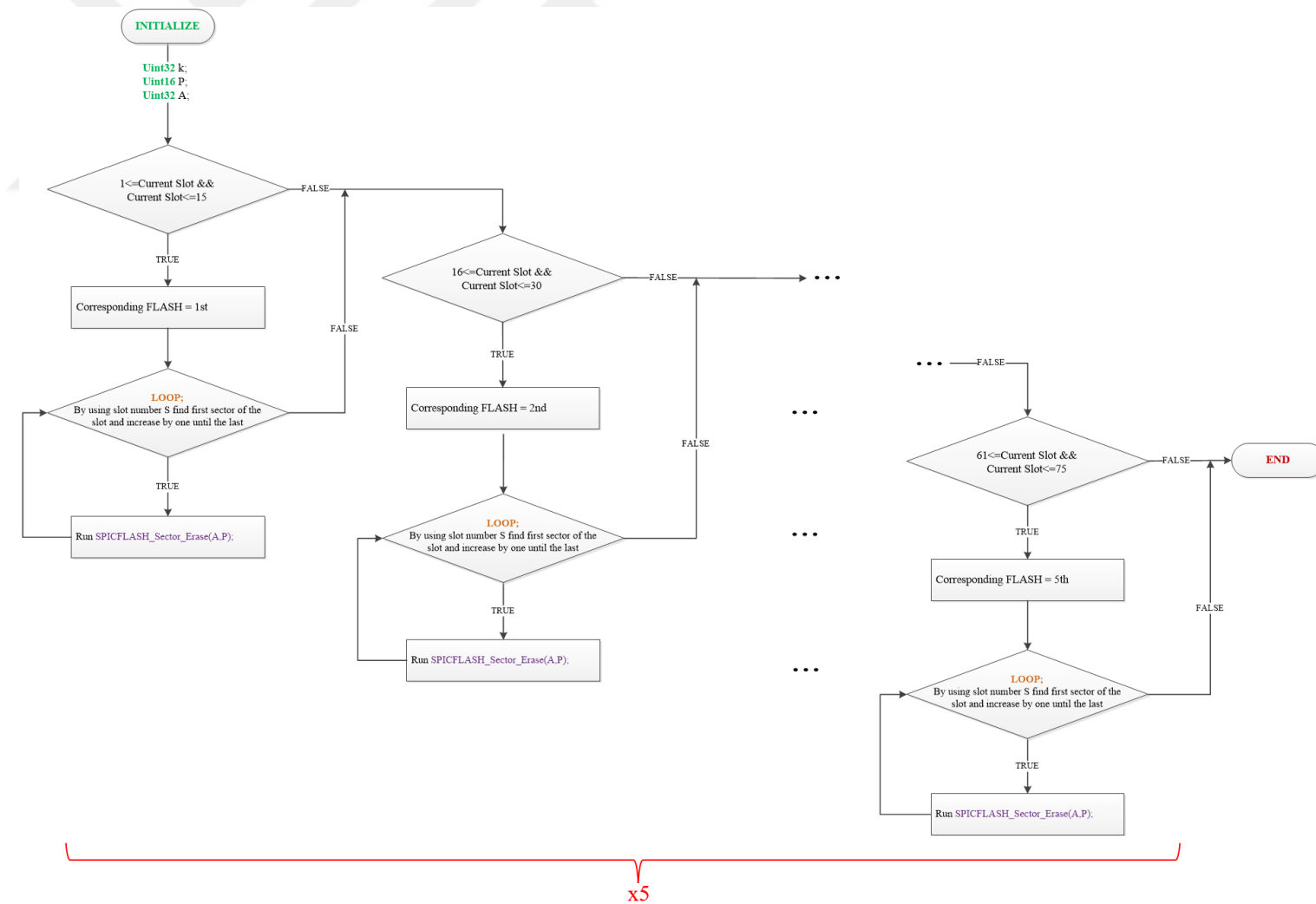


Figure 2.19 : FLASH slot erase function flowchart.

The slot erase function basically consists of parallel processes with 5 separate control steps. These checks are used to detect the relevant FLASH memory (the FLASH memory in which the slot is located) that the error message will be written to. Each FLASH memory device has enough capacity for 15 error messages. To illustrate, the memory space allocated for the 31st error message is the first slot of the 3rd FLASH memory. After the FLASH IC selection is made, the operations performed are logically the same. Only the numerical values used in the operations change depending upon the FLASH number and the corresponding slot addresses that error message will be written to. The operating logic of the algorithm is given below with an example:

For example, let a scenario that there is no unrecorded area or space in FLASH devices (all of the slots are full) and the next error message is the 24th be considered. In this case, the slot number to be erased is 24, and this slot is in the 2nd FLASH IC. As seen in Figure 2.19, the result of the first control loop will be FALSE and algorithm switches to the second one. In order to explain in more details, the steps after the FLASH IC selection are shown in Figure 2.20:

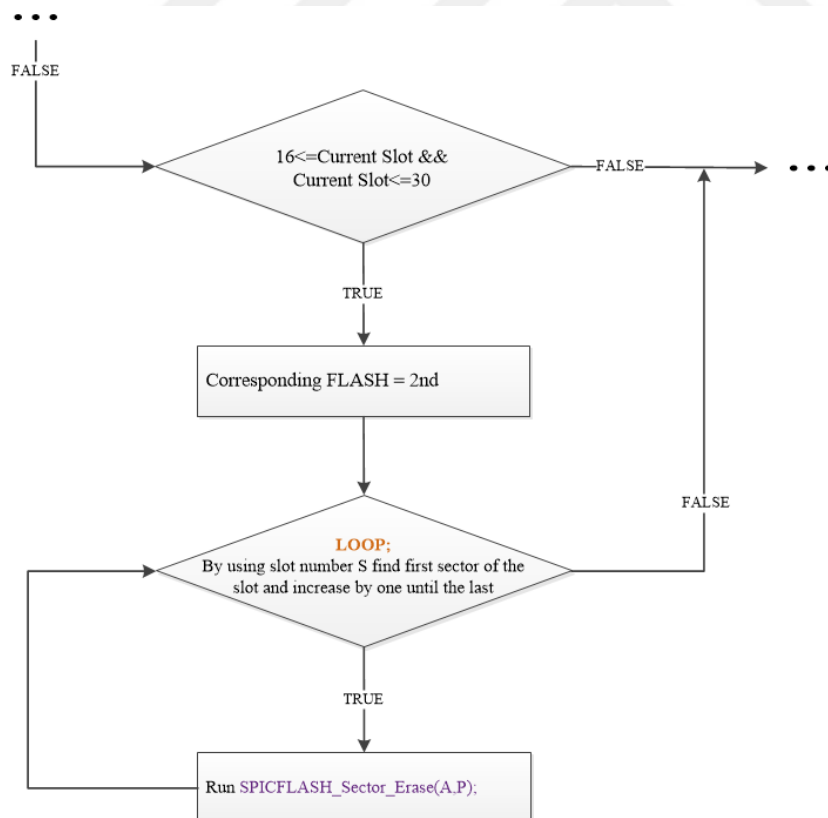


Figure 2.20 : 2nd parallel block of the FLASH slot erase function flowchart.

The slot number is 24 which is between 16 and 30, consequently, the result of the control loop will be TRUE and corresponding FLASH will be designated as 2nd. After this step, software is executed with the second decision structure that is actually a loop. The first sector of the slot is determined by using the slot number and it is erased. As stated above, an address data within that sector need to be determined and sent in order to operate the sector erase function. Hence, after sector assignment for slot erase, address assignment is also executed within the functionality of the sector erase. This assignation is done by executing the mathematical operations that were created with reference to the information that an EEPROM consists of 15 slots and a slot consists of 16 sectors. The erase operation is performed from first sector to last one. When the last sector of the slot is erased, the slot number is increased again. Hence, loop condition is not fulfilled anymore, and the function is completed.

In this section, three different types of deletion functions created for FLASH ICs in line with the device features and project requirements were explained in detail. These erase functions are also summarized with their general characteristics in Table 2.8.

Table 2.8 : FLASH erase functions created within the scope of the thesis.

Function	Bulk Erase	Sector Erase	Slot Erase
Erase Capacity	16MB	64KB	1024KB
Creation Way	According to information given in datasheet	According to information given in datasheet	Post-designed
Code	D8h	C7h/60h	-
Required Information	FLASH Number	FLASH Number Sector Number	Slot Number

2.2.5 EEPROM – MRAM communication function

As stated in the previous sections, in case of an event specified as an error, a 4-second error message which total size of 960.000 bytes will be transferred from MRAMs to the corresponding EEPROM. In this respect, the design of a communication function responsible for transferring data between EEPROM and MRAMs is one of the most important steps in the design of the main function. The tasks to be performed by the EEPROM – MRAM Communication function are briefly as follows:

- Read the slot number which the last error was written in.
- Determine the next slot number according to the slot number.
- Detect the corresponding EEPROM according to the next slot number.
- Delete the slot of the corresponding EEPROM where the error will be recorded into.
- Determine the first and last EEPROM addresses where the error will be written by using the current slot number.
- Read the relevant twenty-four sensor data from MRAMs and write to the corresponding EEPROM addresses.
- Increase the slot number by one and save to the allocated address.
- Reset the corresponding counters and flags.

In the light of information given above, the EEPROM – MRAM Communication function was designed in C programming language.

In the first step of the software, necessary initializations are made for the integers that will be used in function. The function starts with determining the corresponding slot and erase it for new recording. When the relevant slot is empty, software continues by calculating the starting address of the EEPROM. After this step, the sensor values are read from MRAMs and written to the corresponding EEPROM from the starting address to the last one by using a loop. The code flow exits out of the loop when all of the 960.000 bytes are transmitted to permanent error recording structure (EEPROM). The corresponding counters and flags are reset and the process are completed. The base algorithm is demonstrated by using the flowchart given in Figure 2.21.

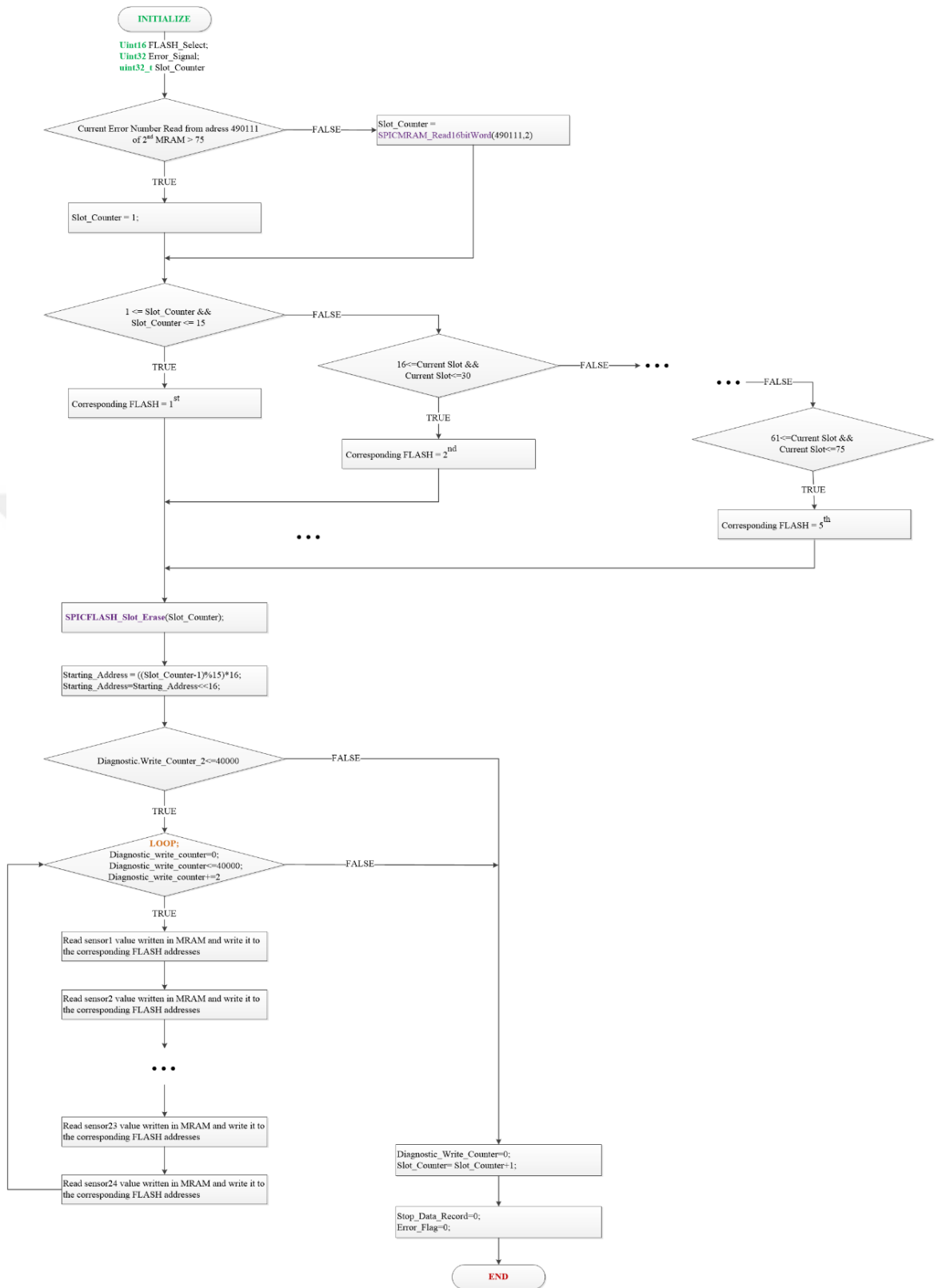


Figure 2.21 : EEPROM – MRAM communication function flowchart.

2.2.6 Final system architecture & design specification

In this section, the main software was designed by using the functions written up to this section. Firstly, the final diagnostic scenario was created based on the tasks that will be executed by the FDS within the scope of the thesis. These tasks are given briefly below:

- Check real time error recording command and error status continuously.
- Read 24 sensors' data continuously and save to MRAM continuous recording structure.
- In case of error, record the sensors' data 2-second more.
- Generate the Error Message in MRAMs by using 2-second data before and 2-second data after the moment of error occurrence.
- Transfer the Error Message from MRAMs to the relevant EEPROM when the 4-second error recording completed.
- Reset the error flag and reactivate MRAM real time data recording.

In the light of information given above, the main software has been designed in C programming language. The base algorithm is demonstrated by using the flowchart given in Figure 2.22.

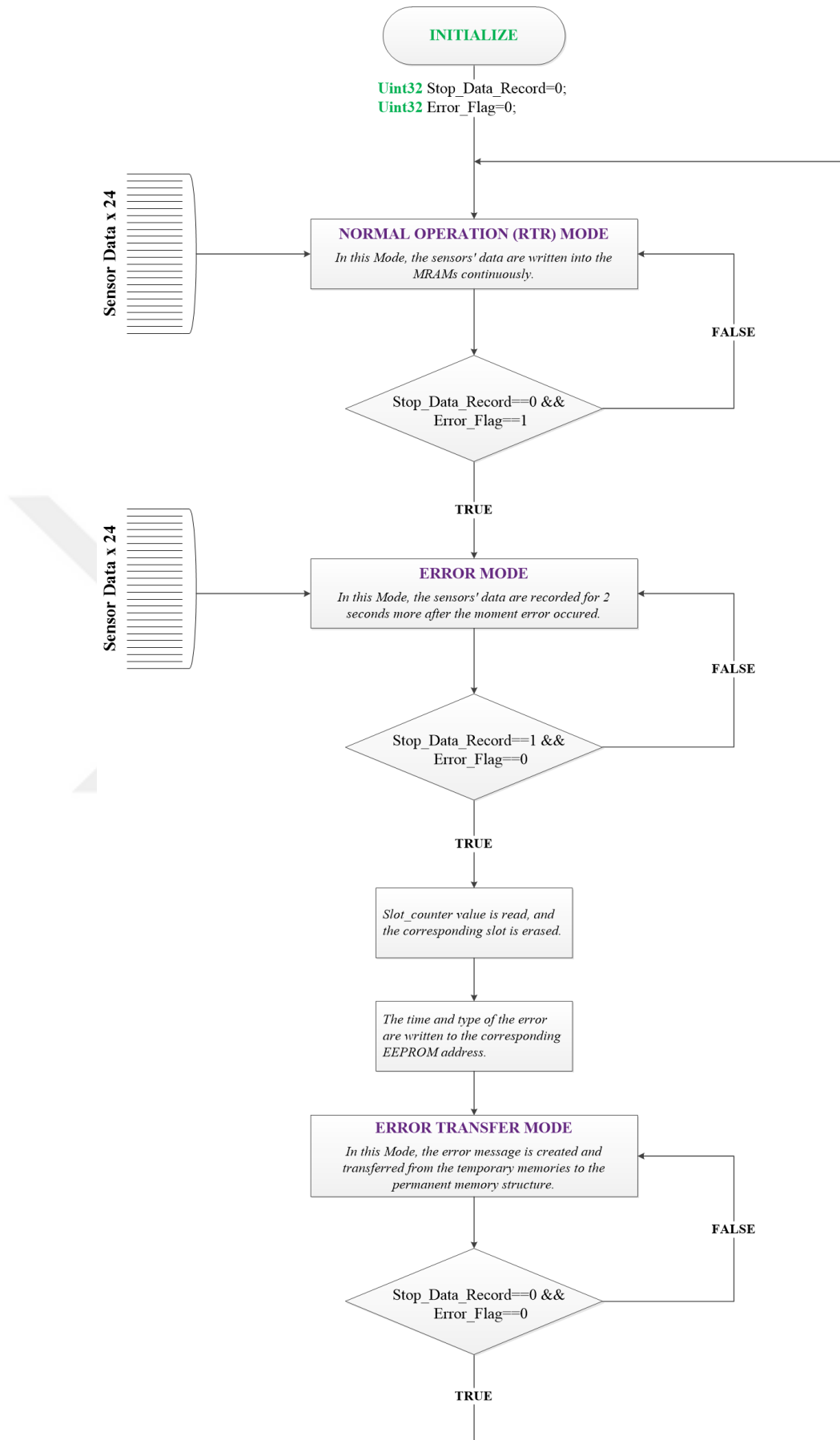


Figure 2.22 : The main function flowchart.

The function has been designed by taking the tasks highlighted above as a reference, yet a short summary is also given to be clear:

In the first step, necessary initializations are executed for the integers that will be used in function. The software only reads and records the sensor data unless the status of error flag is changed. When the error flag is triggered, software keeps recording 2-second more after the moment of error occurrence. A second flag is used to control the generation of the 4-second Error Message in MRAMs. When the Stop Data Record flag is also triggered, the sensor values are read from MRAMs and written to the corresponding EEPROM from the starting address to the last one by using a loop. However, the error recording cannot continue until this loop, which is used for communication between MRAM and EEPROM, is completed. In other words, since the microprocessor cannot run another function at the same time, it is not able to go on recording new data until the transfer of these 960.000 bytes is completed.

The code flow exits out of the loop when the entire error message is transmitted to permanent error recording structure (EEPROM). The corresponding counters and flags are reset and the MRAM real time data logging is reactivated.

3. TESTS FOR THE MICROPROCESSOR-BASED SYSTEM

In this chapter, the test environment for all of the software which were developpt in section 2.2 were created, and the performance, precision and accuracy of software were tested. To perform the tests, a test platform consisting of a test computer, Diagnostic PCB, an inverter control unit as the main unit, supply units, an emulator, probes and measurement devices such as oscilloscope, multimeter etc was set up. All tests for Microprocessor-based FDS were run on this test setup named Microprocessor-based ICU Setup. The views from different perspectives for the setup built for tests is given in Figures 3.1, 3.2 and 3.3.

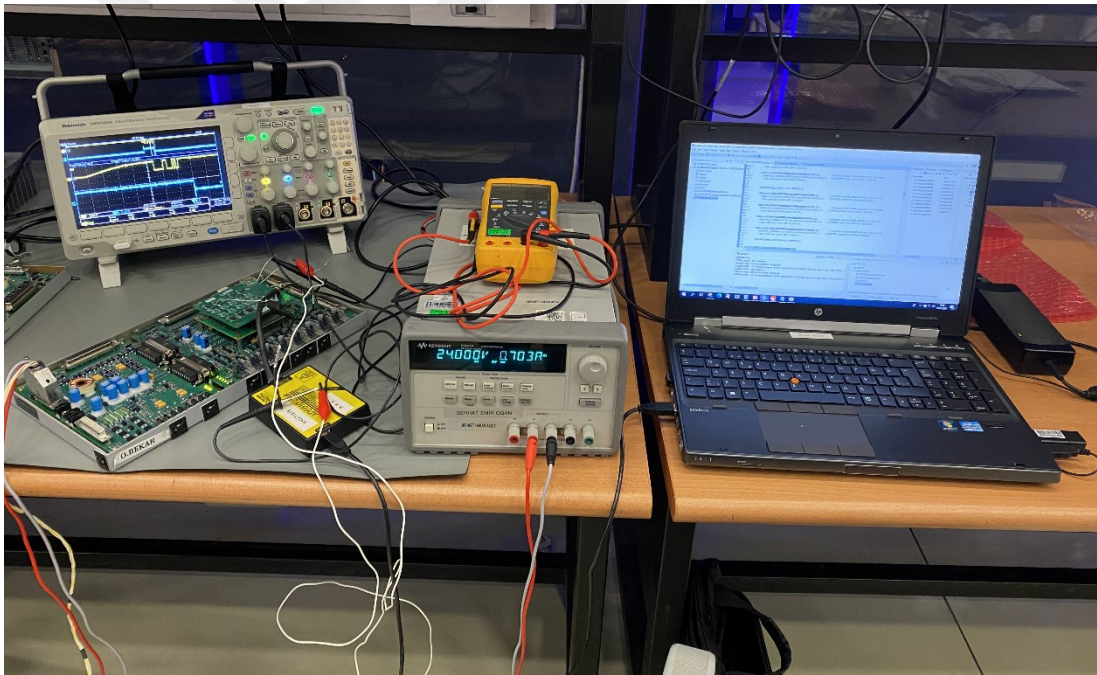


Figure 3.1 : The microprocessor-based ICU setup view-1.

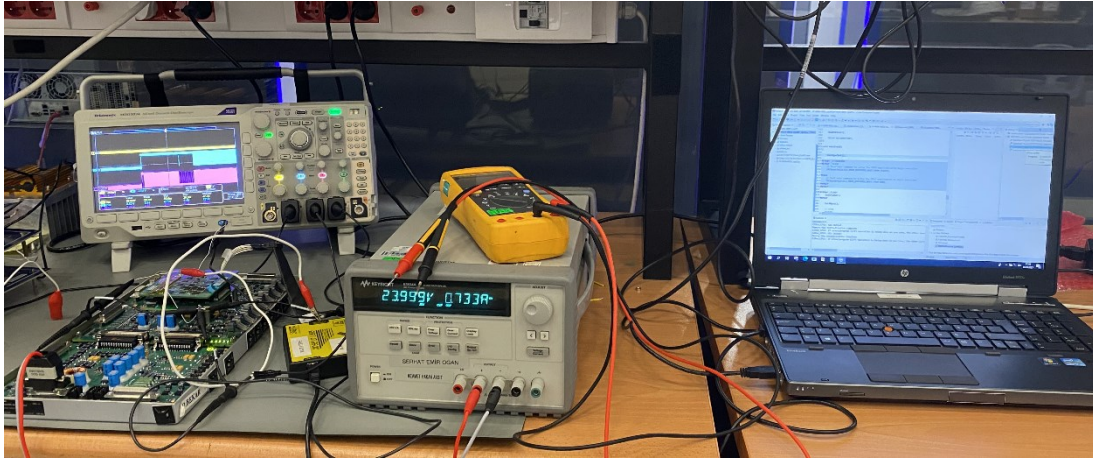


Figure 3.2 : The microprocessor-based ICU setup view-2.

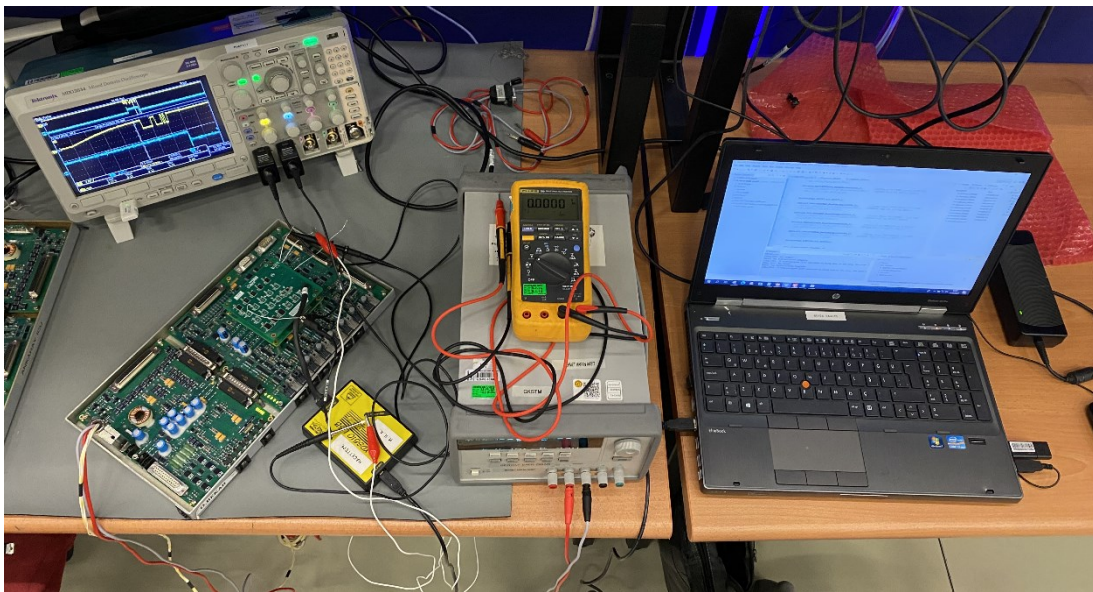


Figure 3.3 : The microprocessor-based ICU setup view-3.

Secondly, test scenarios were built and test software were developed one by one according to these scenarios. The test software basically creates a dummy data traffic to simulate the measurements received from the sensors, realtime recording & error status indications of the main unit, and the operating conditions. By doing so, the circumstances and possible cases may occur during the real operation were modeled, the accuracy and performance of software were tested in the microprocessor-based ICU setup.

For each test, start and end points of the software were determined based on the scenarios built individually. Two breakpoints were set on that points in the software, and a configured I / O was toggled at these points. Thus, signals were analyzed in the

elapsed time (run time) by debugging between the breakpoints. For analyzes of the results two different verification methods were used.

The first method is verification through the Code Composer Studio (CCS) integrated development environment for Texas Instruments (TI) embedded processors. In CCS interface, there is an Expression View window on the upper right and listed variables are able to be seen by using this window. Each time your code execution reaches the breakpoint, the watch will update. Updated values are highlighted with a yellow background. Basically, the watch window allows to examine and edit variables and C expressions. In the Watch window, complex expressions can be expanded and collapsed. Also terms and results are able to be evaluated and displayed in different formats. Within the scope of thesis, this feature was used to verify the functions accuracy. System was paused by using breakpoints and the success rate was evaluated by controlling the return values of read functions that were assigned to the expressions by software.

As the second validity check method, the oscilloscope screen was displayed between the breakpoints and signals were analyzed in that time interval. The corresponding buses were watched by using the probes, and the selected waveforms were saved through a USB flash drive by using the multipurpose knobs. Data on the buses were marked by using oscilloscope cursors, and examined in detail.

In conclusion, all of the results were both checked by using CCS interface and oscilloscope by debugging with breakpoints. Diagnostic software was tested by generating a dummy traffic between ICU and sensors in all possible cases such as unrecording mode, real time recording mode, error conditions etc. Hence, the features like system accuracy and performance were verified, software were validated.

3.1 MRAM Write & Read Functions Test Results

The test software for the MRAM reading and writing functions which creation steps are described in section 2.2.1 were created. The test software was run in the microprocessor-based ICU setup. Within the scope of Test-1, first ten addresses of MRAM1 and different random data were chosen and test data were written in that addresses. Right after writing function, reading function was also run in the following line. Two breakpoints were set before writing and after reading. By doing so, test

software was run partially and measurements were taken separately. While the data and corresponding addresses are given in Table 3.1, test procedure is demonstrated in Figure 3.4.

Table 3.1 : MRAM write & read functions test details.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	MRAM IC <i>Selected</i>
READ1	1	0000000000000001	1	1
READ2	5	0000000000000101	2	1
READ3	17	0000000000010001	3	1
READ4	24	0000000000011000	4	1
READ5	30	0000000000011110	5	1
READ6	61	0000000000111101	6	1
READ7	88	0000000001011000	7	1
READ8	90	0000000001011010	8	1
READ9	122	0000000001111010	9	1
READ10	255	0000000011111111	10	1

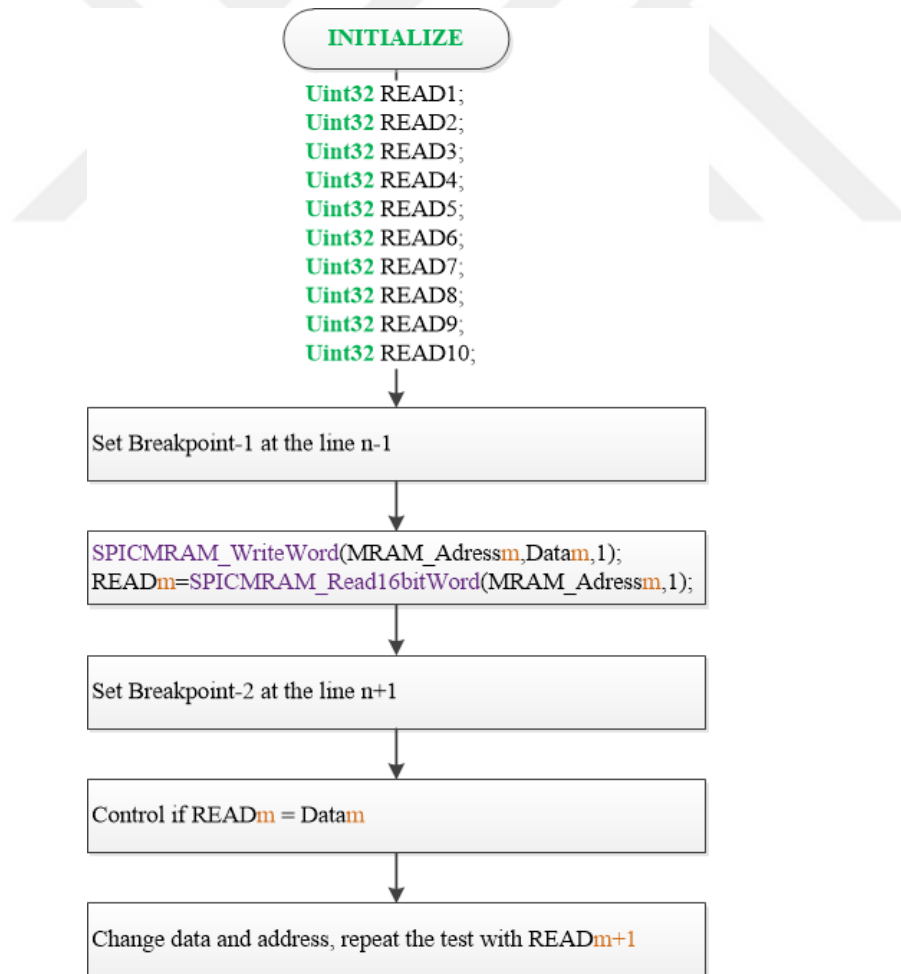


Figure 3.4 : MRAM read and write test procedure.

The results were both checked by using CCS interface and oscilloscope, and are given below:

In READ1, 1st address of the 1st MRAM is chosen for writing and 1 is written as data value. Same address is read in the second line and that value is assigned to the READ1 expressions. The results of both CCS and Oscilloscope are given step by step below.

In oscilloscope measurements, slave output (SO, MISO or SOMI) bus is displayed and time div as 400ns, voltage div is set as 1 volt. The period between the two breakpoints is given in Figure 3.5.

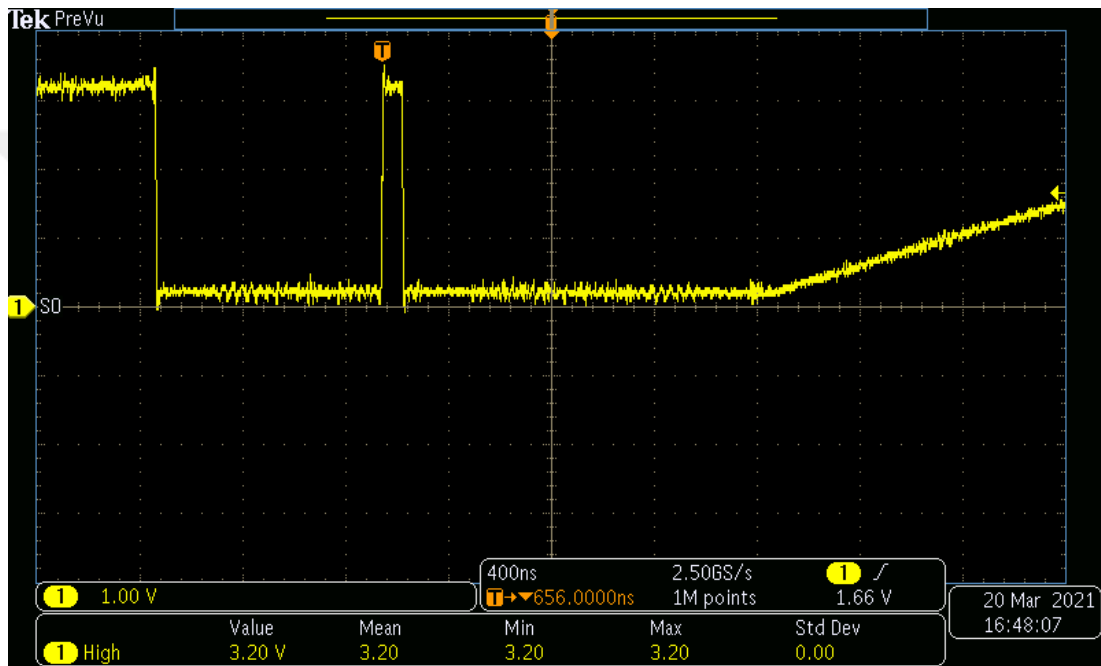


Figure 3.5 : READ1 oscilloscope view.

As seen in Figure 3.5 the SO bus or output goes tri-state. In digital electronics tri-state or 3-state logic allows an output or input pin/pad to assume a high impedance state, effectively removing the output from the circuit, in addition to the 0 and 1 logic levels. This situation is also emphasized in MRAM datasheet. After reading is accomplished, SO bus expectedly goes high impedance state. This point is the moment that chip select goes away. Indeed what is seen is the floating net being pulled-up somewhere. As there is no chip-select it does not matter where that line floats to. It could also go low or go random. The captured oscilloscope screenshot is interpreted by taking this point as the reference and given in Figure 3.6:

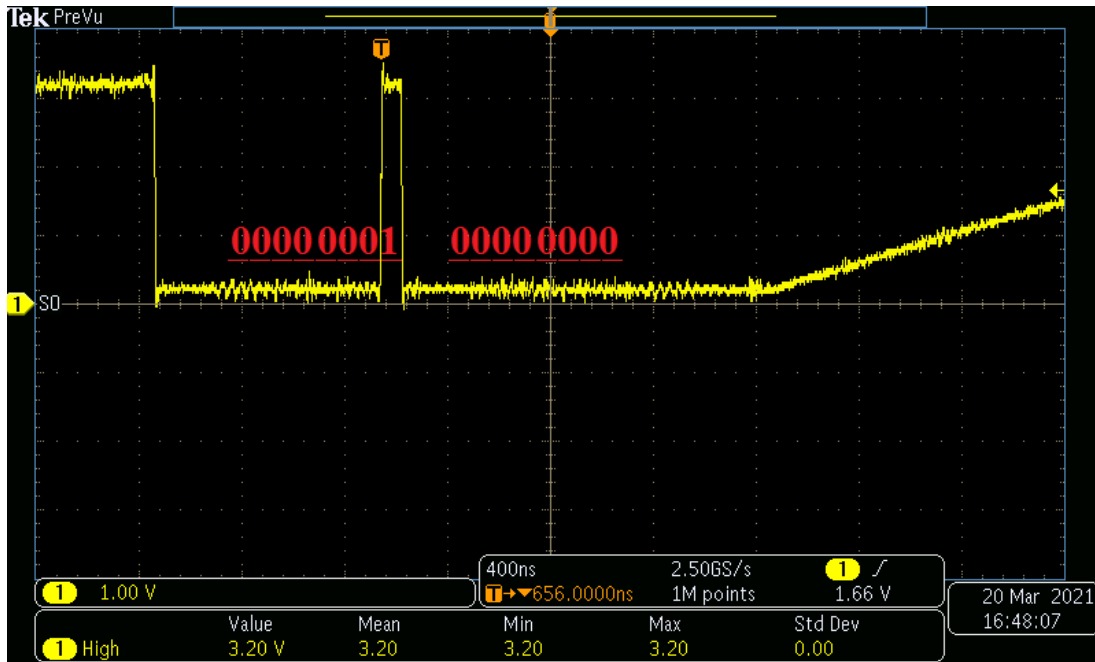


Figure 3.6 : READ1 oscilloscope view interpreted.

As explained in section 2.2.1, first byte is LSB and second one is MSB. So, the data read from MRAM1 Address1 is 0000000000000001 which is equal to 1 in decimal. It is observed that this value, which is equal to 1 in decimal base, is equal to the data we sent; hence, the writing process has been done successfully.

As mentioned in previous sections, the read result was assigned to the READ1 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.7.

Registers			
Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	0	0x0000AAD8@Data
READ3	unsigned long	0	0x0000AADA@Data
READ4	unsigned long	0	0x0000AAEE@Data
READ5	unsigned long	0	0x0000AAEA@Data
READ6	unsigned long	0	0x0000AAEC@Data
READ7	unsigned long	0	0x0000AAE8@Data
READ8	unsigned long	0	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data
Add new expression			

Figure 3.7 : READ1 CCS expression view.

So, using both the oscilloscope and the CCS interface, the READ1 result was double checked and it has been clearly seen that the first data was written and read properly.

The other nine tests were performed in the same way as explained for READ1 in details above. Explanations for the others have not been given in order to avoid repetition, only the results are given below respectively:

READ2:

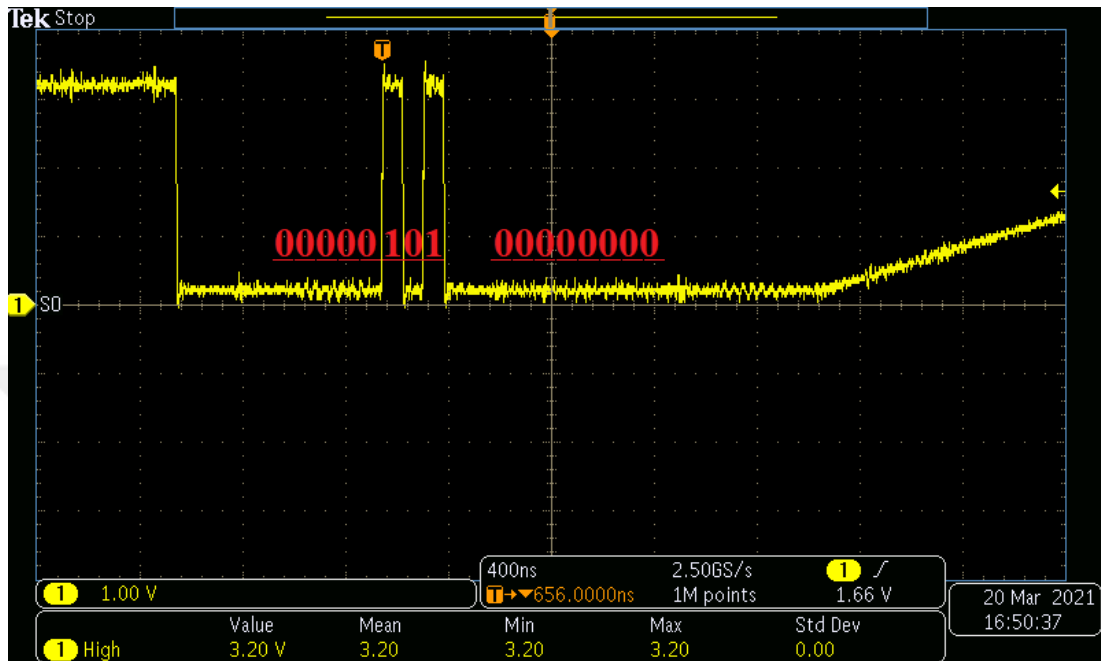


Figure 3.8 : READ2 oscilloscope view interpreted.

When it is considered that first byte is LSB and second one is MSB in Figure 3.8, it is seen that the data read from MRAM1 Address2 is 0000000000000101 which is equal to 5 in decimal. The read result was assigned to the READ2 expression and viewed on CCS View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.9.

Registers			
Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	0	0x0000AADA@Data
READ4	unsigned long	0	0x0000AAEE@Data
READ5	unsigned long	0	0x0000AAEA@Data
READ6	unsigned long	0	0x0000AAEC@Data
READ7	unsigned long	0	0x0000AAE8@Data
READ8	unsigned long	0	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data
Add new expression			

Figure 3.9 : READ2 CCS expression view.

It is observed that the READ2 value is also 5 which is equal to the data sent.

READ3:

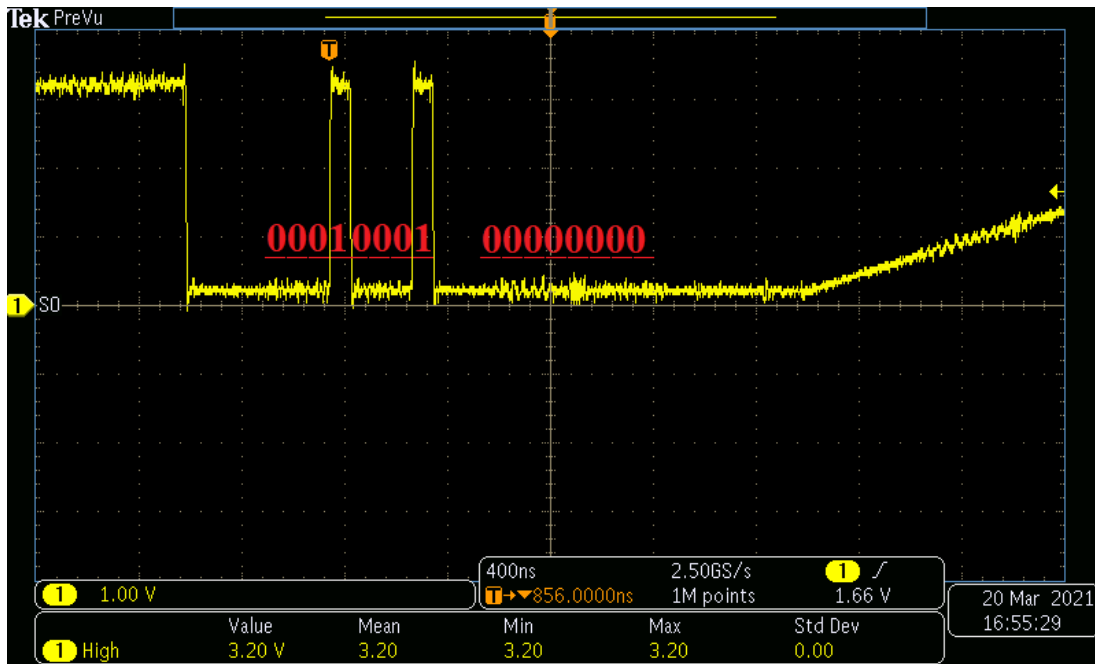


Figure 3.10 : READ3 oscilloscope view interpreted.

It is shown in Figure 3.10 that the data read from 3rd address of MRAM1 is 0000000000010001 which is equal to 17 in decimal.

The read result was assigned to the READ3 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.11.

(x)= Variables Expressions Registers			
Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	0	0x0000AAEE@Data
READ5	unsigned long	0	0x0000AAEA@Data
READ6	unsigned long	0	0x0000AAEC@Data
READ7	unsigned long	0	0x0000AAE8@Data
READ8	unsigned long	0	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data
+ Add new expression			

Figure 3.11 : READ3 CCS expression view.

It is observed that the READ3 value is also 17 which is equal to the data sent.

READ4:

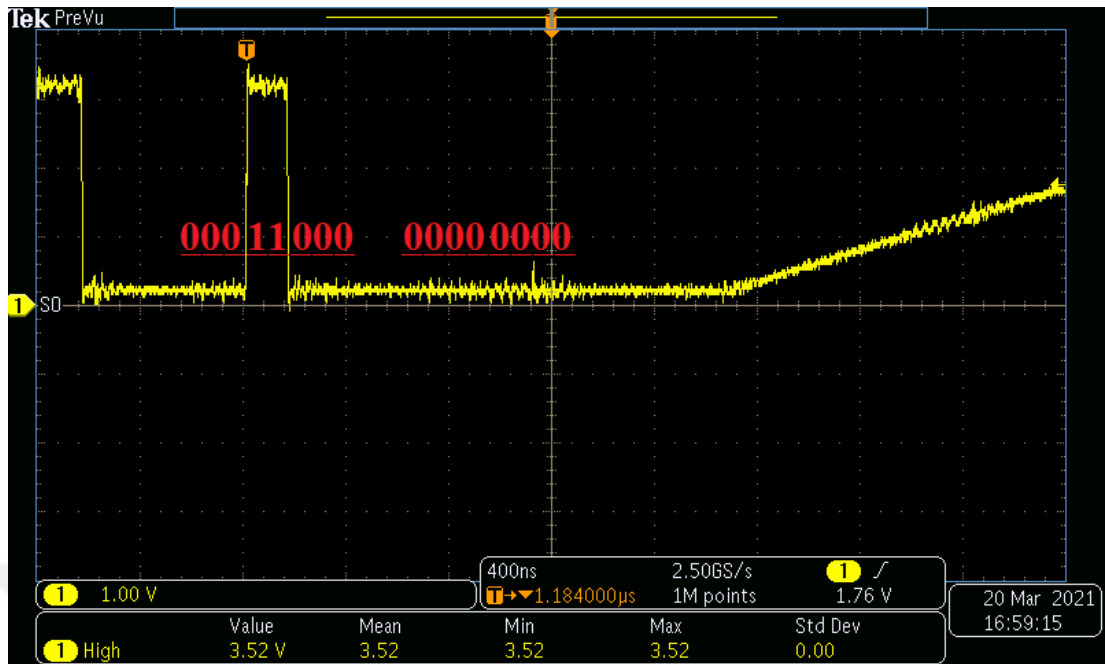


Figure 3.12 : READ4 oscilloscope view interpreted.

It is shown in Figure 3.12 that the data read from 5th address of MRAM1 is 0000000000011000 which is equal to 24 in decimal.

The read result was assigned to the READ4 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.13.

Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	24	0x0000AAEE@Data
READ5	unsigned long	0	0x0000AAEA@Data
READ6	unsigned long	0	0x0000AAEC@Data
READ7	unsigned long	0	0x0000AAE8@Data
READ8	unsigned long	0	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data

Figure 3.13 : READ4 CCS expression view.

It is observed that the READ4 value is also 24 which is equal to the data sent.

READ5:

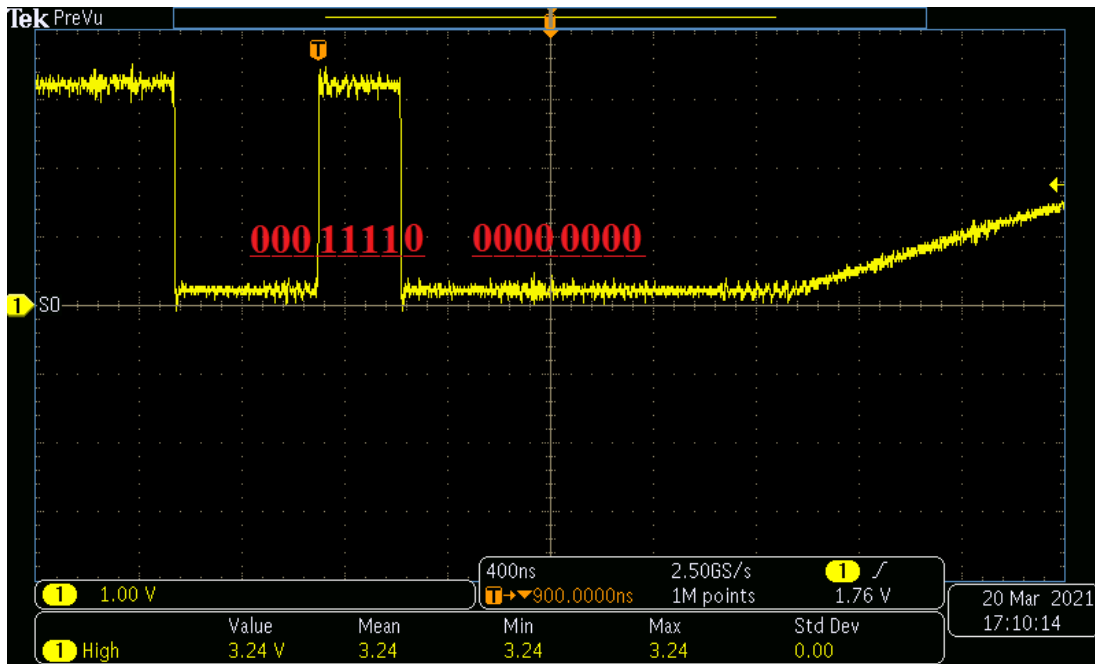


Figure 3.14 : READ5 oscilloscope view interpreted.

It is shown in Figure 3.14 that the data read from 5th address of MRAM1 is 0000000000011110 which is equal to 30 in decimal.

The read result was assigned to the READ5 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.15.

Registers			
Expression	Type	Value	Address
(*)= READ1	unsigned long	1	0x0000AADC@Data
(*)= READ2	unsigned long	5	0x0000AAD8@Data
(*)= READ3	unsigned long	17	0x0000AADA@Data
(*)= READ4	unsigned long	24	0x0000AAEE@Data
(*)= READ5	unsigned long	30	0x0000AAEA@Data
(*)= READ6	unsigned long	0	0x0000AAEC@Data
(*)= READ7	unsigned long	0	0x0000AAE8@Data
(*)= READ8	unsigned long	0	0x0000AAE4@Data
(*)= READ9	unsigned long	0	0x0000AAD4@Data
(*)= READ10	unsigned long	0	0x0000AAD6@Data
+ Add new expression			

Figure 3.15 : READ5 CCS expression view.

It is observed that the READ5 value is also 30 which is equal to the data sent.

READ6:

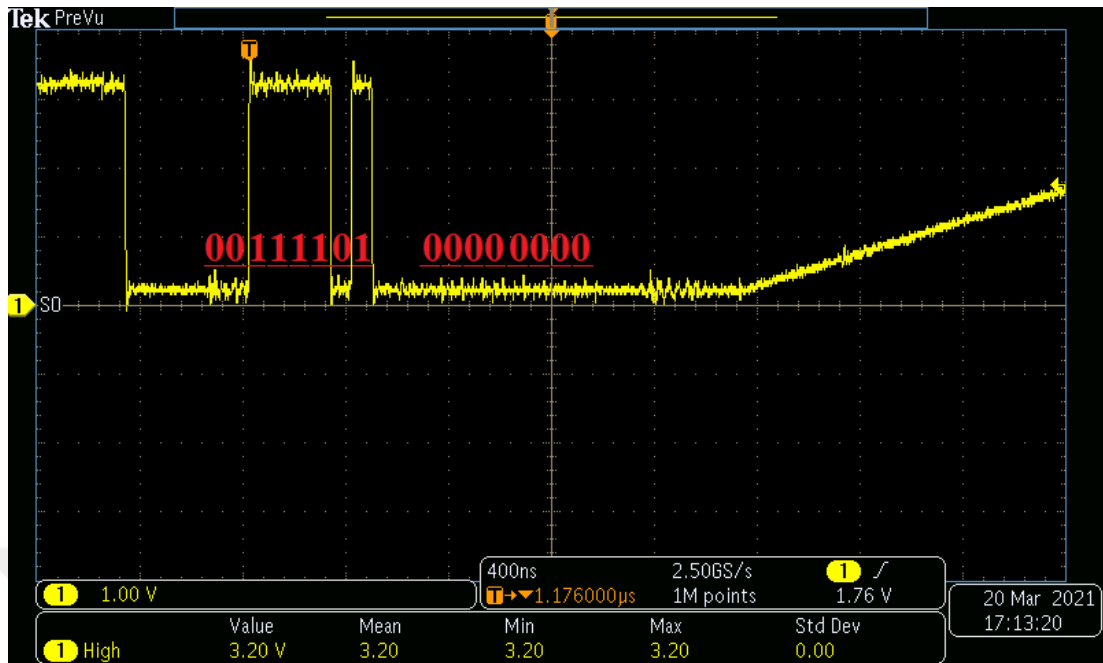


Figure 3.16 : READ6 oscilloscope view interpreted.

It is shown in Figure 3.16 that the data read from 6th address of MRAM1 is 0000000000111101 which is equal to 61 in decimal.

The read result was assigned to the READ6 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.17.

Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	24	0x0000AAEE@Data
READ5	unsigned long	30	0x0000AAEA@Data
READ6	unsigned long	61	0x0000AAEC@Data
READ7	unsigned long	0	0x0000AAE8@Data
READ8	unsigned long	0	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data

Figure 3.17 : READ6 CCS expression view.

It is observed that the READ6 value is also 61 which is equal to the data we sent.

READ7:

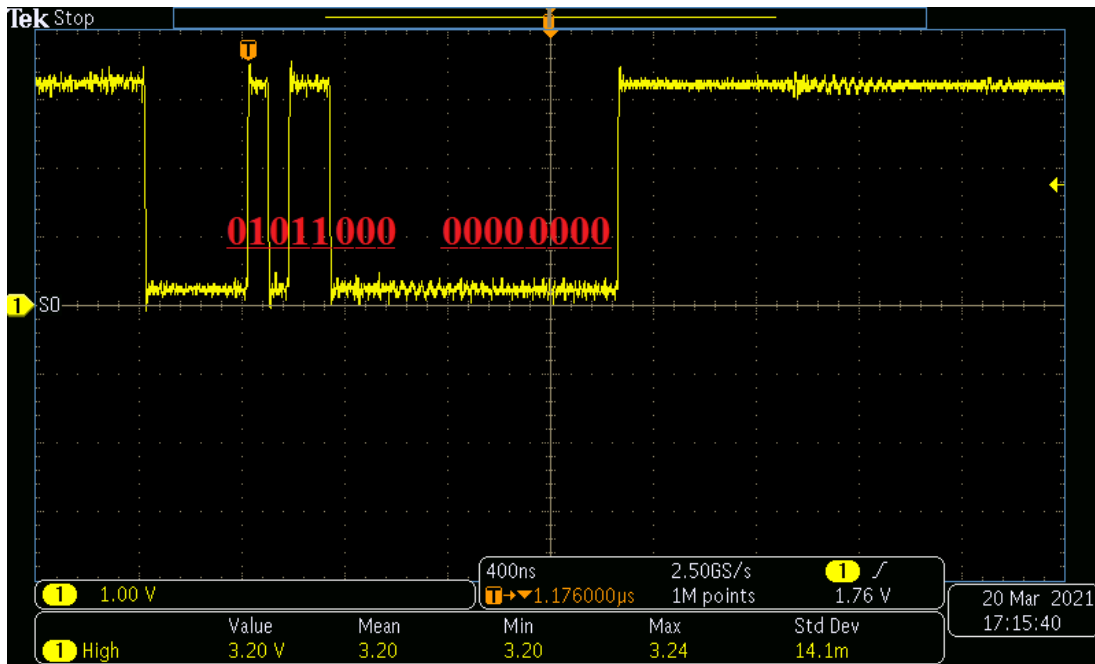


Figure 3.18 : READ7 oscilloscope view interpreted.

It is shown in Figure 3.18 that the data read from 7th address of MRAM1 is 0000000001011000 which is equal to 88 in decimal.

The read result was assigned to the READ7 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.19.

(x)= Variables Expressions Registers			
Expression	Type	Value	Address
(x)= READ1	unsigned long	1	0x0000AADC@Data
(x)= READ2	unsigned long	5	0x0000AAD8@Data
(x)= READ3	unsigned long	17	0x0000AADA@Data
(x)= READ4	unsigned long	24	0x0000AAEE@Data
(x)= READ5	unsigned long	30	0x0000AAEA@Data
(x)= READ6	unsigned long	61	0x0000AAEC@Data
(x)= READ7	unsigned long	88	0x0000AAE8@Data
(x)= READ8	unsigned long	0	0x0000AAE4@Data
(x)= READ9	unsigned long	0	0x0000AAD4@Data
(x)= READ10	unsigned long	0	0x0000AAD6@Data
+ Add new expression			

Figure 3.19 : READ7 CCS expression view.

It is observed that the READ7 value is also 88 which is equal to the data we sent.

READ8:

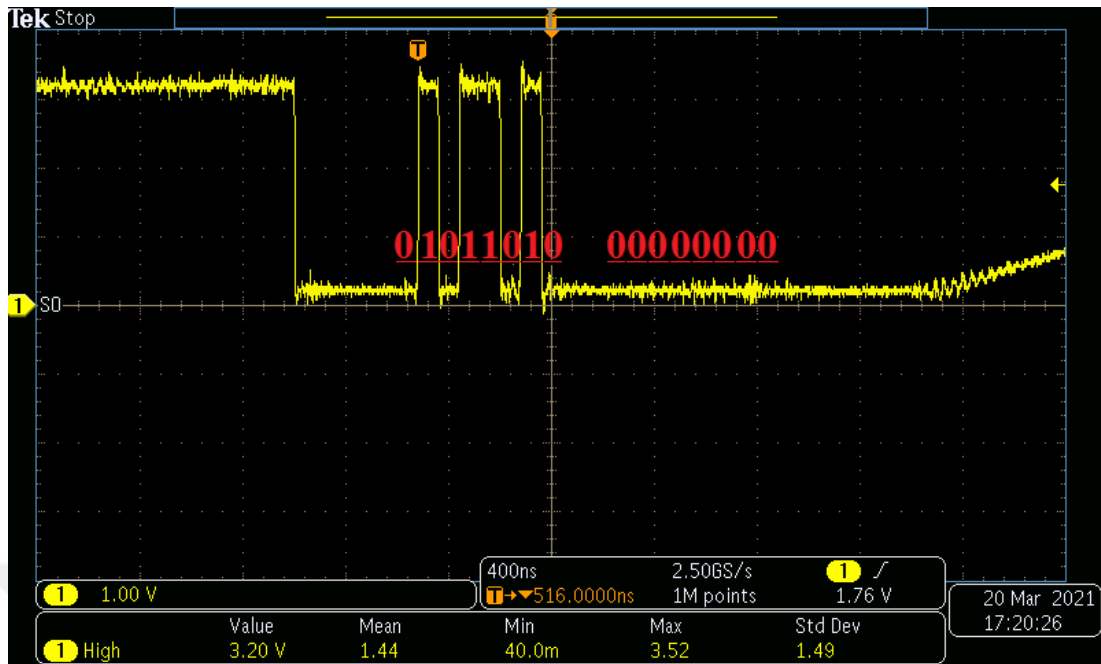


Figure 3.20 : READ8 oscilloscope view interpreted.

It is shown in Figure 3.20 that the data read from 8th address of MRAM1 is 0000000001011010 which is equal to 90 in decimal.

The read result was assigned to the READ8 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.21.

Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	24	0x0000AAEE@Data
READ5	unsigned long	30	0x0000AAEA@Data
READ6	unsigned long	61	0x0000AAEC@Data
READ7	unsigned long	88	0x0000AAE8@Data
READ8	unsigned long	90	0x0000AAE4@Data
READ9	unsigned long	0	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data

Figure 3.21 : READ8 CCS expression view.

It is observed that the READ8 value is also 90 which is equal to the data we sent.

READ9:

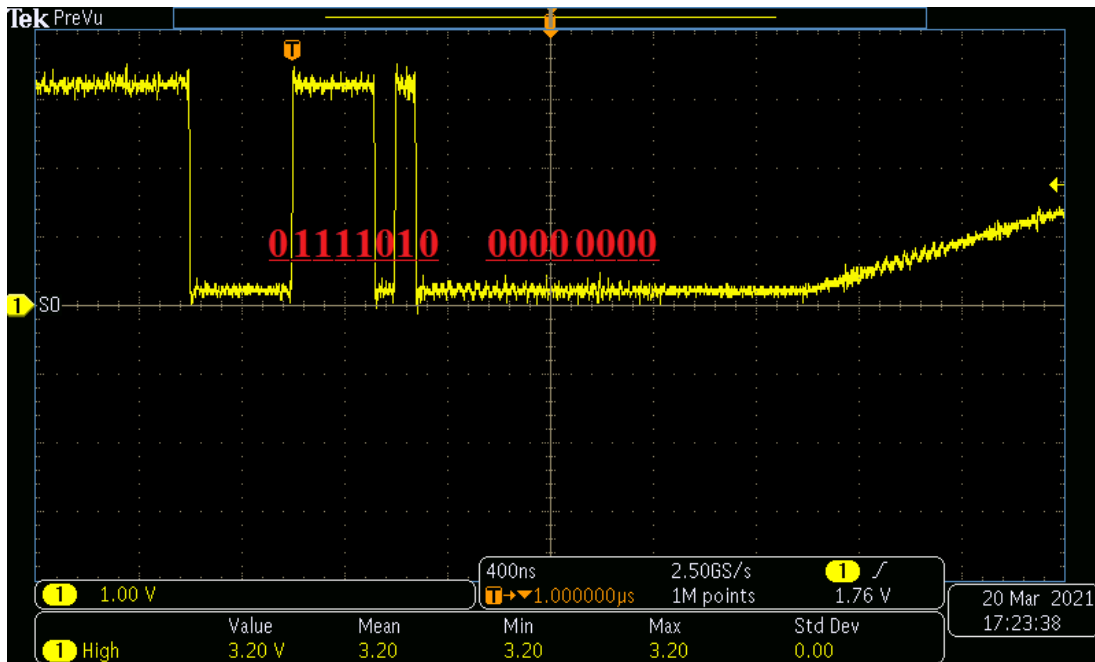


Figure 3.22 : READ9 oscilloscope view interpreted.

It is shown in Figure 3.22 that the data read from 9th address of MRAM1 is 0000000001111010 which is equal to 122 in decimal.

The read result was assigned to the READ9 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.23.

Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	24	0x0000AAEE@Data
READ5	unsigned long	30	0x0000AAEA@Data
READ6	unsigned long	61	0x0000AAEC@Data
READ7	unsigned long	88	0x0000AAE8@Data
READ8	unsigned long	90	0x0000AAE4@Data
READ9	unsigned long	122	0x0000AAD4@Data
READ10	unsigned long	0	0x0000AAD6@Data

Figure 3.23 : READ9 CCS expression view.

It is observed that the READ9 value is also 122 which is equal to the data we sent.

READ10:

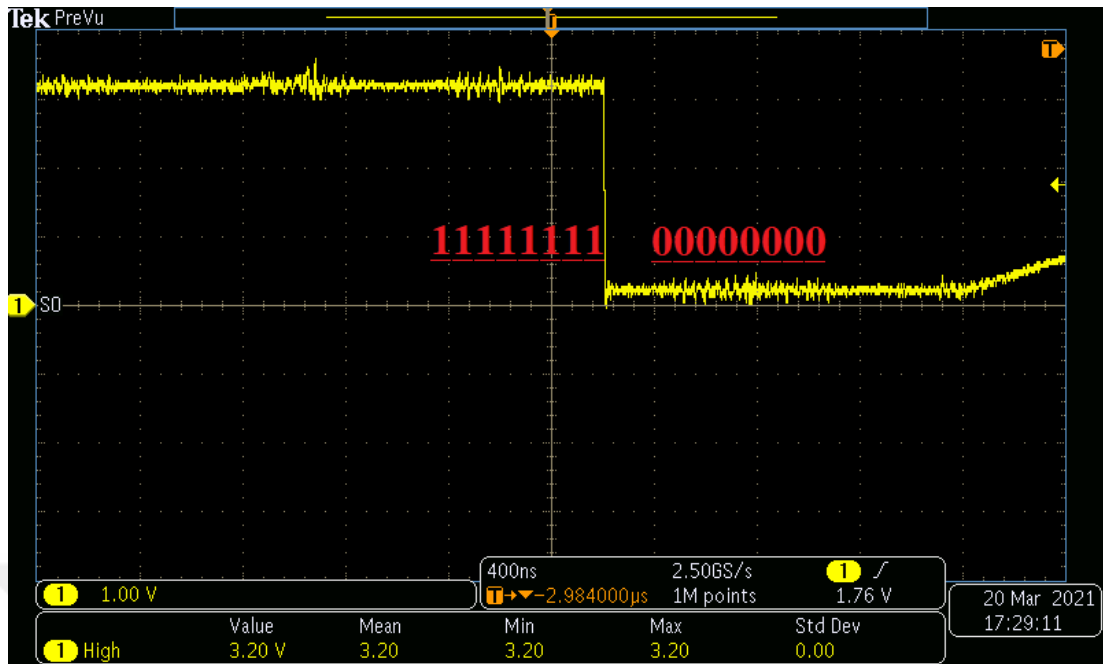


Figure 3.24 : READ10 oscilloscope view interpreted.

It is shown in Figure 3.24 that the data read from 10th address of MRAM1 is 0000000001111111 which is equal to 255 in decimal.

The read result was assigned to the READ10 expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.25.

Registers			
Expression	Type	Value	Address
READ1	unsigned long	1	0x0000AADC@Data
READ2	unsigned long	5	0x0000AAD8@Data
READ3	unsigned long	17	0x0000AADA@Data
READ4	unsigned long	24	0x0000AAEE@Data
READ5	unsigned long	30	0x0000AAEA@Data
READ6	unsigned long	61	0x0000AAEC@Data
READ7	unsigned long	88	0x0000AAE8@Data
READ8	unsigned long	90	0x0000AAE4@Data
READ9	unsigned long	122	0x0000AAD4@Data
READ10	unsigned long	255	0x0000AAD6@Data
Add new expression			

Figure 3.25 : READ10 CCS expression view.

It is observed that the READ10 value is also 255 which is equal to the data we sent.

In the context of this section, MRAM Write and Read Functions were investigated by using first ten addresses of MRAM1. Ten different data value were chosen randomly

and written at that addresses. The success rate of the functions were evaluated by watching the SPI bus with digital oscilloscope and controlling the return values of MRAM Read Function that were assigned by software. So, the results were double checked and it has been clearly seen that the results are identical with the values given in Table 3.1. As the result of the tests, it has been observed that the writing and reading processes have been performed properly. It has been concluded that MRAM Write and Read Functions were created successfully, and have worked properly.

3.2 MRAM Continuous Write Functions Test Results

In this section, MRAM Continuous Write Functions which creation steps are described in section 2.2.2 are tested and results are given. Within the scope of the test, MRAM1 Continuous Write and MRAM2 Continuous Write functions were run separately and their performance was monitored by using both CCS Expression View Window and digital oscilloscope. Two breakpoint were set before and right after the functions. By doing so, functions were run partially and measurements were taken separately. A total of 10 addresses were randomly selected, five from the first MRAM and five from the second MRAM, and their success (accuracy) rate were checked according to the reading results. Furthermore, the run time or execution time of the functions were recorded by using breakpoints and oscilloscope device, and the results are given below:

Firstly, the success (accuracy) of the functions were checked by reading randomly chosen 10 addresses. The addresses and expected data are given in Table 3.2.

Table 3.2 : MRAM continuous write function test details.

Test Step <i>Expressions</i>	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	MRAM IC <i>Selected</i>
READ1	0	0000000000000000	0	1
READ2	2	0000000000000010	2	1
READ3	100	000000001100100	100	1
READ4	146	0000000010010010	146	1
READ5	250	0000000011111010	250	1
READ6	500	0000000111110100	500	2
READ7	1000	0000001111101000	1000	2
READ8	0	0000000000000000	65534	2
READ9	2	0000000000000010	65536	2
READ10	21262	0101001100001110	480000	2

As the operating logic of the MRAM continuous write algorithm is explained in section 2.2.2, address values are written to the addresses which are less than or equal to 65534. For instance, if 40000th address of an MRAM is read, the value that is read from that address should be 40000. However, because READ and WRITE processes are executed as words and a word could be maximally 1111111111111111 which is equal to 65535 in decimal, the value was reset at 65534 for sake of the ease of trace. So, it could be considered that the data which is written to the addresses are determined by performing the modulo operation (mod65534). To illustrate, if the address that writing function is run at is 76546, the data that is written will be 11012. So, the read value will be also 11012 to deduce that function works properly.

Same verification methods used in Test-2 were used again. Breakpoints were set and signals were analyzed in that time interval by debugging between the breakpoints. Thus, the results were checked by using both CCS interface and digital oscilloscope. The results with their explications are given below:

READ1:

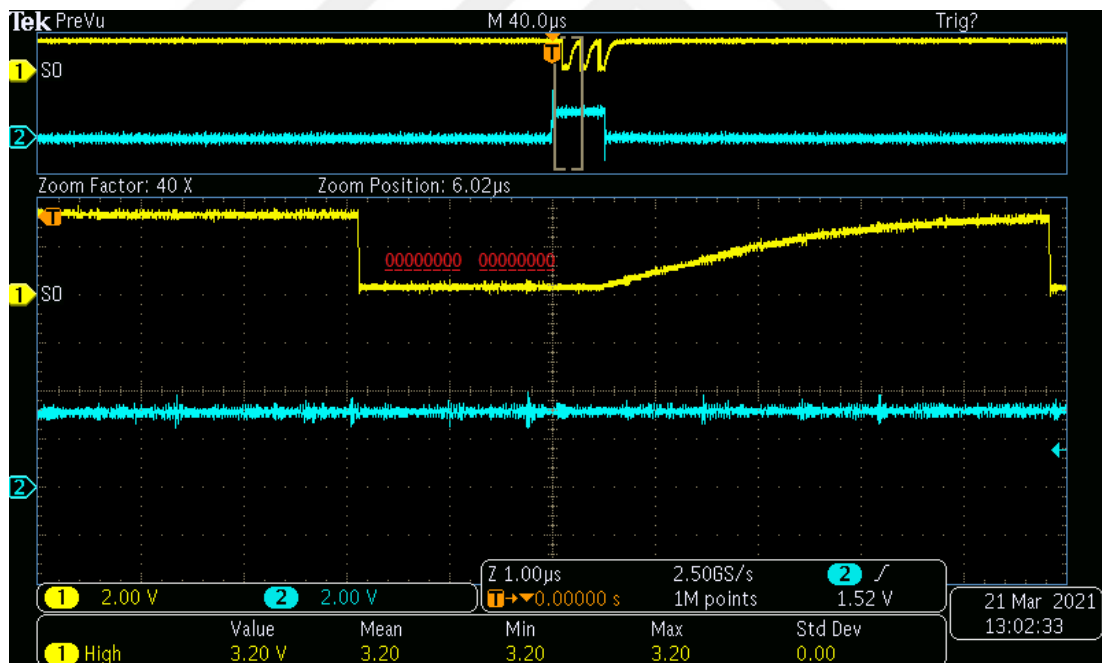


Figure 3.26 : READ1 oscilloscope view interpreted.

It is shown in Figure 3.26 that the data read from address 0 of MRAM1 is 0000000000000000 which is equal to 0 in decimal.

The read result was assigned to the Address1_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.27.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	0	0x0000AACF@Data
Address3_MRAM	unsigned int	0	0x0000AAC3@Data
Address4_MRAM	unsigned int	0	0x0000AAC4@Data
Address5_MRAM	unsigned int	0	0x0000AAC9@Data
Address6_MRAM	unsigned int	0	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.27 : READ1 CCS expression view.

It is observed that the Address1_MRAM value is also 0 which is equal to the data we sent.

READ2:

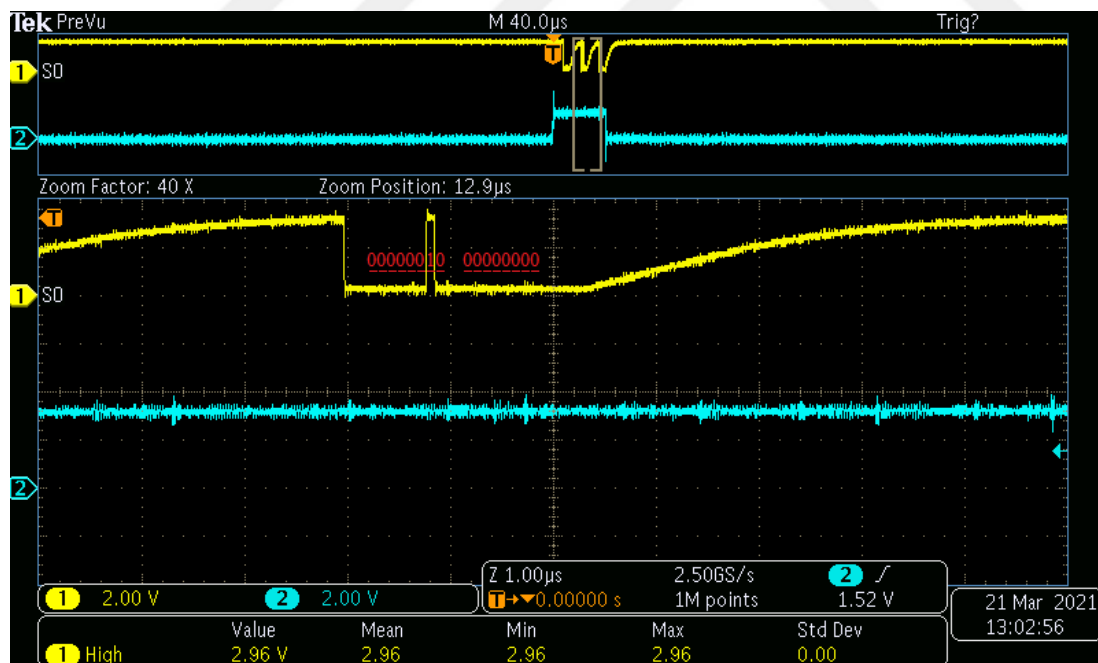


Figure 3.28 : READ2 oscilloscope view interpreted.

It is shown in Figure 3.28 that the data read from 2nd address of MRAM1 is 0000000000000010 which is equal to 2 in decimal.

The read result was assigned to the Address2_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.29

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	0	0x0000AAC3@Data
Address4_MRAM	unsigned int	0	0x0000AAC4@Data
Address5_MRAM	unsigned int	0	0x0000AAC9@Data
Address6_MRAM	unsigned int	0	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data

Figure 3.29 : READ2 CCS expression view.

It is observed that the Address2_MRAM value is also 2 which is equal to the data we sent.

READ3:

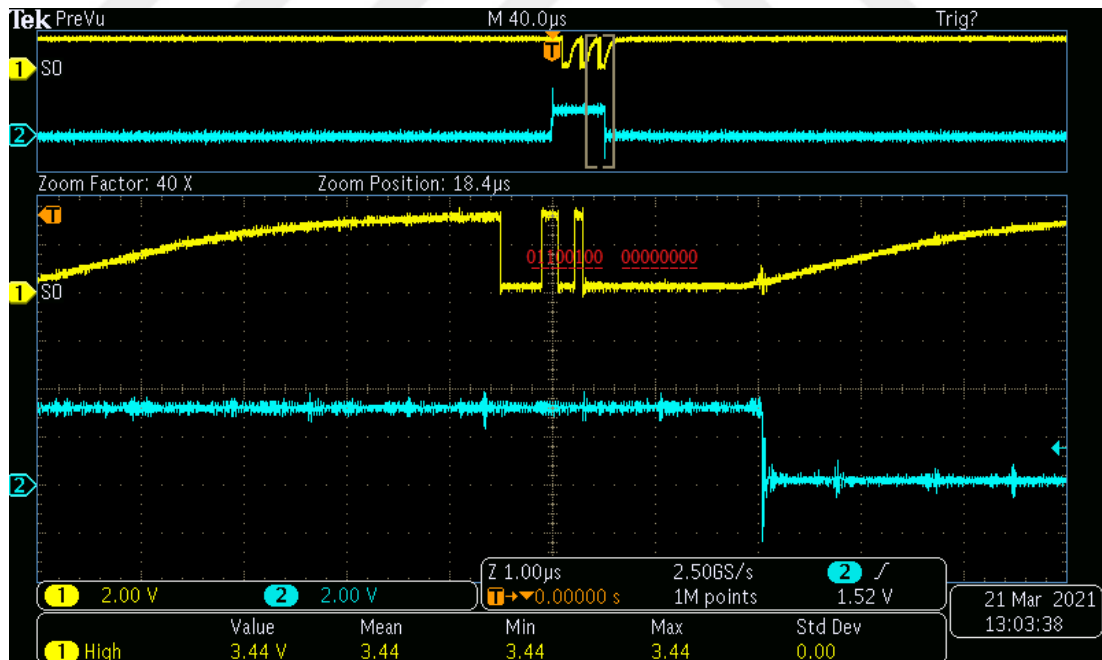


Figure 3.30 : READ3 oscilloscope view interpreted.

It is shown in Figure 3.30 that the data read from 100th address of MRAM1 is 0000000001100100 which is equal to 100 in decimal.

The read result was assigned to the Address3_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.31.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	0	0x0000AAC4@Data
Address5_MRAM	unsigned int	0	0x0000AAC9@Data
Address6_MRAM	unsigned int	0	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.31 : READ3 CCS expression view.

It is observed that the Address3_MRAM value is also 100 which is equal to the data we sent.

READ4:

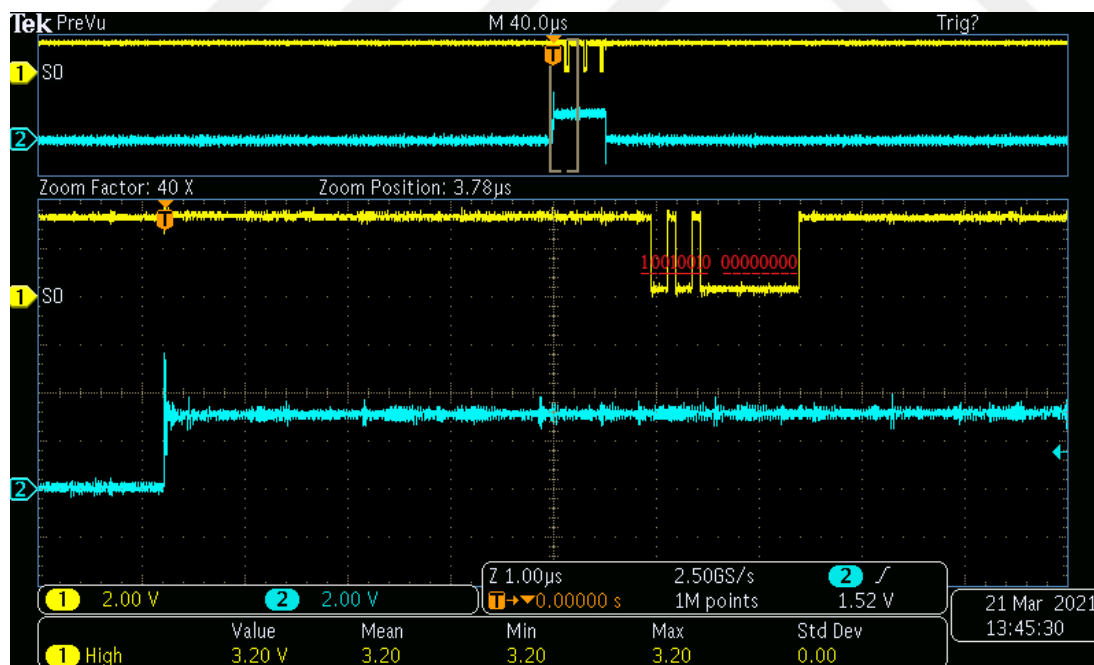


Figure 3.32 : READ4 oscilloscope view interpreted.

It is shown in Figure 3.32 that the data read from 146th address of MRAM1 is 0000000010010010 which is equal to 146 in decimal.

The read result was assigned to the Address4_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.33.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	0	0x0000AAC9@Data
Address6_MRAM	unsigned int	0	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.33 : READ4 CCS expression view.

It is observed that the Address4_MRAM value is also 146 which is equal to the data we sent.

READ5:

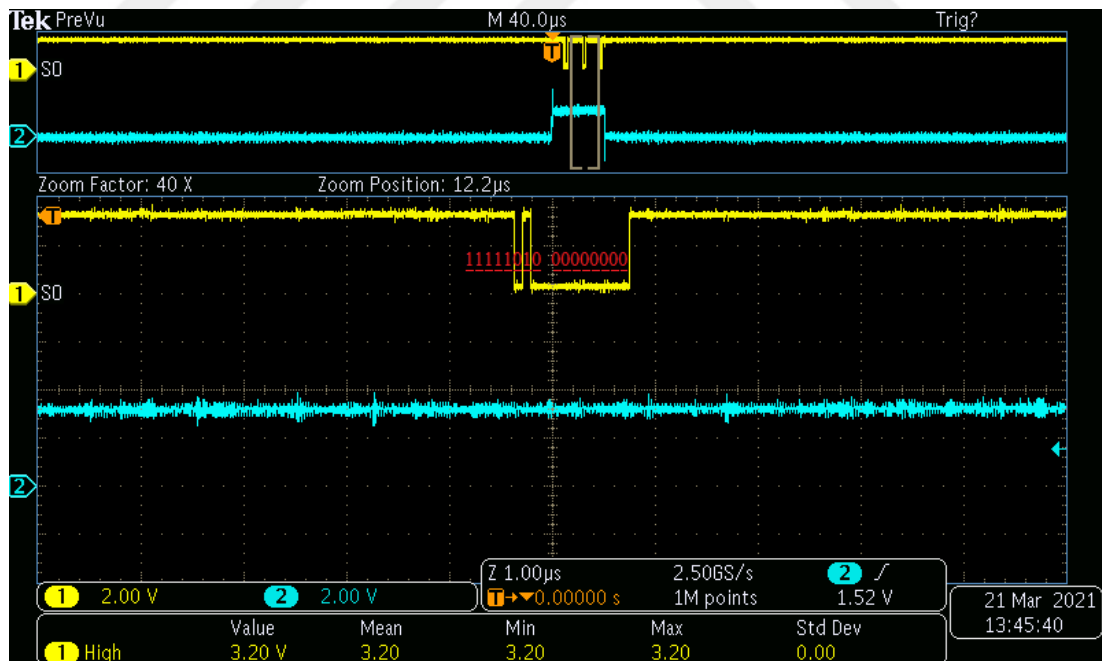


Figure 3.34 : READ5 oscilloscope view interpreted.

It is shown in Figure 3.34 that the data read from 250th address of MRAM1 is 0000000011111010 which is equal to 250 in decimal.

The read result was assigned to the Address5_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.35.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	0	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data

Figure 3.35 : READ5 CCS expression view.

It is observed that the Address5_MRAM value is also 250 which is equal to the data we sent.

As it is explained at the entrance of the section, the error message will be recorded in two different MRAMs as before and after the error. In this context, accuracy checks (read and write comparison) were also made for 2nd MRAM, five random randomly chosen of the results are given below:

READ6:

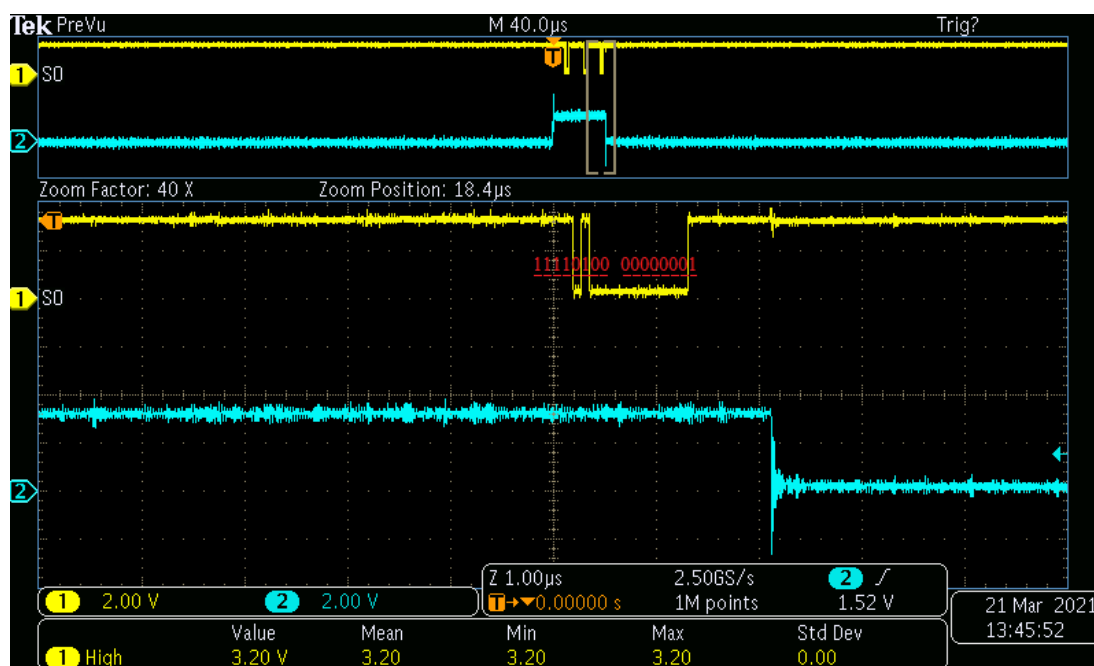


Figure 3.36 : READ6 oscilloscope view interpreted.

It is shown in Figure 3.36 that the data read from 500th address of MRAM2 is 0000000111110100 which is equal to 500 in decimal.

The read result was assigned to the Address6_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.37

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	500	0x0000AAC1@Data
Address7_MRAM	unsigned int	0	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.37 : READ6 CCS expression view.

It is observed that the Address6_MRAM value is also 500 which is equal to the data we sent.

READ7:

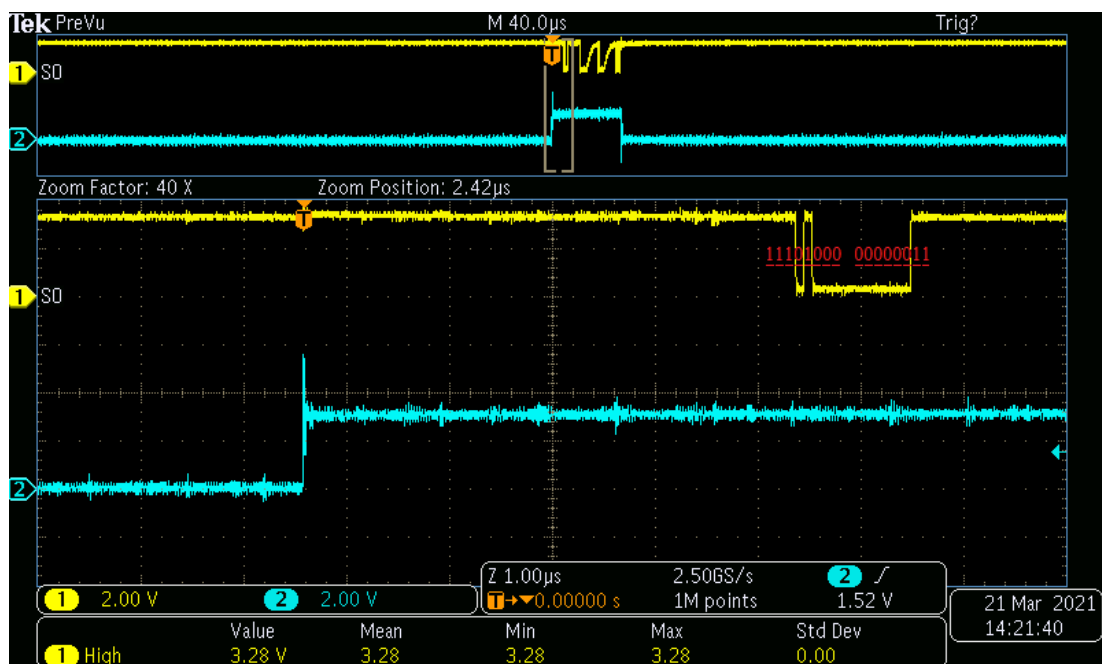


Figure 3.38 : READ7 oscilloscope view interpreted.

It is shown in Figure 3.38 that the data read from 1000th address of MRAM2 is 0000001111101000 which is equal to 1000 in decimal.

The read result was assigned to the Address7_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.39.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	500	0x0000AAC1@Data
Address7_MRAM	unsigned int	1000	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.39 : READ7 CCS expression view.

It is observed that the Address7_MRAM value is also 1000 which is equal to the data we sent.

READ8:

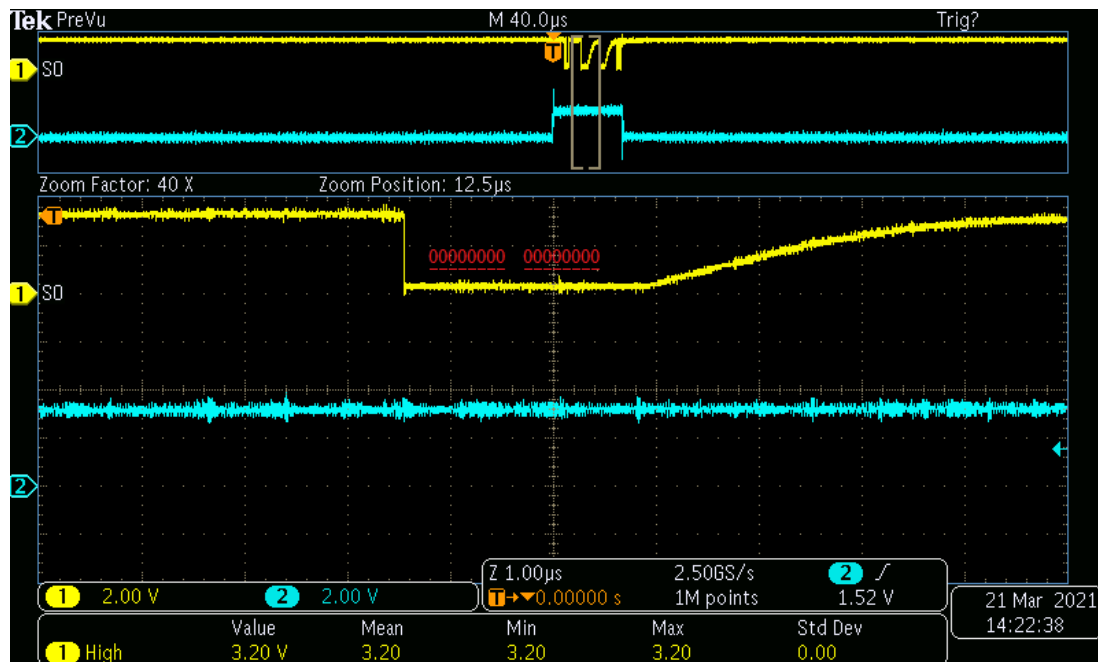


Figure 3.40 : READ8 oscilloscope view interpreted.

It is shown in Figure 3.40 that the data read from address 0 of MRAM2 is 0000000000000000 which is equal to 0 in decimal.

The read result was assigned to the Address8_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.41.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	500	0x0000AAC1@Data
Address7_MRAM	unsigned int	1000	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	0	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.41 : READ8 CCS expression view.

It is observed that the Address8_MRAM value is also 0 which is equal to the data we sent.

READ9:

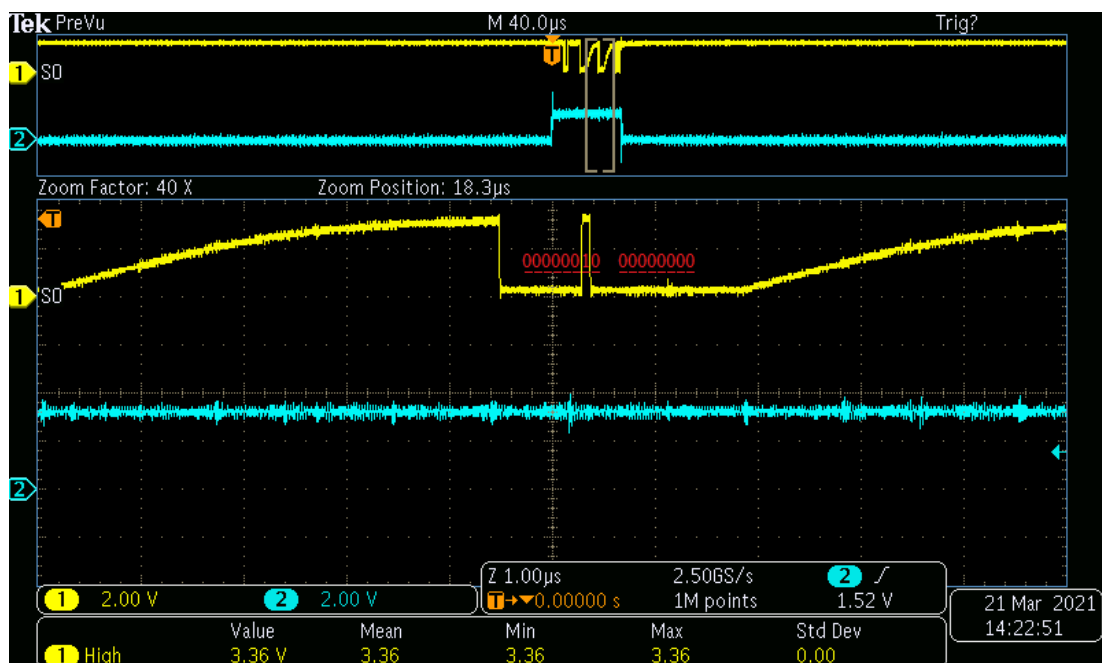


Figure 3.42 : READ9 oscilloscope view interpreted.

It is shown in Figure 3.42 that the data read from 2nd address of MRAM2 is 0000000000000010 which is equal to 2 in decimal.

The read result was assigned to the Address9_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.43.

Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	500	0x0000AAC1@Data
Address7_MRAM	unsigned int	1000	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	2	0x0000AAC2@Data
Address10_MRAM	unsigned int	0	0x0000AAC8@Data
+ Add new expression			

Figure 3.43 : READ9 CCS expression view.

It is observed that the Address9_MRAM value is also 2 which is equal to the data sent.

READ10:

In the Final reading test of Test-3, a reading sample including an address value greater than 65534 was chosen on purpose. The address value selected for this test is 480000 and the value to be read is 21262, which is 480000 modulus 65534. The test results are as follows:

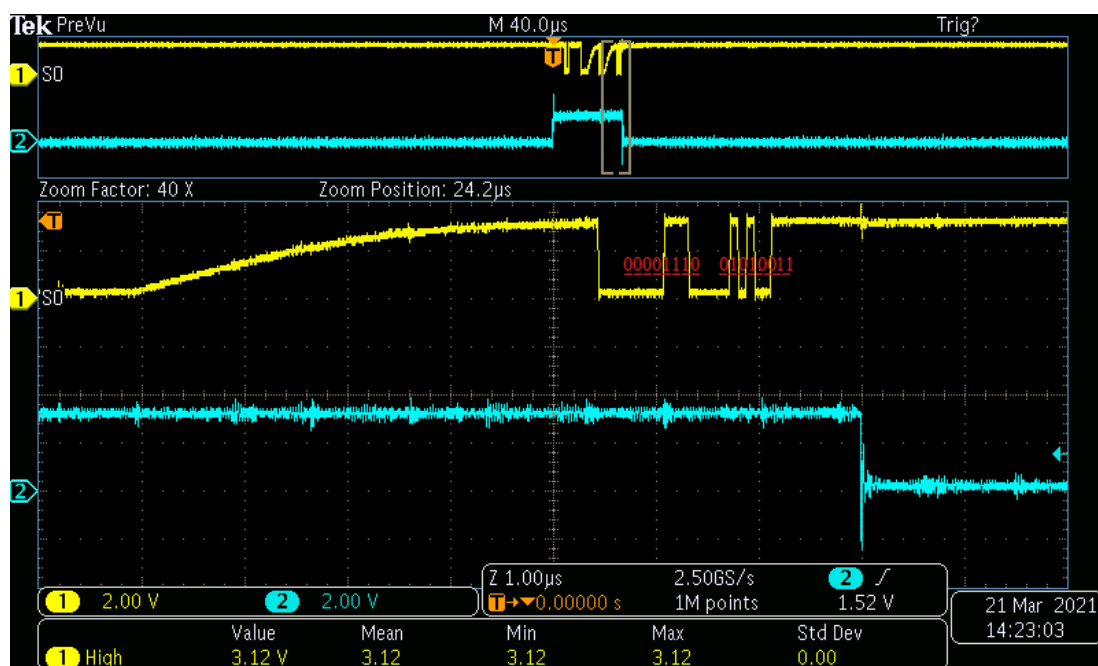
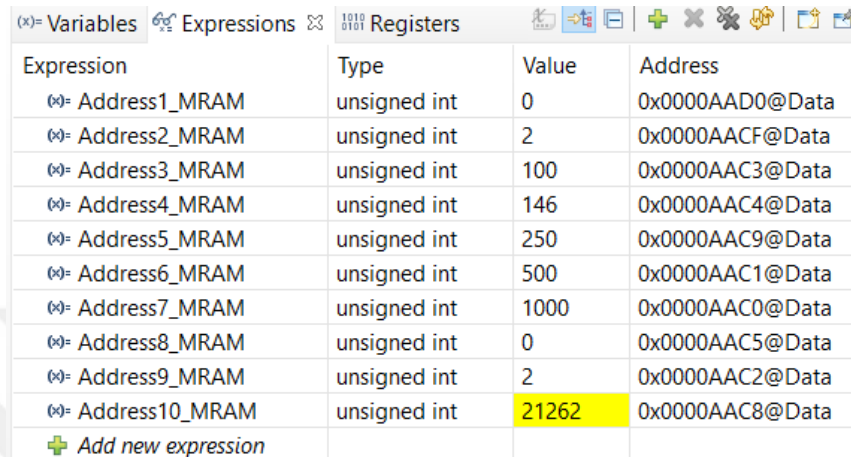


Figure 3.44 : READ10 oscilloscope view interpreted.

It is shown in Figure 3.44 that the data read from 480000th address of MRAM2 is 0101001100001110 which is equal to 21262 in decimal.

The read result was assigned to the Address10_MRAM expression and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.45.



Expression	Type	Value	Address
Address1_MRAM	unsigned int	0	0x0000AAD0@Data
Address2_MRAM	unsigned int	2	0x0000AACF@Data
Address3_MRAM	unsigned int	100	0x0000AAC3@Data
Address4_MRAM	unsigned int	146	0x0000AAC4@Data
Address5_MRAM	unsigned int	250	0x0000AAC9@Data
Address6_MRAM	unsigned int	500	0x0000AAC1@Data
Address7_MRAM	unsigned int	1000	0x0000AAC0@Data
Address8_MRAM	unsigned int	0	0x0000AAC5@Data
Address9_MRAM	unsigned int	2	0x0000AAC2@Data
Address10_MRAM	unsigned int	21262	0x0000AAC8@Data

Figure 3.45 : READ10 CCS expression view.

It is observed that the Address10_MRAM value is also 21262 which is equal to the data we sent.

In this section, the accuracy of MRAM Continuous Write Functions were investigated by using ten random addresses of 1st and 2nd MRAM. A dummy error message sized of 960.000 bytes was written at MRAMs in such a way that 480.000 bytes to the first and 480.000 to the second MRAM. The success rate of the functions were evaluated by watching the SPI bus with digital oscilloscope and controlling the return values of MRAM read functions that were assigned by software. So, the results were double checked and it has been clearly seen that the results are identical with the values given in Table 3.2.

Error signals consisting of 24 different sensor data to be recorded within the scope of the thesis consist of 2 parts. While the MRAM1 Continuous Write function is used to simulate the first part of 480.000 bytes, which contains 2 seconds before the error occurs, the MRAM2 Continuous Write function creates the second 480.000 bytes part.

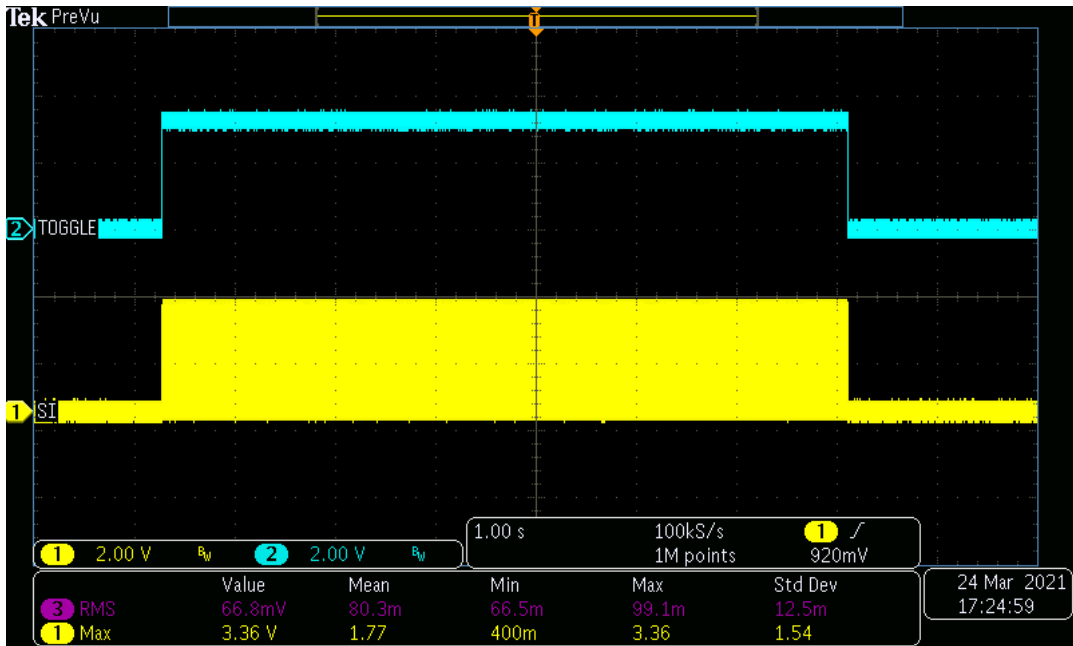


Figure 3.46 : MRAMs continuous write function oscilloscope view.

An unused I/O has been configured to monitor the time interval between start and stop time, and this I/O is labelled TOGGLE on the oscilloscope interface as seen in Figure 3.46. The run time of the continuous write function will be optimized by erasing the control mechanisms and check operations unneeded for the final operation. The SPI bus was zoomed in to check if there is meaningful data. It was confirmed at this step that data writing operations were performed successfully as observed in the accuracy tests as well. Zoomed oscilloscope views are shown in Figures 3.47, 3.48, 3.49.

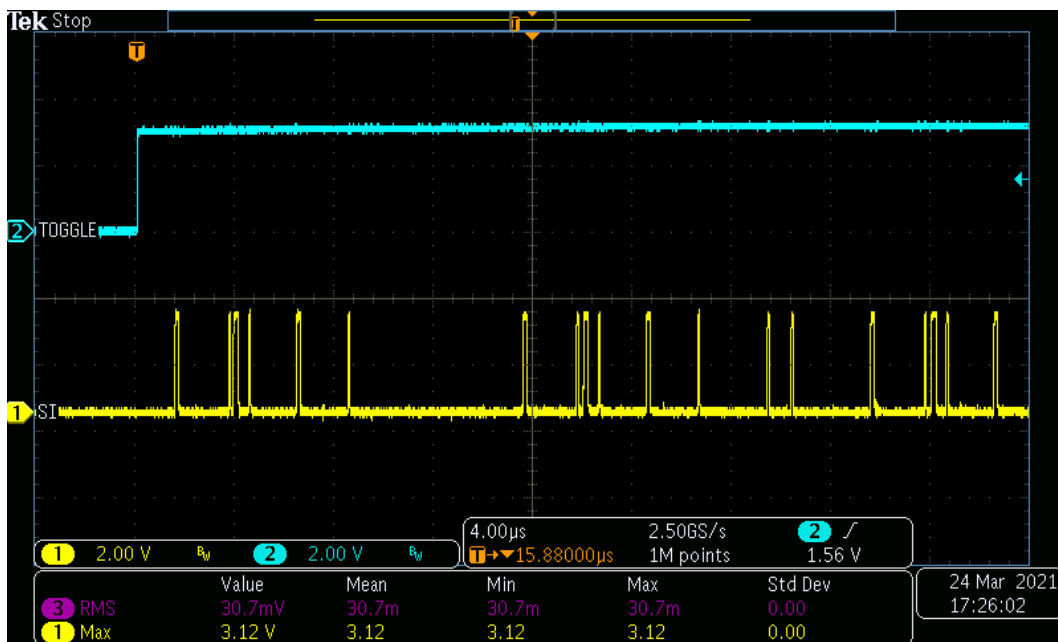


Figure 3.47 : SPI bus zoomed oscilloscope view-1.

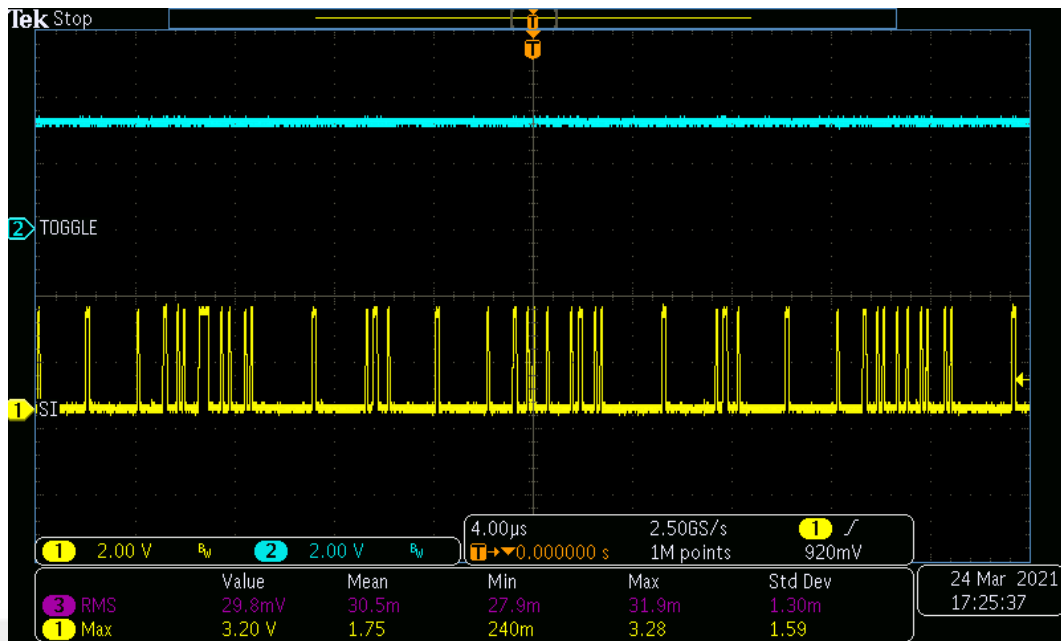


Figure 3.48 : SPI bus zoomed oscilloscope view-2.

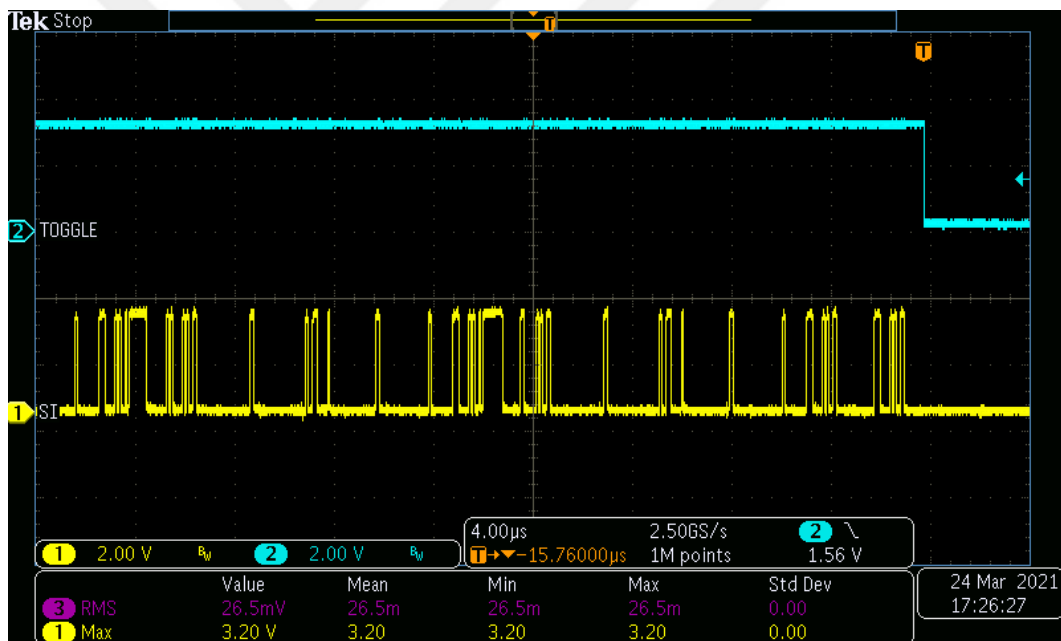


Figure 3.49 : SPI bus zoomed oscilloscope view-3.

3.3 EEPROM Write & Read Functions Test Results

In this section, the test software for the FLASH writing and reading functions designed in section 2.2.3 were built and the tests were performed. Ten different data and FLASH1 addresses were chosen randomly and the test data were written in that addresses. Right after writing functions, reading functions were run as well in the following line. Two breakpoints were set in the beginning and the end of the test

software. While the data and corresponding addresses are given in Table 3.3, test algorithm of STAGE: 4 is demonstrated in Figure 3.50.

Table 3.3 : EEPROM write & read functions test details.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	1	0000000000000001	1	1
Read2 EEPROM	8	0000000000001000	3	1
Read3 EEPROM	10	0000000000001010	5	1
Read4 EEPROM	315	0000000100111011	10	1
Read5 EEPROM	500	0000000111110100	50	1
Read6 EEPROM	46	0000000000101110	100	1
Read7 EEPROM	47	0000000000101111	1000	1
Read8 EEPROM	48	0000000000110000	500	1
Read9 EEPROM	49	0000000000110001	175	1
Read10 EEPROM	50	0000000000110010	480500	1

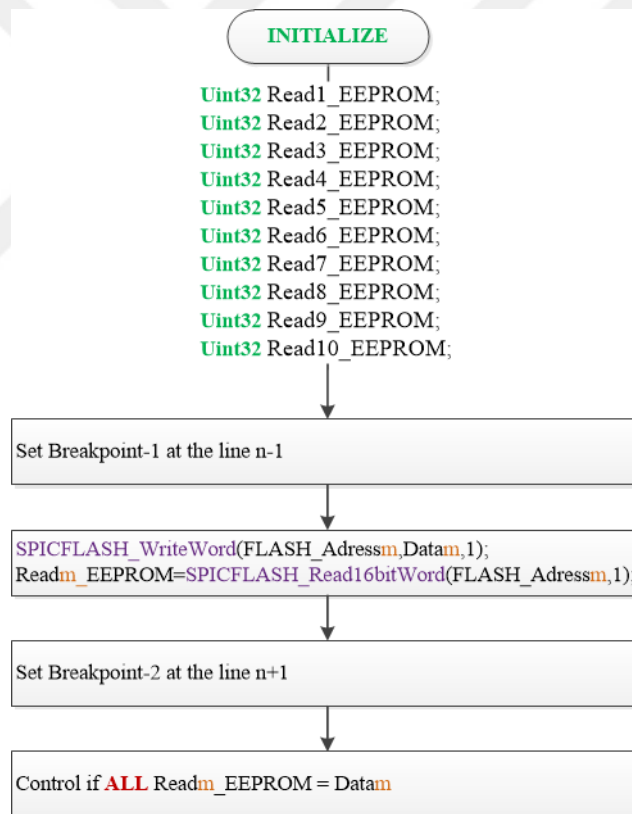


Figure 3.50 : FLASH read and write test procedure.

The results were both checked by using CCS interface and oscilloscope by setting breakpoints. Signals were analyzed in the time interval by debugging between the breakpoints. The oscilloscope results are given respectively by starting from READ1. However, the results monitored by using CCS Expression View Window are given on

one screen to avoid repetition, because CCS Expression View's manner of work was already shown and explained step by step in previous sections:

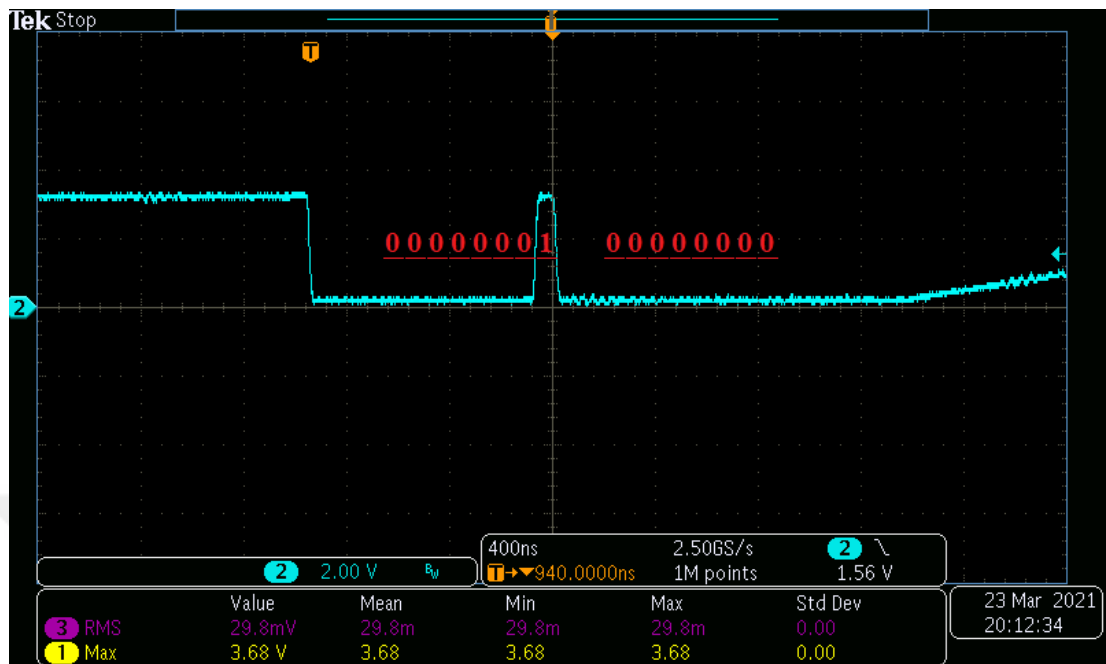


Figure 3.51 : READ1 oscilloscope view interpreted.

It is shown in Figure 3.51 that the data read from address 1 of FLASH1 is 0000000000000001 which is equal to 1 in decimal.

READ2:

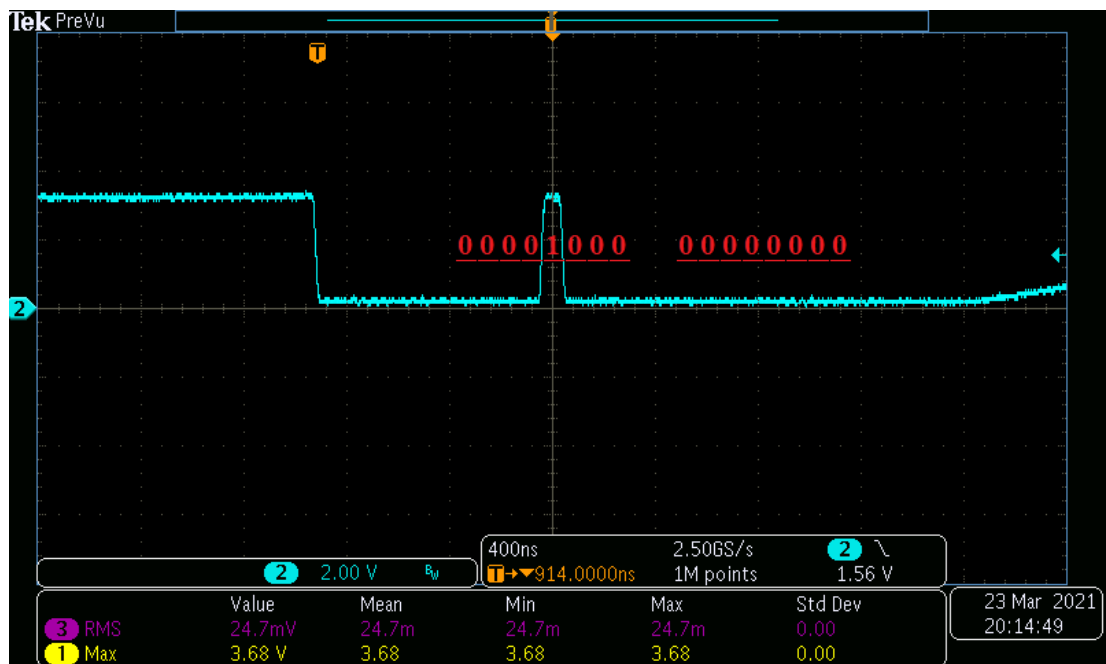


Figure 3.52 : READ2 oscilloscope view interpreted.

It is shown in Figure 3.52 that the data read from address 3 of FLASH1 is 0000000000001000 which is equal to 8 in decimal.

READ3:

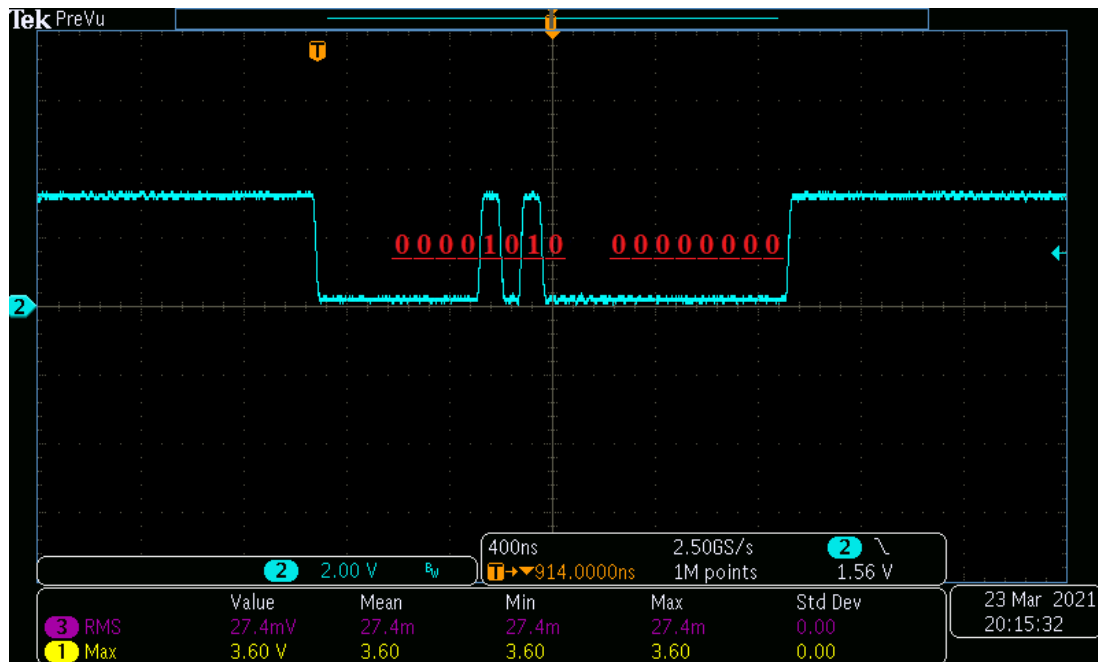


Figure 3.53 : READ3 oscilloscope view interpreted.

It is shown in Figure 3.53 that the data read from address 5 of FLASH1 is 0000000000001010 which is equal to 10 in decimal.

READ4:

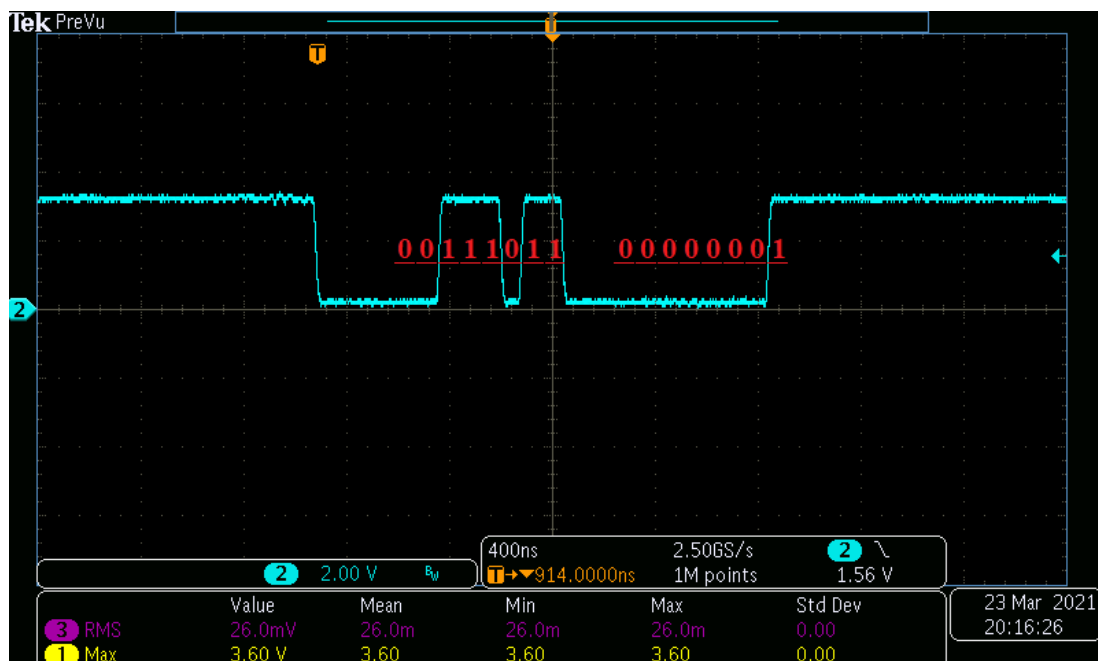


Figure 3.54 : READ4 oscilloscope view interpreted.

It is shown in Figure 3.54 that the data read from address 10 of FLASH1 is 0000000100111011 which is equal to 315 in decimal.

READ5:

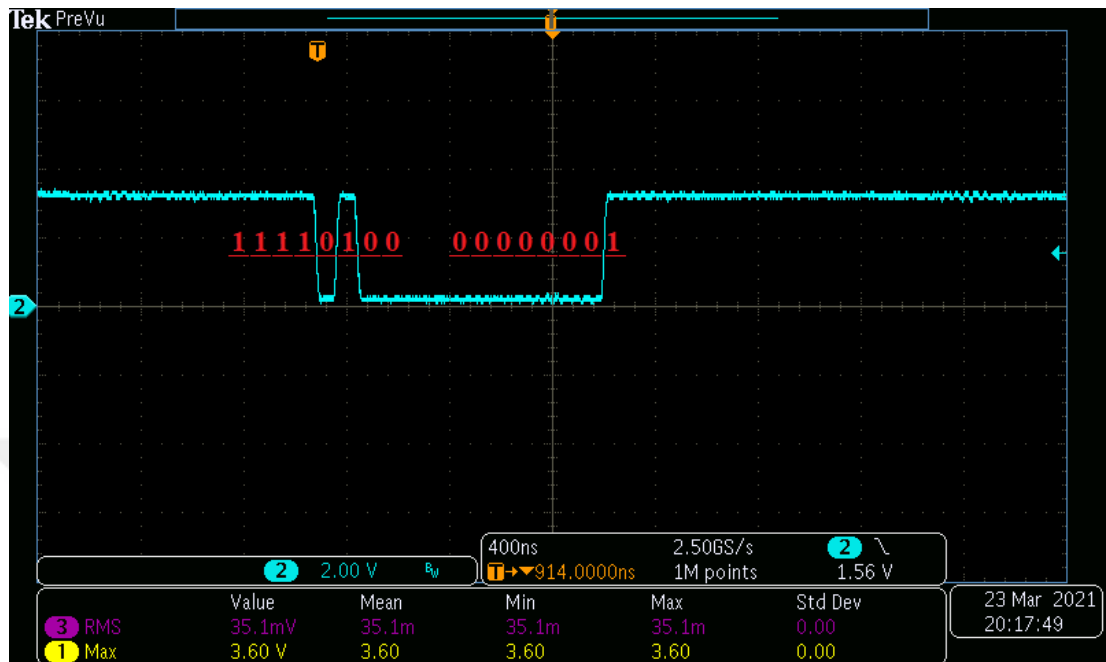


Figure 3.55 : READ5 oscilloscope view interpreted.

It is shown in Figure 3.55 that the data read from address 50 of FLASH1 is 0000000111110100 which is equal to 500 in decimal.

READ6:

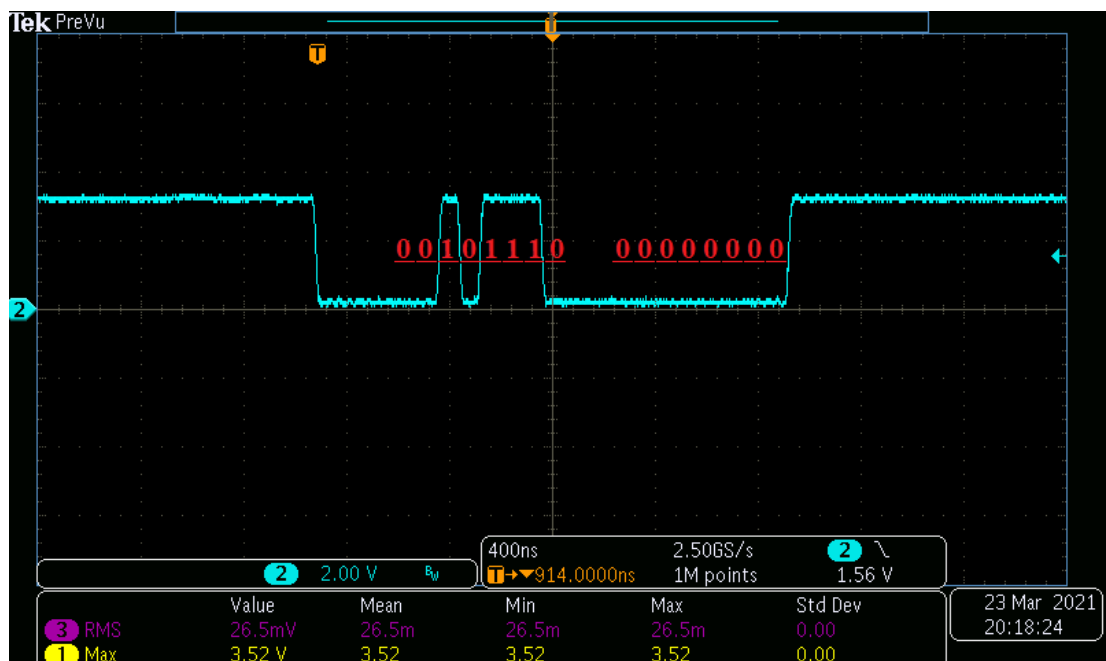


Figure 3.56 : READ6 oscilloscope view interpreted.

It is shown in Figure 3.56 that the data read from address 100 of FLASH1 is 0000000000101110 which is equal to 46 in decimal.

READ7:

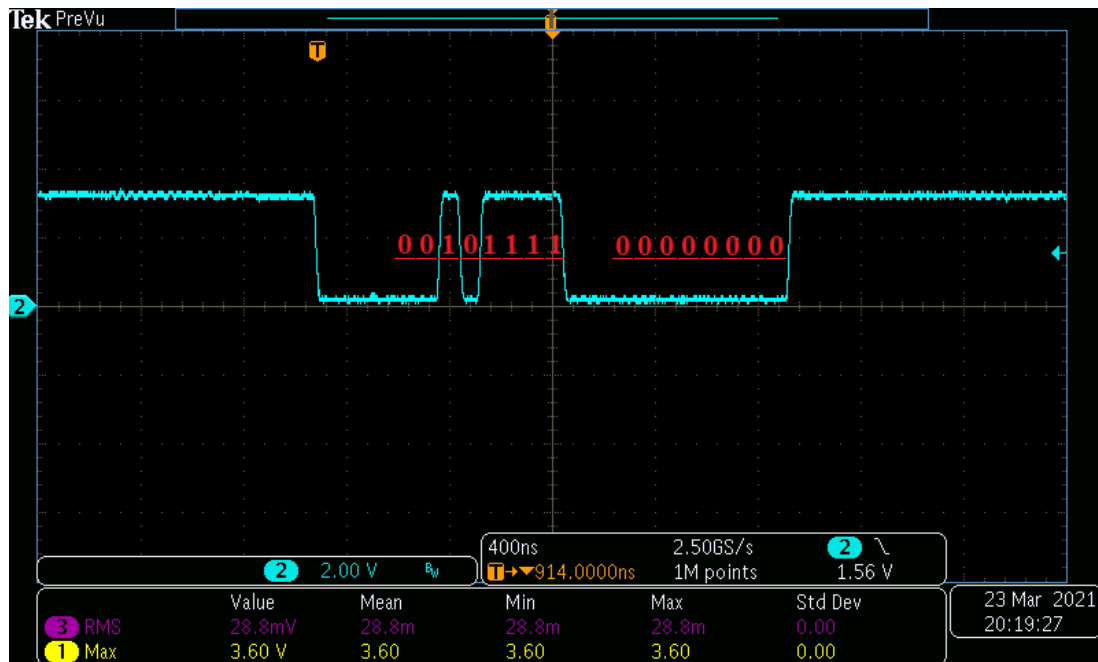


Figure 3.57 : READ7 oscilloscope view interpreted.

It is shown in Figure 3.57 that the data read from address 100 of FLASH1 is 0000000000101111 which is equal to 47 in decimal.

READ8:

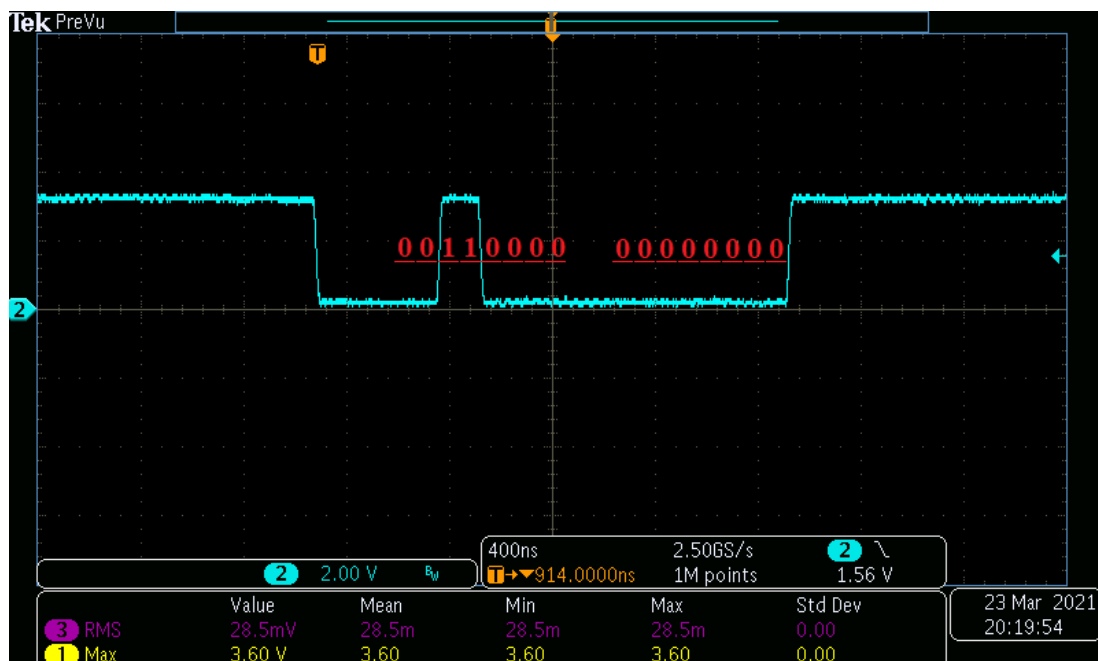


Figure 3.58 : READ8 oscilloscope view interpreted.

It is shown in Figure 3.58 that the data read from address 100 of FLASH1 is 0000000000110000 which is equal to 48 in decimal.

READ9:

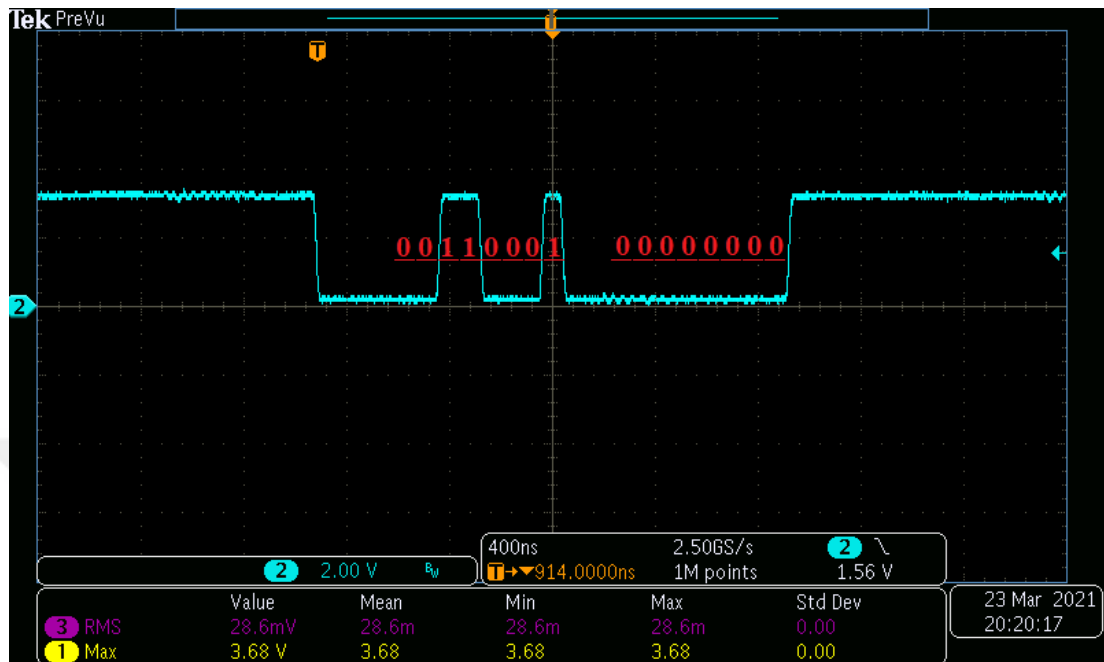


Figure 3.59 : READ9 oscilloscope view interpreted.

It is shown in Figure 3.59 that the data read from address 100 of FLASH1 is 0000000000110001 which is equal to 49 in decimal.

READ10:

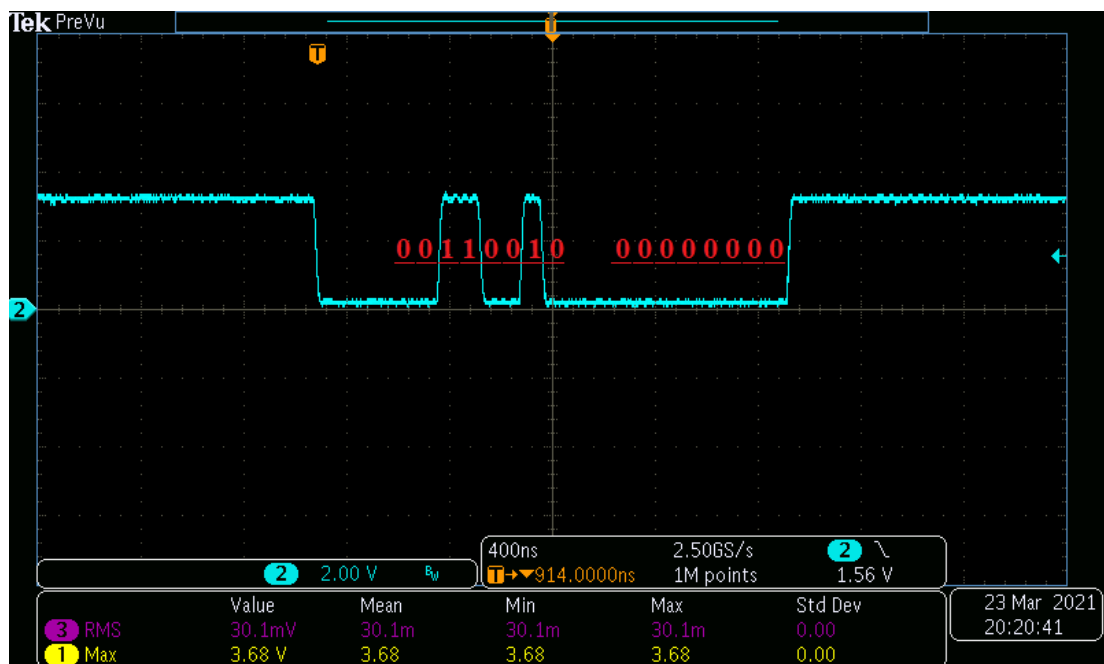


Figure 3.60 : READ10 oscilloscope view interpreted.

It is shown in Figure 3.60 that the data read from address 100 of FLASH1 is 0000000000110010 which is equal to 50 in decimal.

Moreover, the words read were assigned to the expression from Read1_EEPROM to Read10_EEPROM in order and viewed on CCS Expression View Window. The CCS window after debugging in 2.breakpoint is given in Figure 3.61.

Expression	Type	Value	Address
Read1_EEPROM	unsigned int	1	0x0000AA...
Read2_EEPROM	unsigned int	8	0x0000AA...
Read3_EEPROM	unsigned int	10	0x0000AA...
Read4_EEPROM	unsigned int	315	0x0000AA...
Read5_EEPROM	unsigned int	500	0x0000AA...
Read6_EEPROM	unsigned int	46	0x0000AA...
Read7_EEPROM	unsigned int	47	0x0000AA...
Read8_EEPROM	unsigned int	48	0x0000AA...
Read9_EEPROM	unsigned int	49	0x0000AA...
Read10_EEPROM	unsigned int	50	0x0000AA...

Figure 3.61 : All reading results on CCS expression view.

In the context of this section, FLASH Write and Read Functions were tested. Ten different data values and addresses were chosen randomly and writing operations were performed. The success rate of the functions were evaluated by watching the SPI bus with digital oscilloscope and controlling the return values of FLASH Read Function that were assigned by software. In this way, the results were checked by two different methods and it has been clearly seen that the results were identical with the values given in Table 3.3. As the result of the tests, it has been concluded that the writing and reading functions of FLASH have been created successfully, software runs properly.

3.4 EEPROM Erase Functions Test Results

In this section, a test software for the FLASH erase functions which creation steps were described in section 2.2.4 were created and has been run in the microprocessor-based ICU setup.

In the first test, the success and functionality of Bulk Erase Function were tested and noted. Ten different FLASH1 addresses were chosen and the random test data were written in that addresses. Right after the writing functions, reading functions were run as well in the following line by debugging with breakpoints. After all written data were

checked by using FLASH read functions, Bulk Erase Function was executed and reading processes were performed again. The data should be read from the corresponding addresses before and after bulk erasing are given in Table 3.4 and 3.5 respectively:

Table 3.4 : Initial test details.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	1	0000000000000001	2	1
Read2 EEPROM	8	0000000000001000	400	1
Read3 EEPROM	10	0000000000001010	8000	1
Read4 EEPROM	8	0000000000001000	1000	1
Read5 EEPROM	10	0000000000001010	65534	1
Read6 EEPROM	10	0000000000001010	1048581	1
Read7 EEPROM	10	0000000000001010	1200000	1
Read8 EEPROM	10	0000000000001010	1200500	1
Read9 EEPROM	10	0000000000001010	1205500	1
Read10 EEPROM	10	0000000000001010	1505500	1

Table 3.5 : Expected results after bulk erasing.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	65535	1111111111111111	2	1
Read2 EEPROM	65535	1111111111111111	400	1
Read3 EEPROM	65535	1111111111111111	8000	1
Read4 EEPROM	65535	1111111111111111	1000	1
Read5 EEPROM	65535	1111111111111111	65534	1
Read6 EEPROM	65535	1111111111111111	1048581	1
Read7 EEPROM	65535	1111111111111111	1200000	1
Read8 EEPROM	65535	1111111111111111	1200500	1
Read9 EEPROM	65535	1111111111111111	1205500	1
Read10 EEPROM	65535	1111111111111111	1505500	1

The results were both controlled by using the same methods used in previous tests. The observed results were given on one CCS window and one oscilloscope screen to avoid repetition. Values read after the related write operations were performed (before bulk erase):

Expression	Type	Value	Address
Read1_EEPROM	unsigned int	1	0x0000AAD2@Data
Read2_EEPROM	unsigned int	8	0x0000AAD1@Data
Read3_EEPROM	unsigned int	10	0x0000AACD@Data
Read4_EEPROM	unsigned int	8	0x0000AACC@Data
Read5_EEPROM	unsigned int	10	0x0000AACF@Data
Read6_EEPROM	unsigned int	10	0x0000AACE@Data
Read7_EEPROM	unsigned int	10	0x0000AAC2@Data
Read8_EEPROM	unsigned int	10	0x0000AAC4@Data
Read9_EEPROM	unsigned int	10	0x0000AAC3@Data
Read10_EEPROM	unsigned int	10	0x0000AAD0@Data
+ Add new expression			

Figure 3.62 : All reading results on CCS expression view.

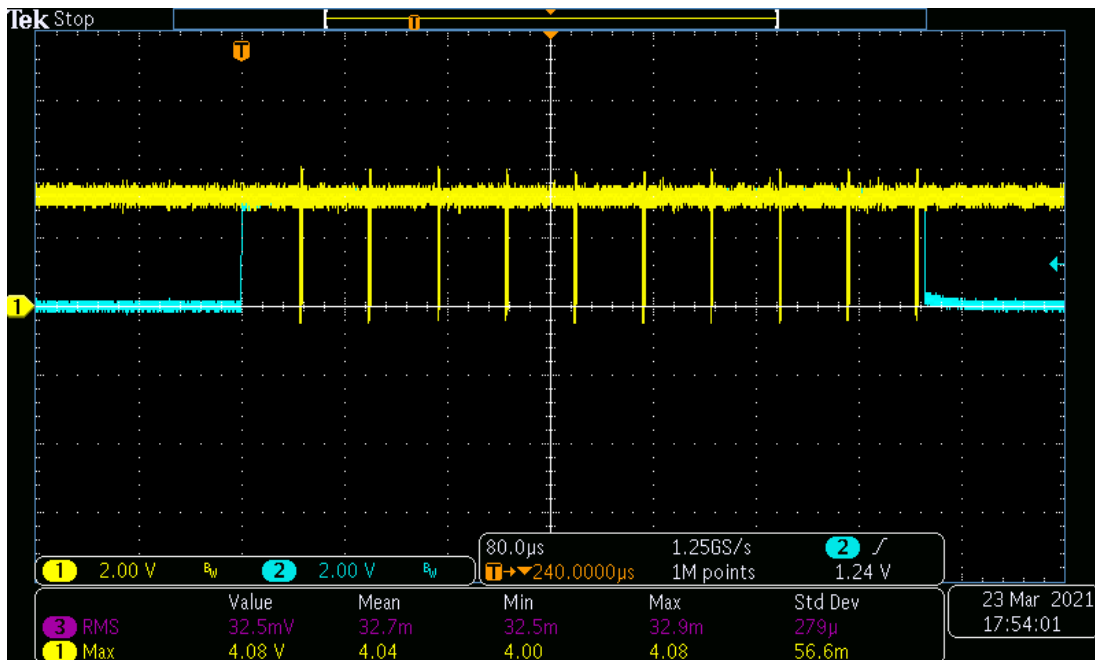


Figure 3.63 : All reading results on oscilloscope screen.

As seen in the related figures, the values obtained as a result of the read operation performed at the corresponding addresses are the same as the written values. In this step of the test, the same addresses were read again after Bulk Erase Function was run and the results are given in Figure 3.64 and 3.65:

Expression	Type	Value	Address
Read1_EEPROM	unsigned int	65535	0x0000AAD1@Data
Read2_EEPROM	unsigned int	65535	0x0000AACE@Data
Read3_EEPROM	unsigned int	65535	0x0000AACD@Data
Read4_EEPROM	unsigned int	65535	0x0000AAD0@Data
Read5_EEPROM	unsigned int	65535	0x0000AACF@Data
Read6_EEPROM	unsigned int	65535	0x0000AAD2@Data
Read7_EEPROM	unsigned int	65535	0x0000AAC1@Data
Read8_EEPROM	unsigned int	65535	0x0000AAC2@Data
Read9_EEPROM	unsigned int	65535	0x0000AAC4@Data
Read10_EEPROM	unsigned int	65535	0x0000AAC0@Data
+ Add new expression			

Figure 3.64 : All reading results on CCS expression view after bulk erasing.

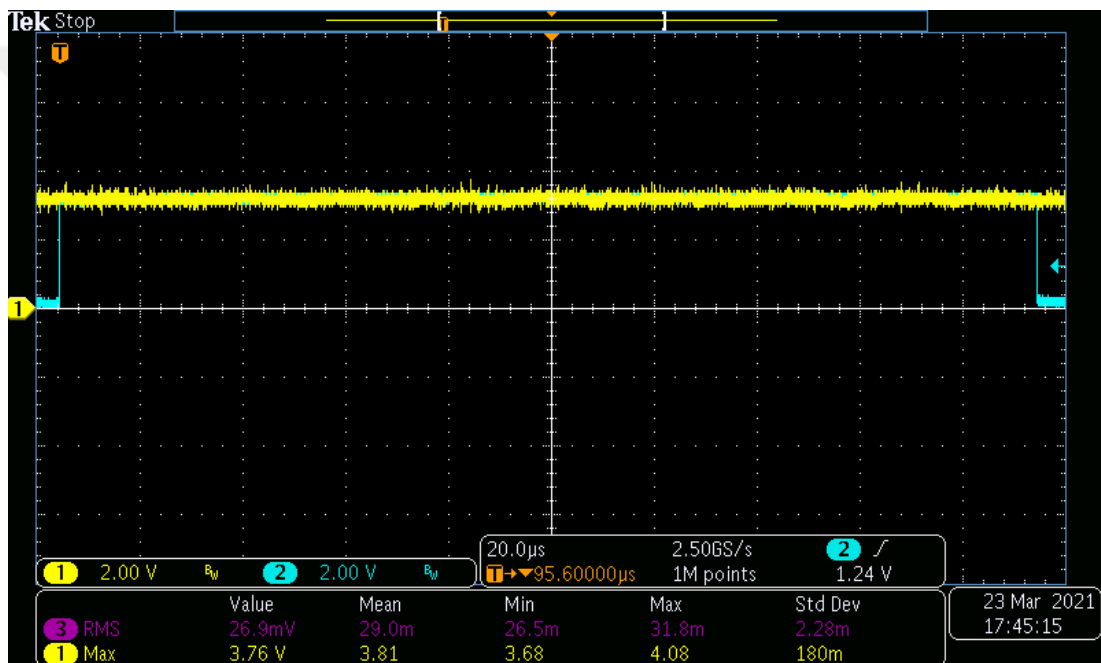


Figure 3.65 : All reading results on oscilloscope screen after bulk erasing.

Erase functions set the corresponding device bits to FFh as stated before. In bulk erasing, the corresponding addresses (or device bits) mean all the addresses of an EEPROM.

As seen in Figure 3.64, 3.65, the results are in concordance with the logic of bulk erasing and the expected values in Table 3.5 that were shown as 65536 which means that all of the bits are 1 in binary. In brief, it has been clearly seen that EEPROM Bulk Erase Function works properly.

Within the second test, the success and performance of Sector Erase Function were investigated. Ten FLASH1 addresses were chosen and test data were written in that

addresses again. In order to see the differences between the erasing function types more clearly, the same values are written to the same addresses in Table 3.4. As the first step of the test, these addresses were read with breakpoints and it was confirmed that the values were exactly the same as the values in Table 3.4. Results were given in Figure 3.66:

Expression	Type	Value	Address
Read1_EEPROM	unsigned int	1	0x0000AAD0@Data
Read2_EEPROM	unsigned int	8	0x0000AAD1@Data
Read3_EEPROM	unsigned int	10	0x0000AAD2@Data
Read4_EEPROM	unsigned int	8	0x0000AACF@Data
Read5_EEPROM	unsigned int	10	0x0000AACD@Data
Read6_EEPROM	unsigned int	10	0x0000AACC@Data
Read7_EEPROM	unsigned int	10	0x0000AAC1@Data
Read8_EEPROM	unsigned int	10	0x0000AAC2@Data
Read9_EEPROM	unsigned int	10	0x0000AAC4@Data
Read10_EEPROM	unsigned int	10	0x0000AAC3@Data
+ Add new expression			

Figure 3.66 : All reading results on CCS expression view.

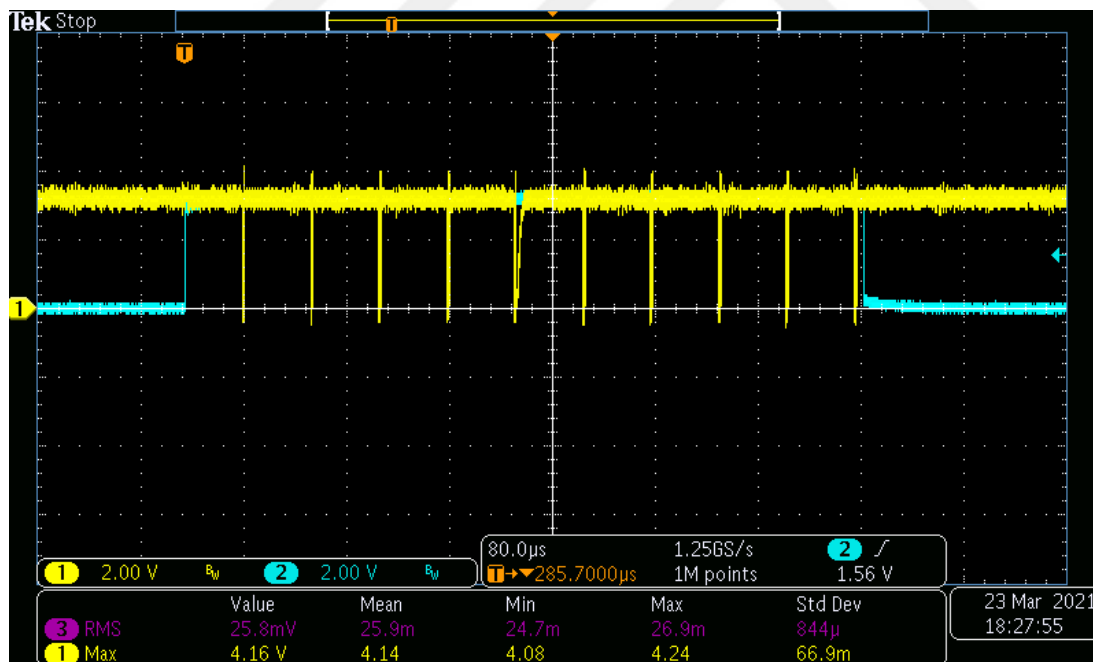


Figure 3.67 : All reading results on oscilloscope screen.

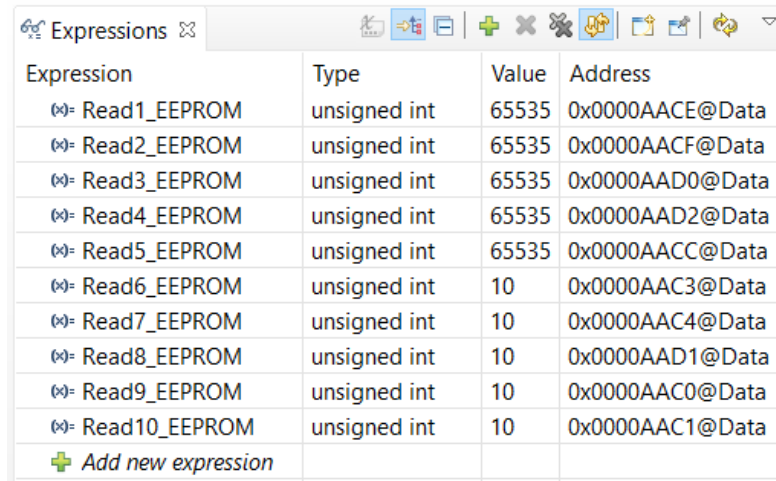
After the processes before the sector erasing were confirmed by using FLASH read functions, Sector Erase Function was run and the results were checked by using read functions. As stated earlier, a sector is a 65536 Byte data field customized in the EEPROM. The test addresses were chosen by paying special attention to not being in

the same sector. Thus, it will be possible to observe whether the Sector Delete function only deletes the addresses belonging to the relevant sector and does not make any changes to the data in the addresses out of the sector. In the sector erase test, the first sector was deleted. In this case, the expected values after the corresponding erase function are as in Table 3.6:

Table 3.6 : Expected results after sector erasing.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	65535	1111111111111111	2	1
Read2 EEPROM	65535	1111111111111111	400	1
Read3 EEPROM	65535	1111111111111111	8000	1
Read4 EEPROM	65535	1111111111111111	1000	1
Read5 EEPROM	65535	1111111111111111	65534	1
Read6 EEPROM	10	0000000000001010	1048581	1
Read7 EEPROM	10	0000000000001010	1200000	1
Read8 EEPROM	10	0000000000001010	1200500	1
Read9 EEPROM	10	0000000000001010	1205500	1
Read10 EEPROM	10	0000000000001010	1505500	1

In this step of the test, the same addresses were read again after Sector Erase Function was run. The results are given in Figure 3.68 and 3.69:



Expression	Type	Value	Address
Read1_EEPROM	unsigned int	65535	0x0000ACE@Data
Read2_EEPROM	unsigned int	65535	0x0000ACF@Data
Read3_EEPROM	unsigned int	65535	0x0000AAD0@Data
Read4_EEPROM	unsigned int	65535	0x0000AAD2@Data
Read5_EEPROM	unsigned int	65535	0x0000AACC@Data
Read6_EEPROM	unsigned int	10	0x0000AAC3@Data
Read7_EEPROM	unsigned int	10	0x0000AAC4@Data
Read8_EEPROM	unsigned int	10	0x0000AAD1@Data
Read9_EEPROM	unsigned int	10	0x0000AAC0@Data
Read10_EEPROM	unsigned int	10	0x0000AAC1@Data
+ Add new expression			

Figure 3.68 : All reading results on CCS expression view after sector erasing.

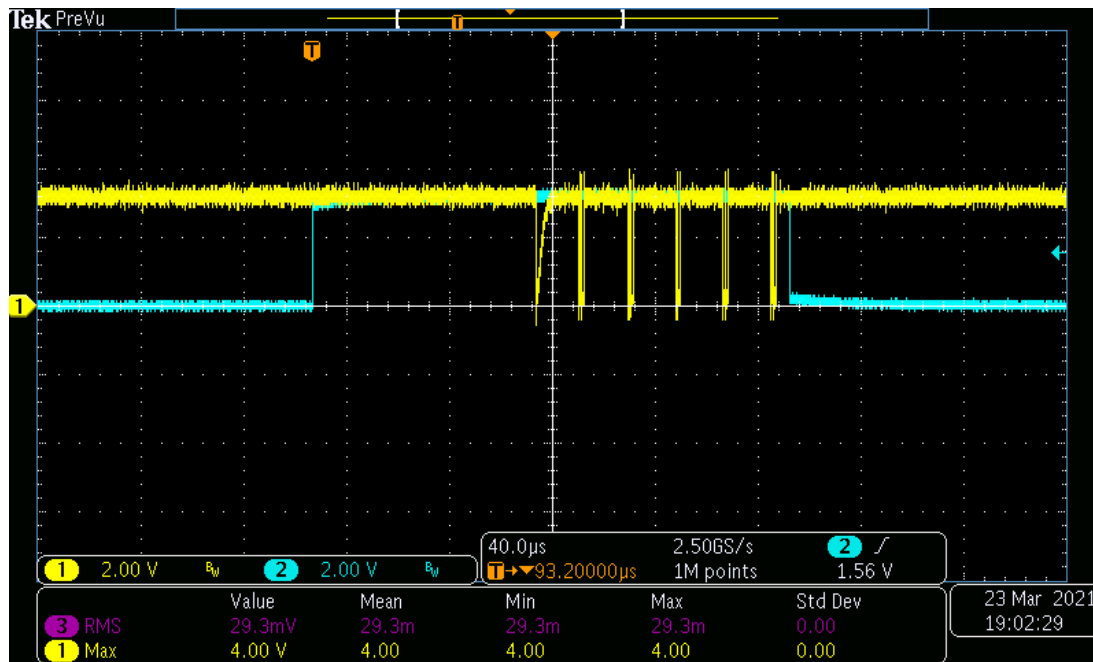


Figure 3.69 : All reading results on oscilloscope screen after sector erasing.

As seen in Figure 3.68, 3.69, the results are the same as the values in expected values in Table 3.6. Consequently, it has been clearly seen that EEPROM Sector Erase Function works properly.

As stated in the previous sections, 4-second error messages will be recorded into the memory areas named as slot. However the related slot must be available for writing, because the addresses are protected for overwriting in FLASH ICs. Therefore, necessary memory size (next slot) must be erased before next write in case of no space in diagnostic memory (75 error messages are already written). In this respect, the primary erase function is the "Slot Erase" function in fact, since it will be the deletion function to be used frequently in final software.

Within the scope of last erase tests, the success and performance of Slot Erase Function were tested. Ten different FLASH1 addresses were chosen and the random test data were written in that addresses. Right after the writing functions, reading functions were run as well in the following line by debugging with breakpoints. After all written data were checked by using FLASH read functions, Slot Erase Function was executed and relevant addresses were read again. The data should be read from the addresses before and after slot erasing are given in Table 3.7 and 3.8 respectively:

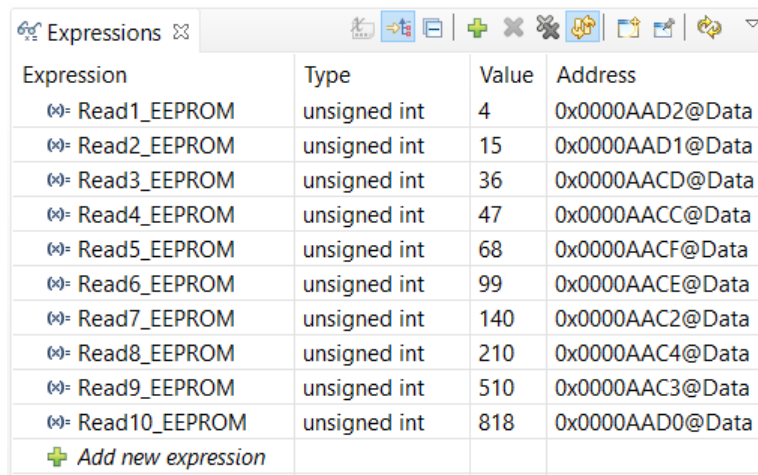
Table 3.7 : Initial test details.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	4	0000000000000100	2	1
Read2 EEPROM	15	0000000000001111	400	1
Read3 EEPROM	36	000000000100100	8000	1
Read4 EEPROM	47	000000000101111	65534	1
Read5 EEPROM	68	000000001000100	1048574	1
Read6 EEPROM	99	000000001100011	1048580	1
Read7 EEPROM	140	000000010001100	1200000	1
Read8 EEPROM	210	000000011010010	1200500	1
Read9 EEPROM	510	0000000111111110	1205500	1
Read10 EEPROM	818	0000001100110010	1505500	1

Table 3.8 : Expected results after slot erasing.

Test Step	Data in Decimal <i>Randomly Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly Chosen</i>	FLASH IC <i>Selected</i>
Read1 EEPROM	65535	1111111111111111	2	1
Read2 EEPROM	65535	1111111111111111	400	1
Read3 EEPROM	65535	1111111111111111	8000	1
Read4 EEPROM	65535	1111111111111111	65534	1
Read5 EEPROM	65535	1111111111111111	1048574	1
Read6 EEPROM	99	000000001100011	1048580	1
Read7 EEPROM	140	000000010001100	1200000	1
Read8 EEPROM	210	000000011010010	1200500	1
Read9 EEPROM	510	0000000111111110	1205500	1
Read10 EEPROM	818	0000001100110010	1505500	1

The results were both checked by use of breakpoints through CCS Expression View Window and oscilloscope. Values read before sector erasing were given in Figure 3.70 and 3.71:



Expression	Type	Value	Address
Read1_EEPROM	unsigned int	4	0x0000AAD2@Data
Read2_EEPROM	unsigned int	15	0x0000AAD1@Data
Read3_EEPROM	unsigned int	36	0x0000AACD@Data
Read4_EEPROM	unsigned int	47	0x0000AACC@Data
Read5_EEPROM	unsigned int	68	0x0000AACF@Data
Read6_EEPROM	unsigned int	99	0x0000AACE@Data
Read7_EEPROM	unsigned int	140	0x0000AAC2@Data
Read8_EEPROM	unsigned int	210	0x0000AAC4@Data
Read9_EEPROM	unsigned int	510	0x0000AAC3@Data
Read10_EEPROM	unsigned int	818	0x0000AAD0@Data
+ Add new expression			

Figure 3.70 : All reading results on CCS expression view.

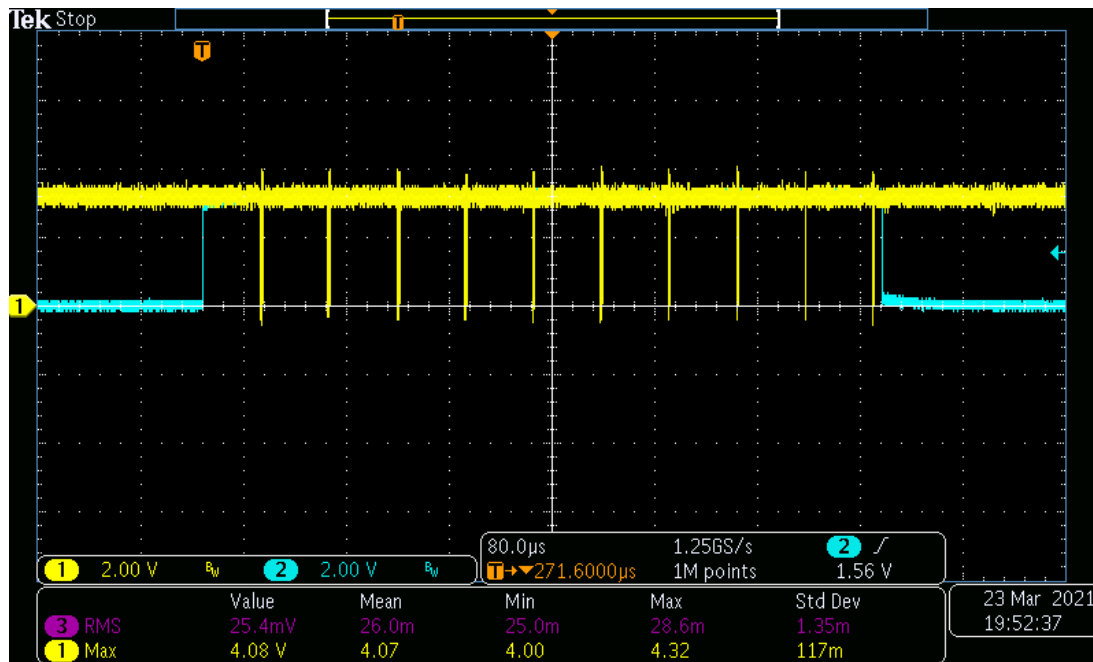


Figure 3.71 : All reading results on oscilloscope screen.

As seen in the related figures, the values obtained as the results of the read operations performed at the addresses are the same as the written values.

In this step of the test, the same addresses were read again according to the test plan. The results after Sector Erase Function was run are given in Figure 3.72, 3.73:

Expression	Type	Value	Address
Read1_EEPROM	unsigned int	65535	0x0000AACE@Data
Read2_EEPROM	unsigned int	65535	0x0000AACF@Data
Read3_EEPROM	unsigned int	65535	0x0000AAD0@Data
Read4_EEPROM	unsigned int	65535	0x0000AAD2@Data
Read5_EEPROM	unsigned int	65535	0x0000AACC@Data
Read6_EEPROM	unsigned int	99	0x0000AAC3@Data
Read7_EEPROM	unsigned int	140	0x0000AAC4@Data
Read8_EEPROM	unsigned int	210	0x0000AAD1@Data
Read9_EEPROM	unsigned int	510	0x0000AAC0@Data
Read10_EEPROM	unsigned int	818	0x0000AAC1@Data
Add new expression			

Figure 3.72 : All reading results on CCS expression view after slot erasing.

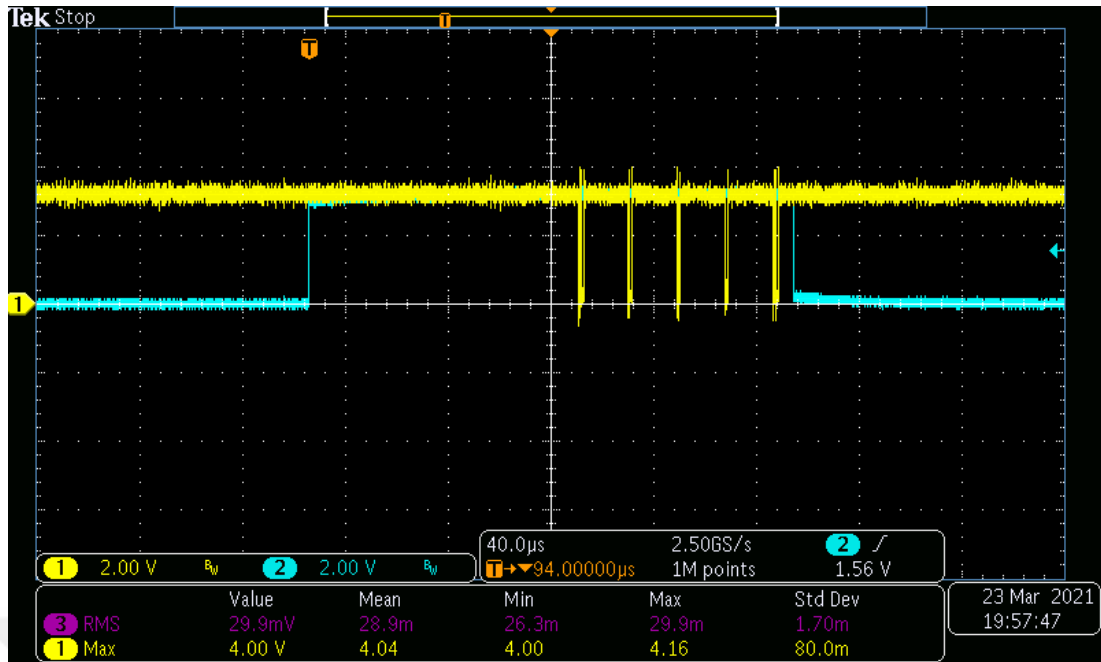


Figure 3.73 : All reading results on oscilloscope screen after slot erasing.

In the final test, the slot erase function was tested. Address selections were made by taking the slot size as a reference, as five addresses smaller and the other five being greater than the slot limit. Addresses 1048576 and 1048580 are particularly chosen for testing the slot limits.

As seen in Figure 3.72, and 3.73, data in the addresses smaller than 1048576 are 65536 that means all of the bits are 1 in binary, while the greater ones still keep their data. So, the results are same as the values in expected values in Table 3.8. Consequently, it has been clearly seen that EEPROM Slot Erase Function works properly.

In the context of this section, performance analyses of EEPROM Erase Functions were made on the microprocessor-based ICU setup. Different addresses were specified according to the erase operation type created for satisfying the project need, and test data were written randomly at that addresses. The success rate of the functions were evaluated by watching the SPI bus with digital oscilloscope and the controlling the return values of read functions that were assigned by software.

Consequently, it has been clearly seen that the results are identical with the expected values, and all the three erase functions have been performed properly; therefore, EEPROM Erase Functions were created successfully.

3.5 The Case of Error Final Tests (The Final Error Scenario Tests)

In the final test section, system-level accuracy and performance tests were made in the microprocessor-based ICU setup for the FDS. Within the scope of the test, a dummy sensor traffic was generated by using the MRAM Continuous Write Functions while the FDS was activated. When the first 480.000 byte signal part (2-second sensor data before the error occurs) was recorded, the error flag was triggered externally by force. Thus and so, the sensor recording process, which would continue for 2-seconds more after the error, was launched.

The Stop Data Record flag was automatically triggered, when all of the 4-second error message which total size of 960.000 bytes, sampled at 5 kHz frequency, was saved. By doing so, real time error logging was interrupted and the transfer from MRAMs to the corresponding EEPROM was performed. The test was completed by resetting the flags (reactivating the MRAM real time data recording), just after all of the 960.000 bytes were transmitted to the permanent error recording structure. The base algorithm for the final error test scenario is illustrated by using the flowchart demonstrated in Figure 3.74.

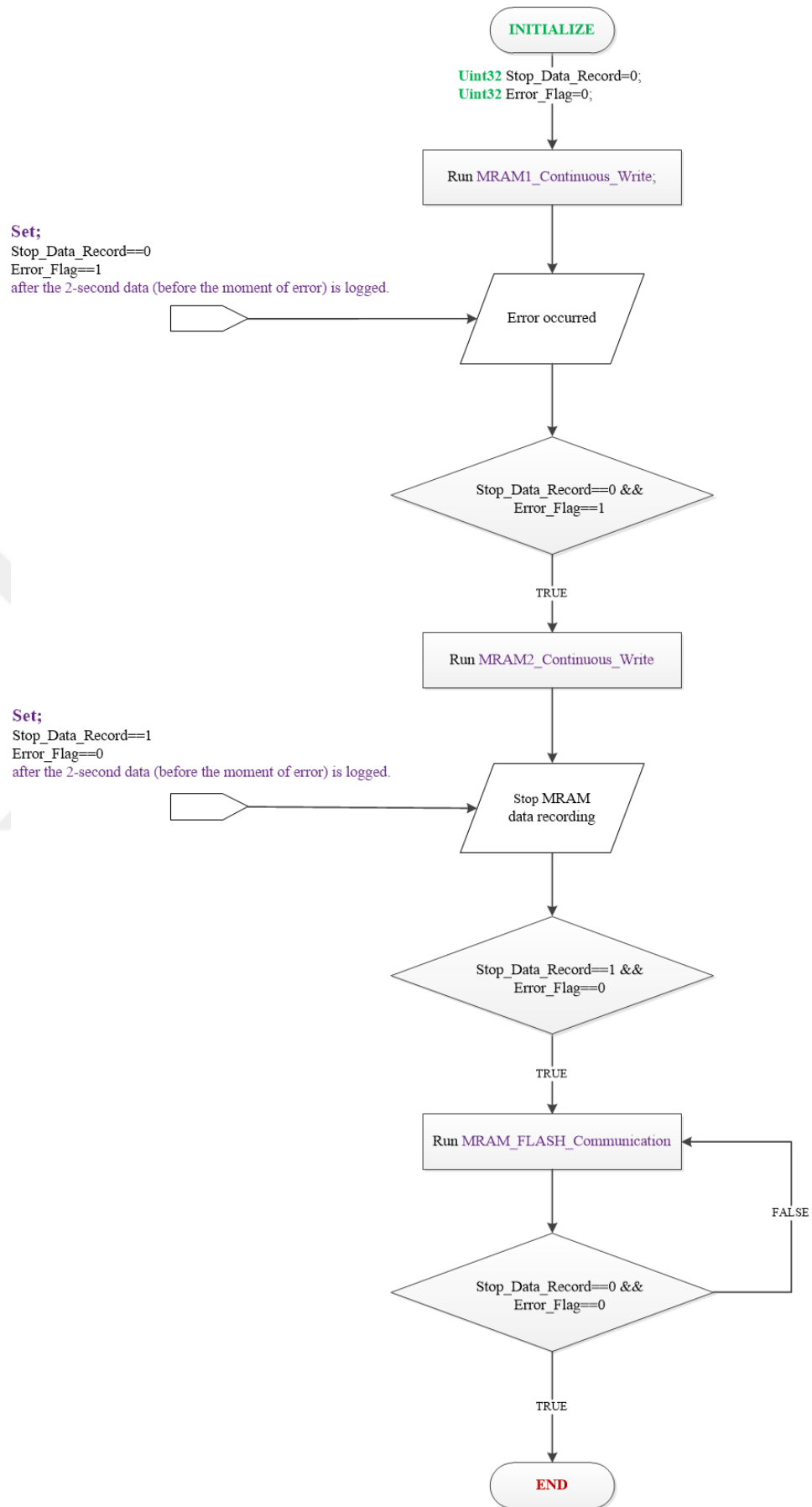


Figure 3.74 : The error recording function final test scenario flowchart.

3.5.1 Accuracy of the diagnostic software

In the first step of the final test section, the accuracy of the data recorded by the diagnostic software was tested. Here, a dummy sensor traffic was created, and an error case was simulated. Random addresses in the relevant MRAMs were selected and the corresponding addresses on the EEPROM side were determined. Reading operations were made from these addresses and return values were assigned to the expressions. The compliance of the values was checked. Ten of these addresses to be read from both MRAM and EEPROM, and expected values are given in Table 3.9.

Table 3.9 : Test details.

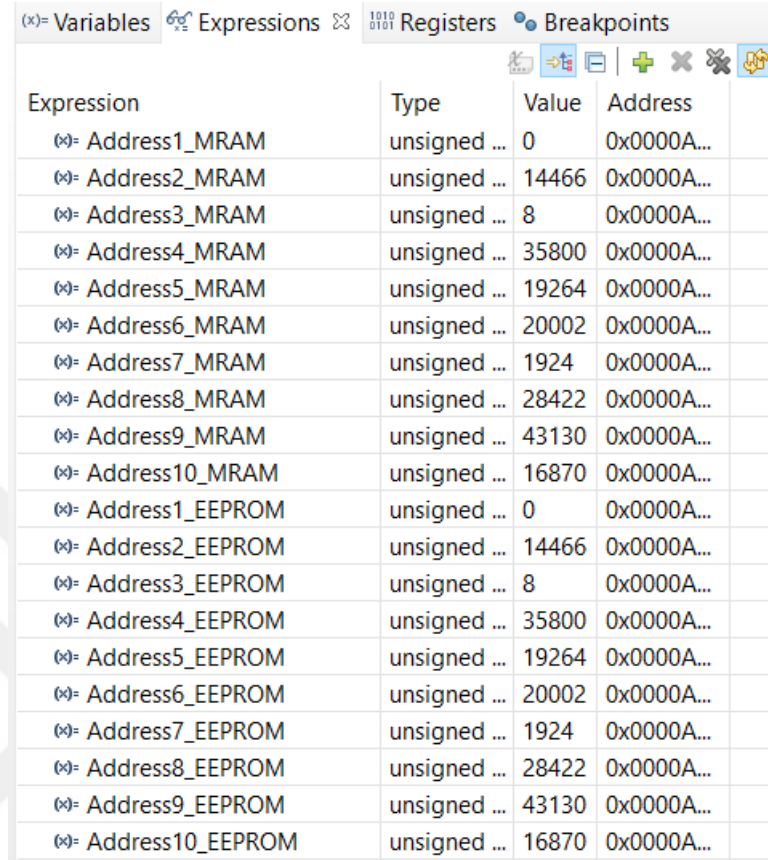
Test Step <i>Expressions</i>	Data in Decimal <i>Randomly</i> <i>Chosen</i>	Data in Binary <i>Randomly Chosen</i>	Address <i>Randomly</i> <i>Chosen</i>	IC <i>Selected</i>
Address1 MRAM	0	0000000000000000	0	1 st MRAM
Address2 MRAM	14466	0011100010000010	80000	1 st MRAM
Address3 MRAM	8	0000000000001000	8	1 st MRAM
Address4 MRAM	35800	1000101111011000	35800	1 st MRAM
Address5 MRAM	19264	0100101101000000	478002	1 st MRAM
Address6 MRAM	20002	0100111000100010	20002	2 nd MRAM
Address7 MRAM	1924	0000011110000100	198526	2 nd MRAM
Address8 MRAM	28422	0110111100000110	225024	2 nd MRAM
Address9 MRAM	43130	1010100001111010	370800	2 nd MRAM
Address10 MRAM	16870	0100000111100110	475608	2 nd MRAM
Address1 EEPROM	0	0000000000000000	0	1 st EEPROM
Address2 EEPROM	14466	0011100010000010	80000	1 st EEPROM
Address3 EEPROM	8	0000000000001000	8	1 st EEPROM
Address4 EEPROM	35800	1000101111011000	35800	1 st EEPROM
Address5 EEPROM	19264	0100101101000000	478002	1 st EEPROM
Address6 EEPROM	20002	0100111000100010	500002	1 st EEPROM
Address7 EEPROM	1924	0000011110000100	678526	1 st EEPROM
Address8 EEPROM	28422	0110111100000110	705024	1 st EEPROM
Address9 EEPROM	43130	1010100001111010	850800	1 st EEPROM
Address10 EEPROM	16870	0100000111100110	955608	1 st EEPROM

When the expected values in Table 3.9 were calculated, two important rules were taken into consideration:

1-) As explained in section 2.2.2, the operating logic of the MRAM continuous write algorithm; while address values are written to the addresses which are less than or equal to 65534, and for the values greater than 65534 the mod65534 value of the address numbers are written.

2-) Because the memory size allocated in MRAMs is 480000, the corresponding addresses of the MRAM2 in EEPROM begins with 480000.

The observed results were given on one CCS window to avoid repetition since the analysis methods were already explained step by step in previous sections:



Expression	Type	Value	Address
Address1_MRAM	unsigned ...	0	0x0000A...
Address2_MRAM	unsigned ...	14466	0x0000A...
Address3_MRAM	unsigned ...	8	0x0000A...
Address4_MRAM	unsigned ...	35800	0x0000A...
Address5_MRAM	unsigned ...	19264	0x0000A...
Address6_MRAM	unsigned ...	20002	0x0000A...
Address7_MRAM	unsigned ...	1924	0x0000A...
Address8_MRAM	unsigned ...	28422	0x0000A...
Address9_MRAM	unsigned ...	43130	0x0000A...
Address10_MRAM	unsigned ...	16870	0x0000A...
Address1_EEPROM	unsigned ...	0	0x0000A...
Address2_EEPROM	unsigned ...	14466	0x0000A...
Address3_EEPROM	unsigned ...	8	0x0000A...
Address4_EEPROM	unsigned ...	35800	0x0000A...
Address5_EEPROM	unsigned ...	19264	0x0000A...
Address6_EEPROM	unsigned ...	20002	0x0000A...
Address7_EEPROM	unsigned ...	1924	0x0000A...
Address8_EEPROM	unsigned ...	28422	0x0000A...
Address9_EEPROM	unsigned ...	43130	0x0000A...
Address10_EEPROM	unsigned ...	16870	0x0000A...

Figure 3.75 : Accuracy tests CCS expression view.

Within the context of these tests, the accuracy of the FDS was tested by generating a dummy traffic between ICU and sensors. In consequence of the test, it has been clearly seen that the data written into the MRAMs and EEPROM are identical. Furthermore, that data are also same with the values given in Table 3.9. In other words, it has been observed that there is no difference between the source and destination data, no bad and/or missed data packets. It has been concluded that the FDS designed with in the scope of the thesis works faultlessly.

3.5.2 Performance of the diagnostic software

The FDS developed as microprocessor-based, has a fixed architecture consisting of one CPU reserved for communication protocols and one SPI bus due to the fact that the system structure is not reconfigurable and many I/Os are currently used or assigned to different tasks. In this case, the running time of the function is as important as the accuracy of the recorded data checked in the previous tests. As explained in the

previous sections, the real time error recording cannot be executed during the transfer of the error message from MRAMs to EEPROM permanent recording structures. Since the microprocessor and the bus are busy with the traffic between MRAMs and EEPROM, new sensor data cannot be recorded through the duration between the start of error transfer process and the reactivation of the real-time error record. This means that new errors that occurs during the relevant process will be missed. Therefore, the runtime (or execution time) of the EEPROM – MRAM Communication Function is also crucial.

In this section, the time interval which new sensor measurements that cannot be performed (the time period that new errors cannot be recorded) was analyzed according to the test results and possible improvements were done after the tests. The first tests were started without any changes in the software used in the accuracy tests.

Two breakpoint were set before and right after the error flag status changes in the software and a configured I / O was toggled at these points. In oscilloscope screen, slave output (SO, MISO or SOMI), slave input (SI, MOSI or SIMO) buses were displayed and analyzed. In addition, the configured I / O was added to the oscilloscope screen with the name of TOGGLE. Thus, the start and end points of the transfer function was marked by using oscilloscope cursors and run time was measured. The period between the two breakpoints is given in Figure 3.76.

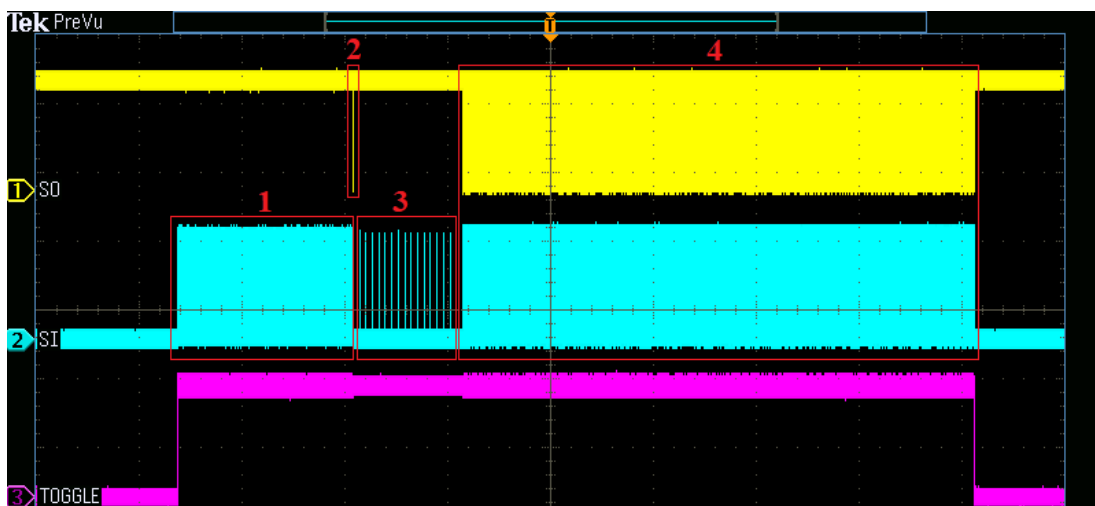


Figure 3.76 : Error recording process oscilloscope view.

The first measurement performed between error flag status changes is given in Figure 3.76. The important parts are highlighted with red colour in Figure and their explanations are given below according to the numbers respectively:

1. In this part, it has been seen that the sensor recording transaction which should be performed for 2-seconds more after the error, continued as it should be.
2. This instantaneous data seen on the SO bus is the part of where the slot number to which the related error message should be transferred (where the error message should be registered) after the MRAM error recording process is completed has been learned.
3. This part is the section that the necessary preparations steps are done in the EEPROM before the transfer begins. Here, the relevant slot which its number learned in the 2nd part was deleted with related commands sending on SI bus. By doing so, the EEPROM was made ready for error recording.
4. In the last part, the relevant twenty-four sensor data (error message) recorded into MRAMs were read from MRAMs and written to the corresponding EEPROM slot. The commands, addresses send, read and written data were seen clearly on SI and SO buses. Additionally, the corresponding counters and flags were configured properly.

In the first test, the running time of transfer an error message was measured as 24.28 seconds. The result of the relevant test is given in Figure 3.77 as an example.

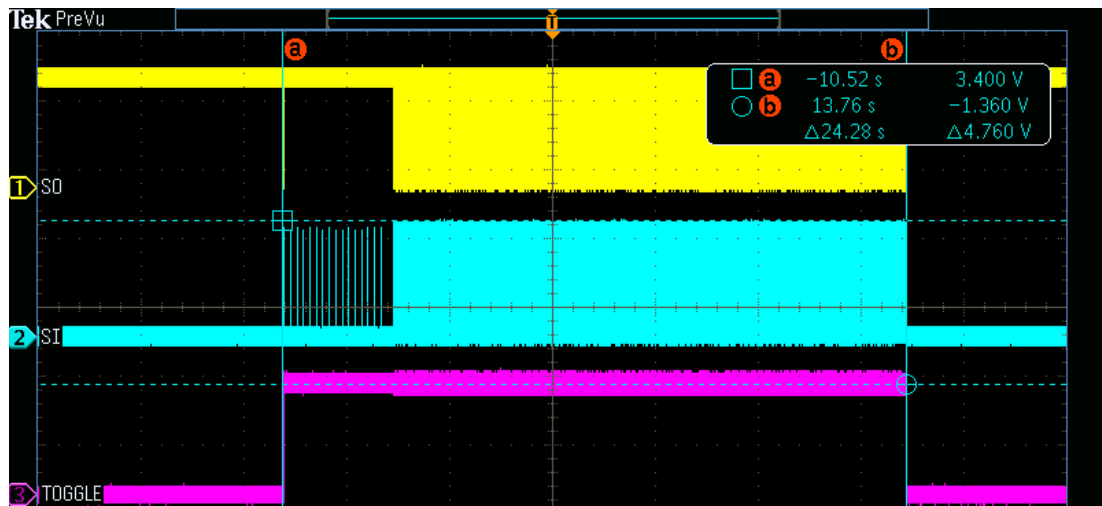


Figure 3.77 : The error recording period.

The test was repeated many times, and it was observed that the first version of EEPROM – MRAM Communication executed for permanent error recording lasted 24s on average.

In the accuracy tests, some expressions were created and read functions for every written address were run after writing processes. Read data were assigned to these expressions and the written data was checked in this way. However, the extra reading functions that were added the software on the purpose of verification (accuracy control), also have important effects on the run time. Since the software already has been verified, the extra reading functions were removed in the second revision of the software in order to eliminate these effects and reach optimized results.

As explained in the previous sections, the write and read operations in MRAM and EEPROM ICs are performed as 16-bit packets. However, it has been detected that there is no need to repeat the processes such as triggering the chip select, sending write-read commands, which must be done before these operations for the relevant devices in sequential operations. In other words, it has been seen that the same function can be run sequentially as long as the relevant addresses are sent consecutively. The process could be executed repeatedly without any need of selecting the device or operation again, after the chip selection and the command are sent once for the MRAM or EEPROM IC.

For instance, when the relevant IC is selected for reading and the read command is sent once, the data in the all addresses can be read, as long as the addresses are sent respectively and the selection bit is not toggled (disabled the relevant integrated). Similarly, after the chip select and the write command is transmitted to the relevant device one time, all data values could be written to the addresses sent sequentially until the operation is aborted.

After determining this situation, new optimizations and improvements were made in EEPROM - MRAM Communication Function. In the context of improvement works:

- A new function named MRAM Long Read Function that is capable of reading 480.000 byte signal part (2-second sensor data before the error occurs) with one run was created,
- A new function named EEPROM Long Write Function that is capable of writing 960.000 bytes (The Error Message) at one time was created.

Thus, one more essential optimization has been made in the software. In the first test with the finalized software, it has been seen that recording an error message which is

960.000 bytes in terms of data size lasts 11.18 seconds. The result of the relevant test is given in Figure 3.78 as an example.

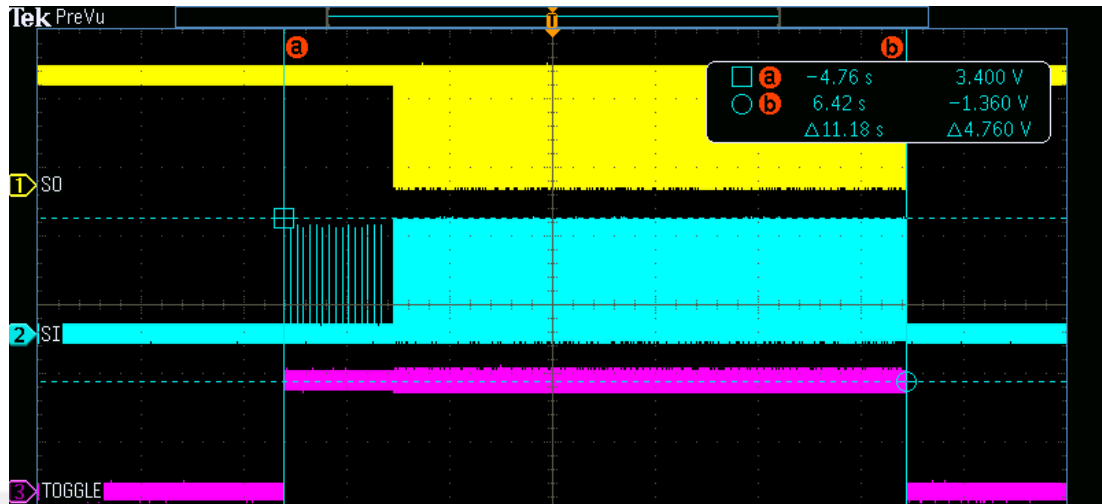


Figure 3.78 : The error recording period after optimizations.

The FDS performance was retested with the finalized software many times, thus and so the new recording time of an error message was re-measured again and again. In conclusion, it has been observed that transfer of an error message which is the time interval new sensor measurements cannot be recorded, lasts 11 seconds on average.



4. AN FPGA-BASED PROPOSAL FOR THE FDS DESIGN

A microprocessor based FDS was designed step by step according to the tasks to be executed within the scope of thesis. In addition to the verification tests made at all steps, final tests were also performed by modeling the system behaviour in case of an error for testing the entire system accuracy and performance.

The last study is an FPGA-based improvement proposal for the time period that a new error cannot be recorded. However, it should be emphasized that an FPGA-based system is only proposed within the scope of this chapter. That is to say, the related system has been designed as high-level, developed by using VHDL and tested in the VIVADO Design Suite environment. The Vivado simulator is a Hardware Description Language (HDL) simulator that provides to perform behavioral, functional, and timing simulations for VHDL, Verilog, and mixed-language designs [19]. In other words, all VHDL designs made for the relevant proposed system have been validated with the dummy slaves and data traffic generated in the simulation environment. No hardware design or tests on physical setup (equipment) have been made, differently from the system developed based on microprocessor.

A new system model has been created by considering the advantages of using FPGA for embedded applications such as parallel processing, speed, flexibility, extendibility. Therefore, a new high-level design was built starting from zero.

4.1 SPI Word Transfer Function

The FDS developed as microprocessor-based is basically a master recording unit that registers the read sensor data, managing the error scenarios, communicating with different type memory units it contains and managing the traffic between them.

When considered from the point of technology it uses, it is seen that basically an SPI-based communication management is carried out by the diagnostic system. In other words, the system is an error recorder performing several tasks such as transferring various data, sending commands and enabling (or disabling) directives via SPI

protocol by interfacing with a number of units. Therefore, the first step in the proposal of the new system design to be created as FPGA-based is the design of an SPI Master Module that provides communication suitable for 16-bits word transmission.

While designing the system, the classical SPI protocol was examined and a generic system was created as to be the basis for the diagnostic system to be designed. The block diagram of the system is given in Figure 4.1.

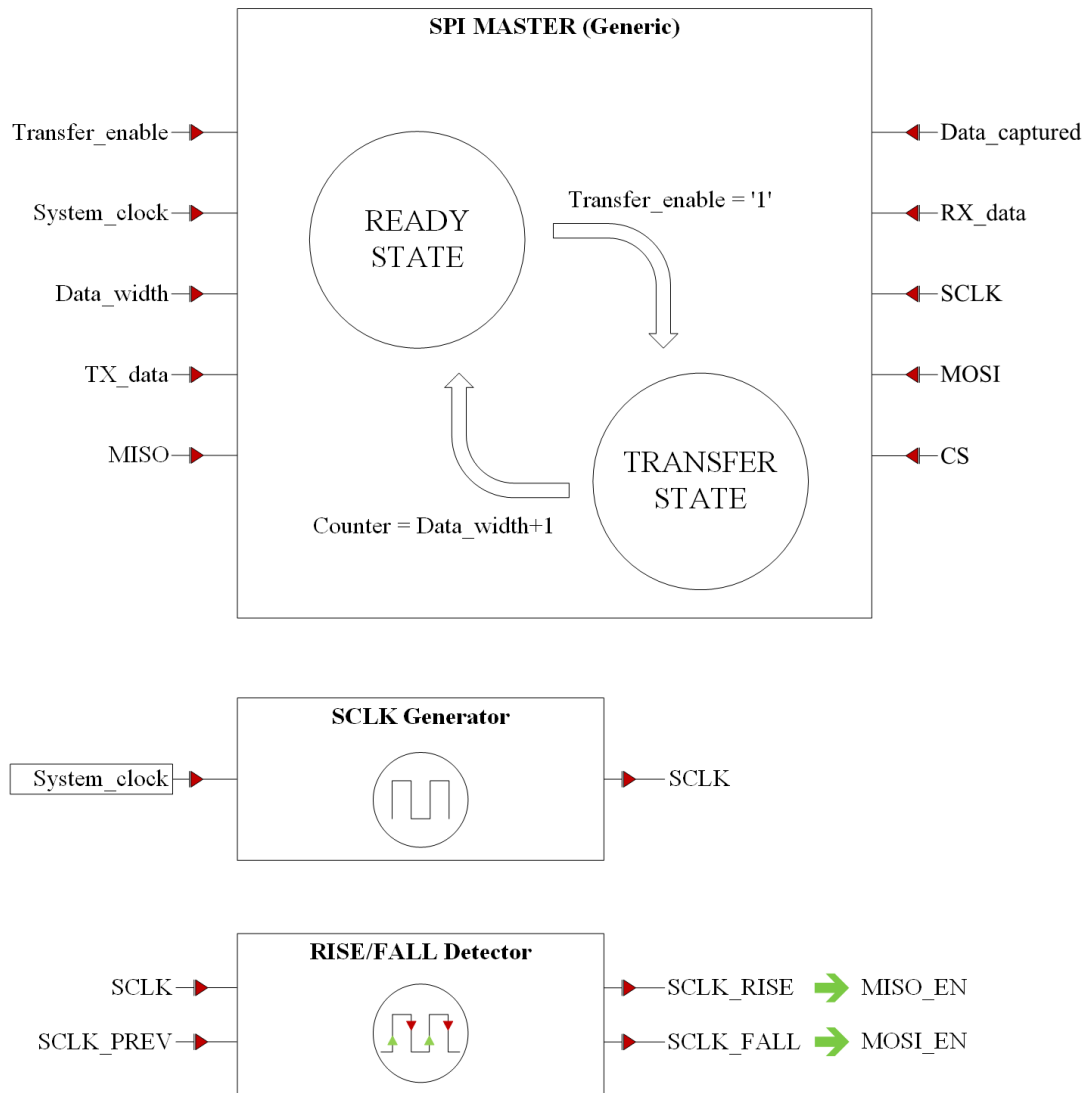


Figure 4.1 : Block diagram of SPI word transfer function.

The software basically consists of 3 processes, 1 main and 2 utility processes.

In the first utility process, the SPI clock pulse to be used in SPI communication is generated by using the system clock received from main system. As shown in the block

diagram in Figure 4.1, the only input required for operation is system clock, SCLK is generated by using this external clock signal and internal counters.

The timing of the reading and writing a bit is an important step to prevent incorrect operations or data losses. The second auxiliary process created for this purpose is used in detecting the rise and fall of the SPI clock signal to ensure synchronization. Thus, timely read/write operations on MISO and MOSI buses are performed. Because the slave devices used in the FDS support SPI Mode:0, also the system has been designed in accordance with SPI Mode:0. (CPOL:0, CPHA:0)

In the main process, the SPI protocol is realized, which allows communication with 16-bit words. Unless the transfer is enabled, the master waits ready for transmission. In ready state, no data is sent or read. All slave chips are disabled and all counters are reset. When the Transfer_enable input enabled, the transition from Ready State to Transfer State occurs.

In Transfer State, data transmission is started by enabling the relevant slave. During each SPI clock cycle, a full-duplex data transmission occurs. The master sends a bit at the falling edge of the SPI clock into MOSI line, while reading one bit at the rising edge on the MISO. The data is shifted out via registers with the most-significant bit first. Process continues for 17 number of clock cycles. When the each word transmission on the MISO and MOSI buses is completed, the master generates a data captured pulse, stops toggling the clock signal, and deselects the slave. The base algorithm is demonstrated by using a flowchart and given in Figure 4.2 and 4.3.

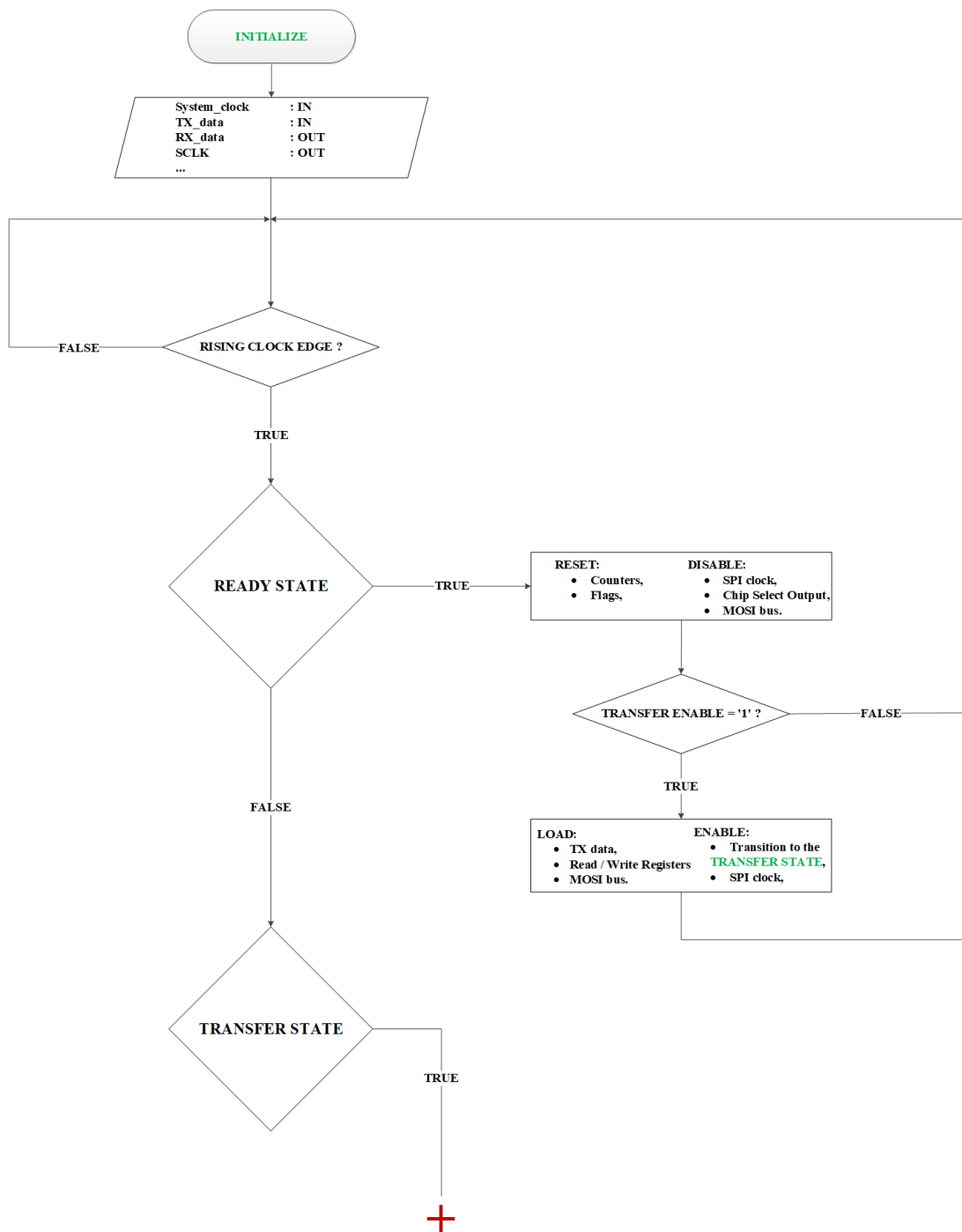


Figure 4.2 : SPI word transfer function flowchart (part-1).

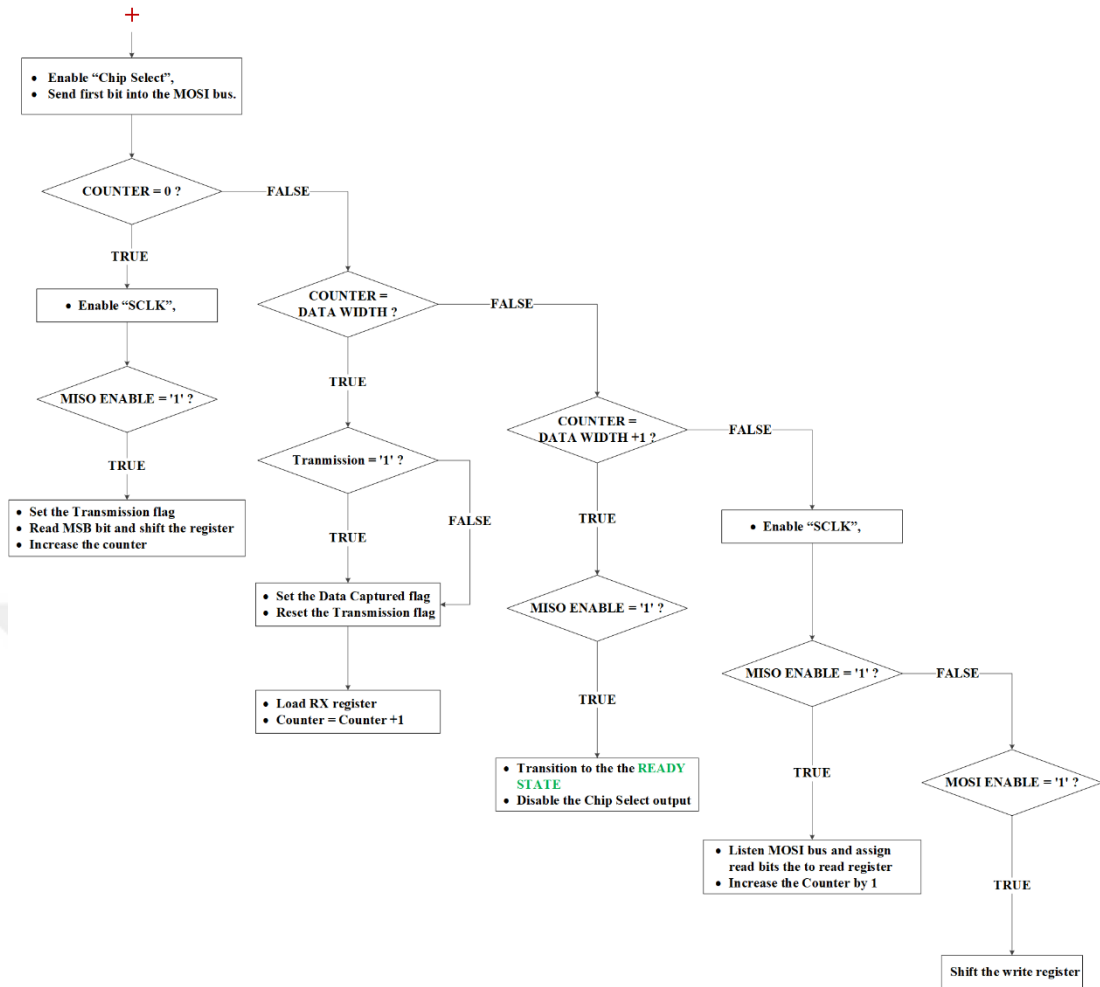


Figure 4.3 : SPI word transfer function flowchart (part-2).

If the Transfer_enable is still activated, the transition from Ready State to Transfer State occurs again and processes are repeated. As stated earlier, all steps are generically created with reference to the final diagnostic system design. Therefore, the logic of the relevant algorithm is also built to allow to continue the communication as 16-bit words, unless the Transfer_enable bit is deactivated.

4.2 Fault Diagnostic System Design

In the previous section, the word transfer function, which will be the basis of the FDS, has been designed. It is an SPI Master Module that provides 16-bit uninterrupted word transmission with SPI protocol as long as the relevant system is triggered appropriately. In this part, the related function was upgraded and an FPGA-based FDS was designed as an alternative solution to the microprocessor-based design. It is aimed

that the FPGA-based system is able to perform all the operations performed by the microprocessor-based system and successfully execute the same error recording scenario. Differently from microprocessor-based design processes, the design was carried out by using parallel logic and digital design approach in order to profit by the advantages of FPGA in maximum, while designing this system.

The main software can basically be thought of as a state machine (or Turing machine) consisting of 4 states, a certain number of transitions and actions that can be modeled with flow graphs and the path of the logic can be determined automatically according to the conditions. Since the FPGA-based system has been designed according to the same error logging scenario, the operating logic of the model is similar to the microprocessor-based software. The block diagram of the system is given in Figure 4.4.

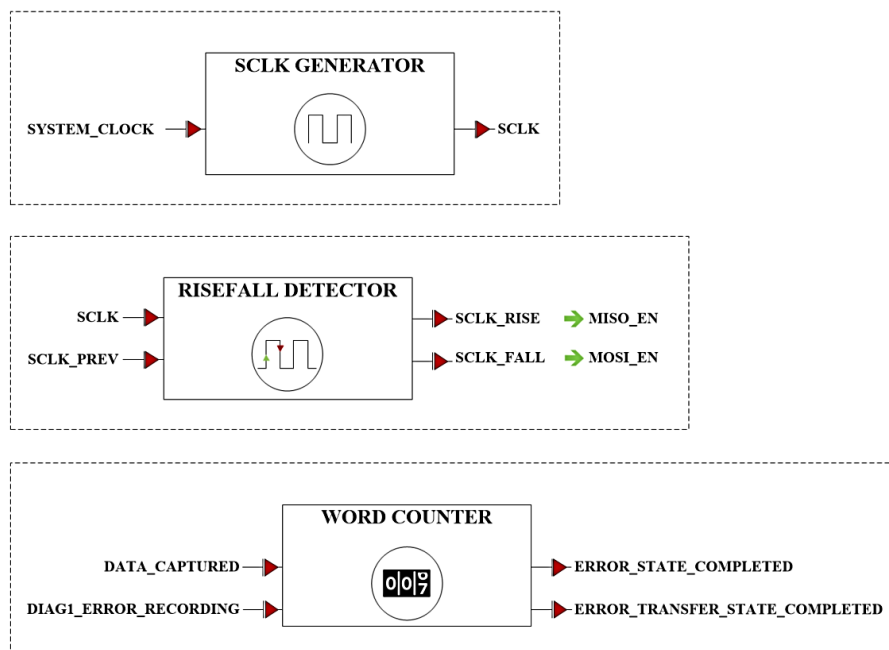
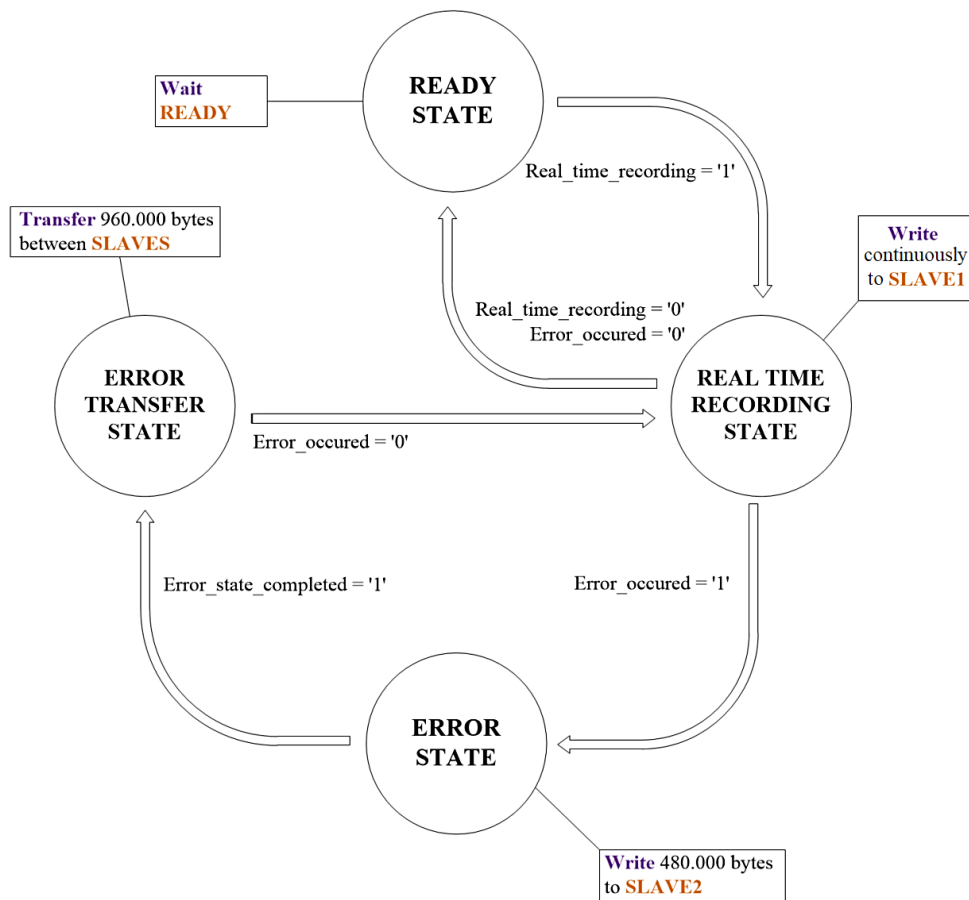


Figure 4.4 : Block diagram of the diagnostic system (FPGA).

The system starts from the Ready State. In the Ready State, all flags, counters and buffers of the system are initialized and the system is made ready for recording. In addition, the SPI clock is not produced and the slave units are disabled in order to prevent any mistaken write/read operation. System waits at the ready (or on standby) for recording unless Real Time Recording input is activated.

With the request to start the real-time recording process of the main system which the FDS will be integrated into, the system switches to the Real Time Recording state. As explained in detail before, the data transmitted from 24 sensors are recorded in the MRAM devices in the relevant case. The SPI protocol will be used for the transmission of data to be recorded, and the relevant transmission will continue as long as the real-time error record request continues or no error occurs. While the system switches back to ready state in case of withdrawal of the error registration order, it transitions to the Error State with logical 'HIGH' value of the error flag.

The logic of the communication software in the error state is technically the same as in the real time recording state. The biggest behavioural difference with the real time recording state, always a data which has a certain length is recorded in the error state. While the communication in the real time recording state will continue unless a request from the main system or an error occurred, always a 2-second message is recorded in the error state. In the error state, the system automatically transitions to the error transfer state after all operations are executed properly. Error state is a state in which all the operations to be made are followed by counters, if the related operations are completed, the forward transition is made, and no state transition can be made with any outside intervention.

The last state in the software is the error transfer state in which the 4-second error message which total size of 960.000 bytes sampled at 5 kHz frequency is transferred from MRAMs to the corresponding EEPROM. Actually, the flow of the software is identical with in the Error State. Only the counter used for calculating the number of successful word transmissions was extended as able to count 960.000 bytes transfer and chip select command was changed to enable the EEPROM, not MRAMs.

Lastly, the subprograms (or process blocks) that form the system are described with their functions below:

RISE / FALL DETECTOR: A process that samples a signal and produces a stimulus when an edge occurs in the related signal. It is used in the FDS for detecting the rise and fall time of the SPI clock. Thus and so, time the word shall be written or read is determined precisely and accurately according to the selected SPI mode. (MISO, MOSI enable times)

WORD COUNTER: A process that calculates the number that how many times a successful word transmission occurred and produces a stimulus when required. There are two different word counters serving at similar purposes are used in the FDS. They basically follow and count every successful word transfer and trigger the transitions according to the software scenario. (Triggering the start and end time of the states).

CLOCK GENERATOR: A process that provides clocks according to the requirements. It is used in the diagnostic system to synchronize the shifting and sampling of the data. It is a crucial task to be performed since the data transmitted between the master and the slave is synchronized to the clock generated by the master (FDS).

4.3 Multitasking Fault Diagnostic System Design

This section is the last step of the FPGA-based software development process and also the main software is developed within the scope of this section. First of all, the reason of this Multitasking Fault Diagnostic System (MFDS) proposal explained below:

As detailed before, by command of start the real-time recording process of the main system, FDS starts operating. The data of 24 sensors are recorded in the MRAM device, and this recording continues until the real-time error record request is recalled or an error occurs. In the case of an error, these data received from 24 sensors, recorded for 4 seconds and transferred to the EPPROMs which are the permanent recording units of the FDS. However, this error transfer process takes approximately 11 seconds and new errors cannot be recorded until the process is completed.

Up to this section, the sub-modules of the MFDS have been designed and verified. These sub-systems were already designed of set purpose of the MFDS. Nevertheless, management and interconnections of them are also another important steps. In this section, it is aimed to design the MFDS by designing a management module and integrating the subsystems designed in previous sections together. Design details and the tasks of the subsystems in the system are explained in following sections.

4.3.1 Error recording decision

MFDS is an error logger structure consisting of 4 redundant diagnostic systems operating in parallel briefly. However, communication and synchronization between these parallel systems are other important issues besides the systems being accurate and working in accordance with the error recording scenario. Interferences between different systems and tasks can be a severe problem for operating simultaneously. The effect of such interface problems will vary according to the configuration, from slight disturbances to complete malfunctioning. In this sense, a new unit responsible for management and synchronization has become a crucial need and a new module has been designed in order to prevent incorrect operations and/or data losses etc. by controlling the timing and the triggering of each systems' status. The tasks to be performed by Error Recorder Decider are outlined below:

- Performing role assignments at the initialization of the MFDS.
- Monitoring current status and following operations for all FDSs.
- Determining the current and next FDS according to the relevant situation.
- Generating and issuing the enable/disable commands for all FDSs according to the possible scenarios. (Only one system can be responsible and enabled for real time recording.)
- Determining the number of diagnostic systems to be run at the same time based on the Real Time Record request and Error Occurred indications received from the host system.

In the light of information given above, Error Recording Decider module has been designed in VHDL programming language. The base algorithm is demonstrated by using the flowchart given in Figure 4.5.

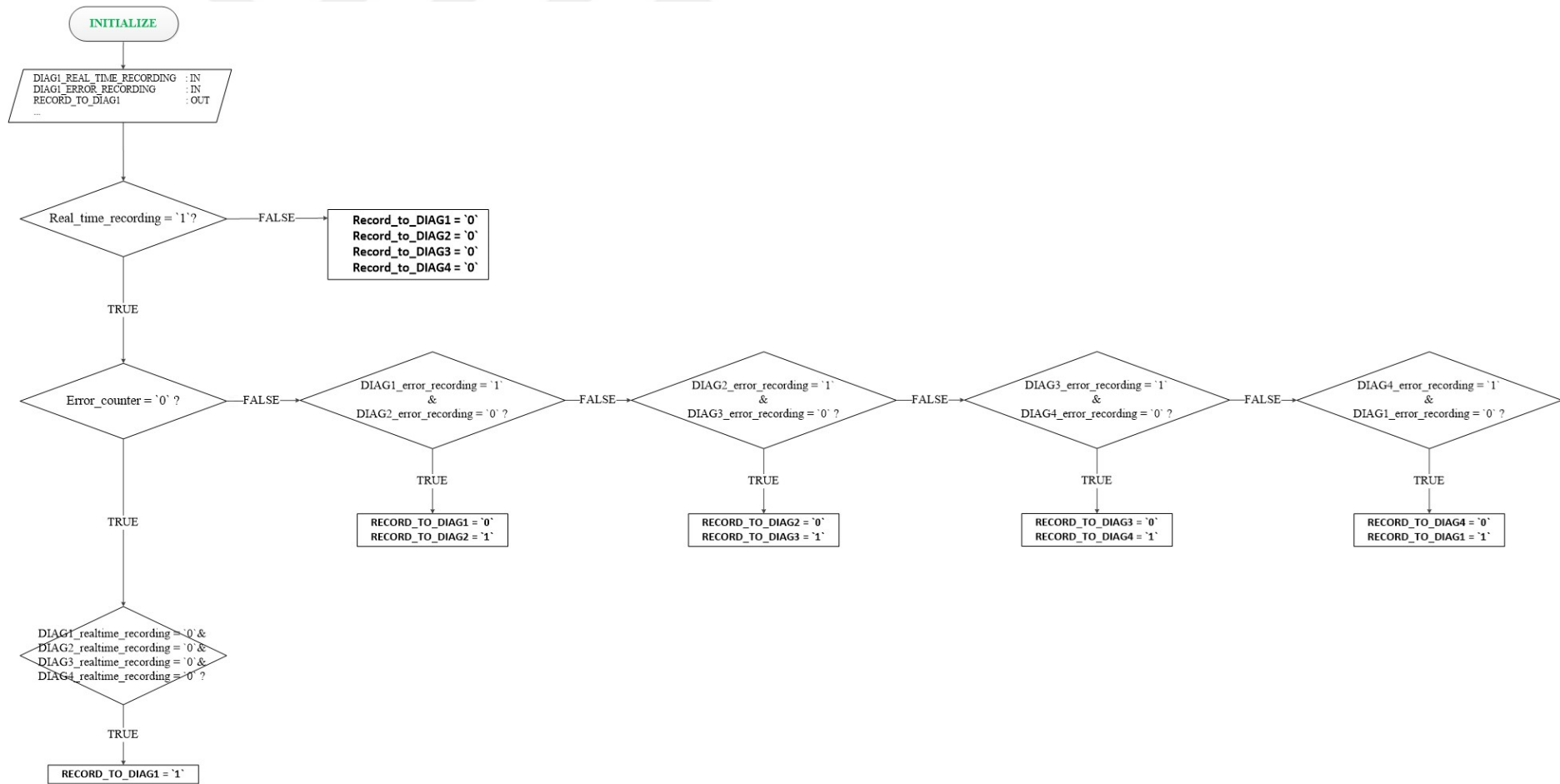


Figure 4.5 : Error recording decision flowchart.

4.3.2 Final system architecture & design specification

As detailed before, the base of this proposal is the inherent parallelism which is one of the most important features of using FPGAs for a design. In the final part of the proposal, a new fault diagnostic system has been designed by using that intrinsic parallel nature of FPGA architecture.

The system works in accordance with the error recording scenario specified and described in detail before, and carries out all the tasks given below performed by the system designed using microprocessor.

- Read data from the sensors continuously and save to real time recording structure.
- Check real time error recording command and error status continuously.
- In case of an error, record the sensors' data 2-second more.
- Generate the Error Message in real time recording structures by using 2-second data before and 2-second data after the moment of error occurrence.
- Transfer the Error Message from real time recording structures to the relevant permanent recording structure after the 4-second error recording completed.

In addition to these tasks, the related system has been designed with the principle of parallel operation in such a way that it will continue to record new errors that may occur in case of an error transmission. The new system named MFDS, basically forms from FDS more than one for uninterrupted logging, an Error Recording Decider for managing the active/standby statuses and auxiliary subprograms such as counters, detectors, timers etc.

According to the tests performed on the actual hardware and software of the microprocessor-based system and the theoretical calculations, the number of FDSs should be used in the finalized system has been determined as four in order for MFDS to provide uninterruptedly recording and fully redundancy. In the light of information given above, the MFDS was coded in VHDL programming language and the block diagram of the system is demonstrated in Figure 4.6.

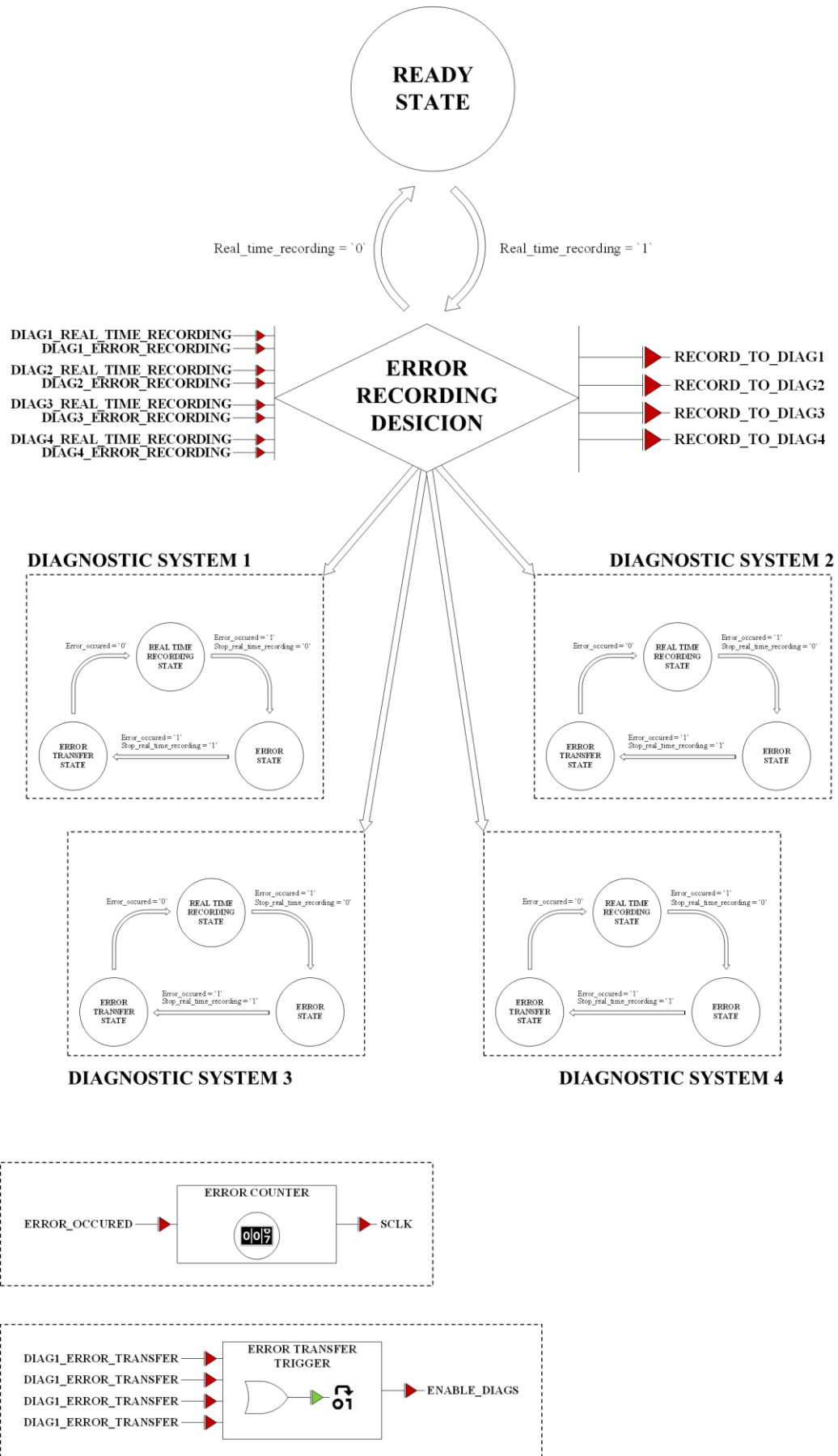


Figure 4.6 : Block diagram of the multitasking diagnostic system.



5. TESTS FOR THE FPGA-BASED SYSTEM

In the third chapter, accuracy and performance tests for microprocessor-based FDS design were performed. The related tests were conducted by using the ICU hardware that one of its PCBs was designed in scope of the thesis. However, FPGA-based development is just a behavioural proposal for future, hardware processes are out of the thesis' scope for FPGA, as emphasized before. So, in contrast to the system development stages based on microprocessor, no tests on physical setup (unit or equipment) are performed in FPGA-based design stages. For the very reason, modeling the behaviour of the external units that FDS will operate with or integrated into is another important issue needed for the tests of FPGA-based system.

While creating the test scenario and modules, the scenario followed in the microprocessor-based design processes is taken as reference, and the simulation model are set. The external process blocks (or modules) that designed for related tests are given with their explanations in corresponding sections below.

5.1 Tests for SPI Word Transfer Function

In this section, the SPI Word Transfer Function which creation steps are explained in section 4.1, is tested and results are given. Within the scope of the tests, 5 data in 16-bit word size were randomly selected for the transmission from the master and slave units. While the data to be transmitted by the SPI Word Transfer function, which is designed as the Master unit, is registered into the TX data input buffer, a new test software has been created to simulate the behaviour of the Slave unit. The related test software is basically a slave code that triggered by the chip select input and transmits the words selected for the test through MISO sequentially, in case that the packet transmission and reception is completed successfully. Block diagram of the test structure and the data expected to be transmitted by the master and slave peers according to the test flow are given in Figure 5.1 and Table 5.1 respectively.

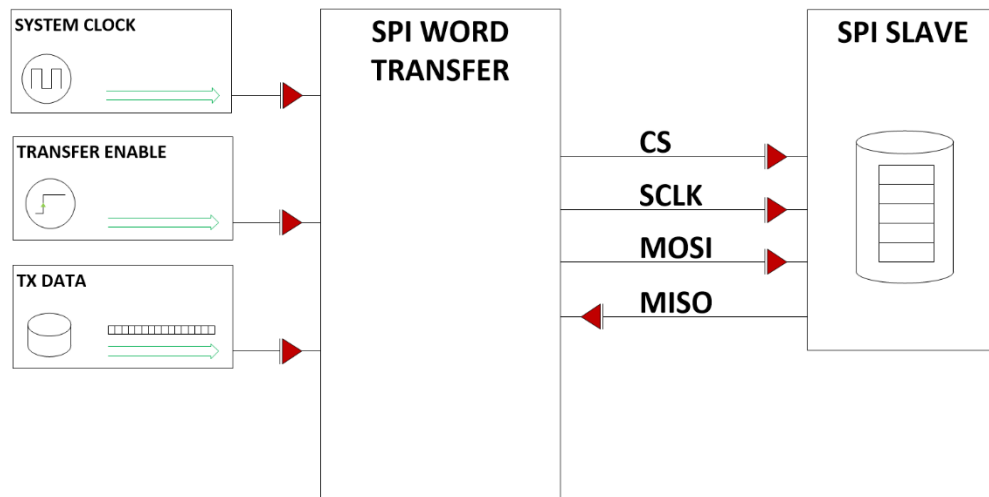


Figure 5.1 : Block diagram of the test structure.

Table 5.1 : Test details.

Test Step	Master Data <i>Hexadecimal</i>	Master Data <i>Binary</i>	Slave Data <i>Hexadecimal</i>	Slave Data <i>Binary</i>
Transmission1	A7B2	1010011110110010	0000	0000000000000000
Transmission2	B813	1011100000010011	A7B2	1010011110110010
Transmission3	3D76	0011110101110110	B813	1011100000010011
Transmission4	D481	1101010010000001	3D76	0011110101110110
Transmission5	63F0	0110001111110000	D481	1101010010000001

Within the scope of the test, MISO and MOSI buses were monitored and verified through behavioural simulation on the VIVADO Design Suite environment, and the accuracy of the communication managed by the master was checked. In addition, the communication was analyzed in detail by probing the relevant results in terms of parameters such as the timing, clock and enable values etc. The results with their explanations are given below:

As mentioned above, it has been aimed to send 5 data in word format from master and slave units. Some of the expected indications during the transmission of this data are given below:

- Transition from ready state to transfer state with the activation of the transfer flag before sending the first data.
- Logic LOW chip select value during the transmission and transition to the HIGH level when the transmission is completed.
- Triggering the Data_capture flag after every successful word transfer.

- Bit transactions on MISO and MOSI buses.
- Significant words in the Transmit and Receive Registers.
- No clock pulse on the SPI clock line, except for data transmission.

Figure 5.2 : General view of the test.

Before the transmission (before enabling the transfer):

Figure 5.3 : Period before the transmission.

After this stage, the results of the five transfers carried out are given below, respectively. As highlighted in earlier chapters, the system has been designed to operate in CPOL Mode:0. For this reason, rising edges of the SCLK signal were indicated using blue cursors so that the read/write operations could be displayed more

clearly in Figures. Thus, all bits of the data which accuracy were checked digitally during the tests were marked also in the results for the sake of clarity.



Figure 5.4 : First word transfer.

It is seen that as a result of the Transfer_enable input taking the logic HIGH value, the relevant slave device was enabled and the transition to the transfer state was performed. When the related test result in Figure 5.4 is analyzed from the point of accuracy, it is seen that the data sent to the MOSI bus is 1010011110110010 and it is equal to A7B2 in hexadecimal. The data read on the MISO bus is 16-bit zero as expected.

When the end of first word transmission is investigated, it is seen that Data_captured signal was triggered properly after word transfers, because the transmission were performed successfully. Another important issue needs attention is that Transfer_enable input was checked by software, and the communication continued in Transfer State for new words since the related input has still logic HIGH value. Accordingly, there has been no change in either Chip Select or SPI Clock.

Explanations for the other transmissions have not been given in order to avoid repetition, only the results are given below respectively:

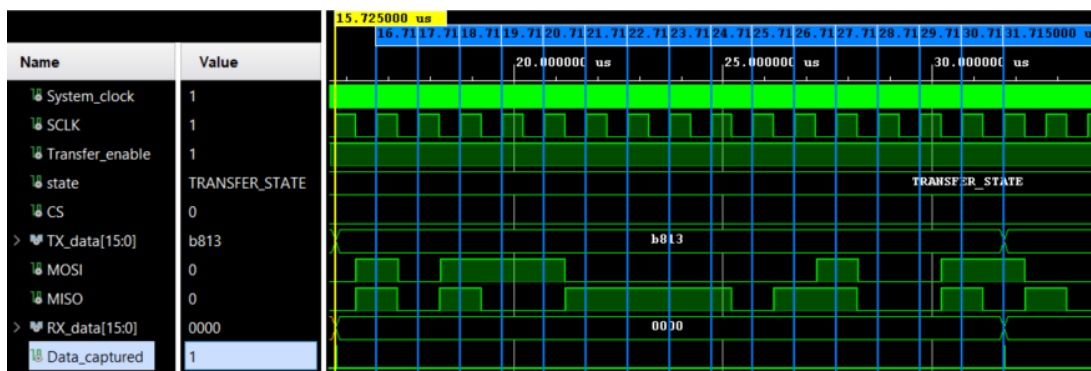


Figure 5.5 : Second word transfer.

The data sent to the MOSI bus are 1011100000010011 in order, which is equal to B813 in hexadecimal, while the bits on MISO bus are 1010011110110010 which is equal to A7B2 in hexadecimal (Figure 5.5).

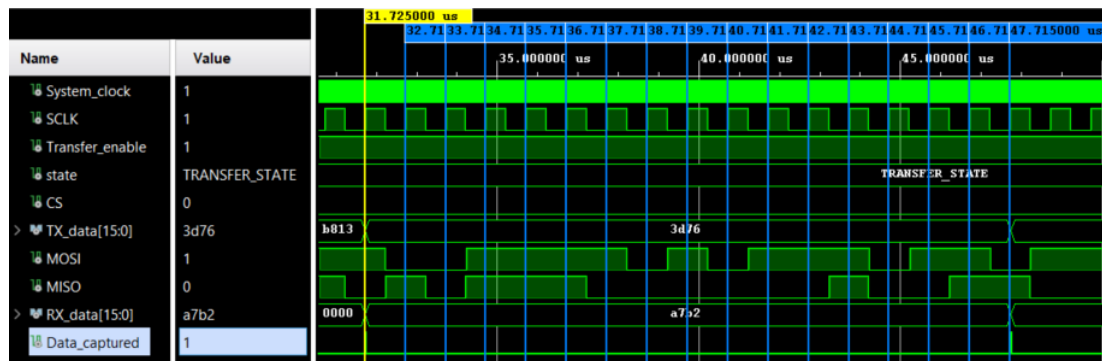


Figure 5.6 : Third word transfer.

As seen in Figure 5.6, the data sent to the MOSI bus are 0011110101110110 which is equal to 3D76 in hexadecimal, and the bits on MISO bus are 1011100000010011 equal to B813 in hexadecimal.

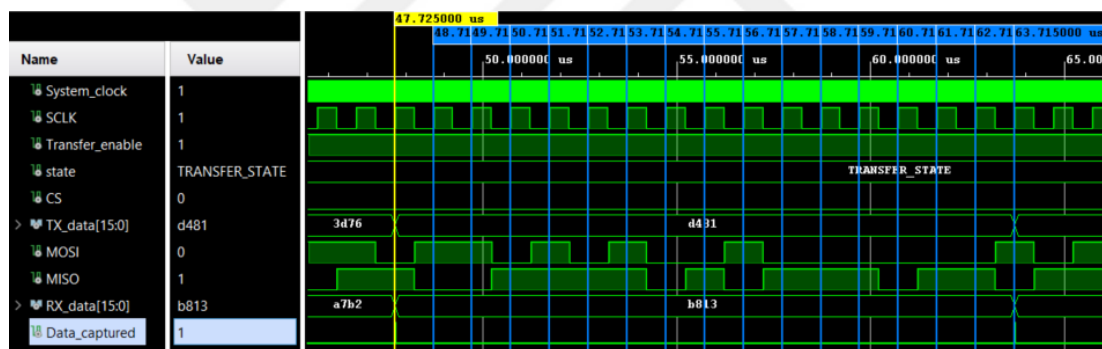


Figure 5.7 : Fourth word transfer.

Data sent to the MOSI bus are 1101010010000001 in order, which is equal to D481₁₆, while the bits on MISO bus are 0011110101110110 equal to 3D76₁₆ (Figure 5.7)

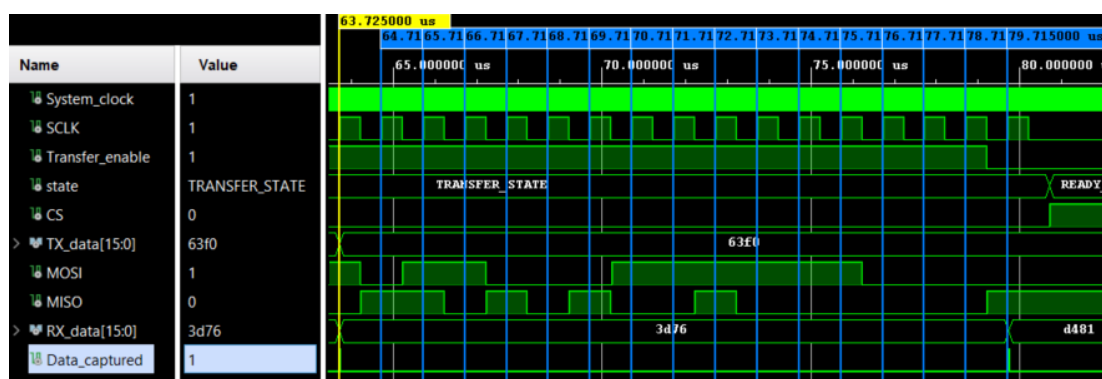


Figure 5.8 : Last word transfer and end of the program.

In the last transfer operation, it is seen that the data on the MOSI line is 0110001111110000, which is equal to 63F0 in hexadecimal number system. The data read on the MISO bus is 1101010010000001 which is equal to D481 in hexadecimal (Figure 5.8).

Furthermore, the write and read bits were assigned to the corresponding buffers and viewed on with Vivado Design Suite's Behavioural Simulator as mentioned before. The data read from related buffers are given in Figure 5.9 as consisting of all transfer results.

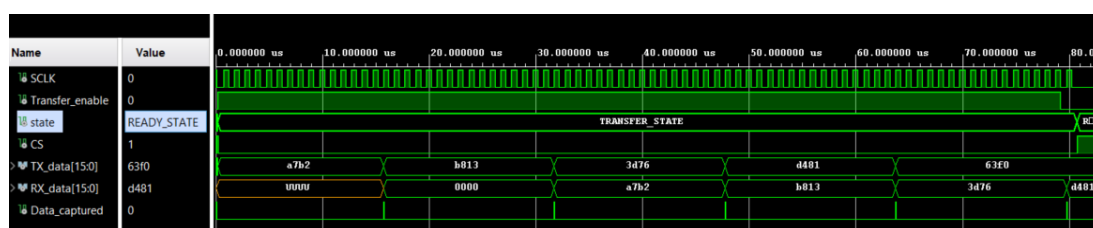


Figure 5.9 : Data of transmission and receive buffers.

Watching both the MOSI/MISO buses and the return values of buffers, the results were double checked and it has been clearly seen that all data is accurate. Thus and so, the accuracy which is one of the most important parameters of the test has been verified.

Another important issue needs attention is that the Data_captured signal was triggered after all successful word transfers. In addition, it is seen that the relevant slave device was disabled and the transition to the ready state was occurred as a result of the Transfer_enable input taking the logic LOW value.

Within the context of these tests, the operating mechanism and accuracy of the SPI Word Transfer Function were tested by generating a dummy traffic consisting of 5 packet transmission, and it has been concluded that all processes have run successfully as planned.

5.2 Final Error Scenario Tests for FDS

In this section, accuracy and performance of the FDS were tested and the sanity check for the system performed through behavioural simulation on the VIVADO Design Suite environment. Within the scope of the test, a dummy sensor traffic was generated and the same scenario was created by taking the scenario followed in the microprocessor-based design processes as reference. The first 480.000 byte signal part

(2-second sensor data before the error occurs) was recorded, the error flag was triggered. Thus and so, the sensor recording process, which would continue for 2-seconds more after the error, was launched. After these steps, the error transfer state was performed and the whole error message consists of 960.000 bytes were transferred.

As a first step of the test, the external module needs were determined according to the test scenario and designed. In the second step, test were performed and results were analysed in detail.

5.2.1 The test environment

In the third chapter, accuracy and performance tests of microprocessor-based FDS design were performed. The related tests were conducted by using the ICU hardware that one of its PCBs was designed. However, FPGA-based development is just a behavioural proposal for future, hardware processes are out of the thesis' scope for FPGA, as emphasized before. So, in contrast to the system development stages based on microprocessor, no tests on physical setup (unit or equipment) have been performed in FPGA-based design stages. And therefore, modelling the behaviour of the external units that diagnostic system will operate with or integrated into is another important issue needed for tests.

While creating the test scenario and modules, the scenario followed in the microprocessor-based design processes was taken as reference, and the simulation model were set. The external process blocks (or modules) that designed for tests are given with their explanations below:

SLAVES SIMULATORS: A process designed to model the behaviour of MRAM and EEPROM slave devices used in the thesis. It transmits the data coming from the data generator module to the MISO line in bits in accordance with the clock phase and polarity. Unless the module is enabled and / or receives a SPI clock, it does not send data.

DATA GENERATOR: A process that generates data according to some pattern instead of reading data from file, database, or any other data source. It is used in the FDS to simulate the sensor and slave data in real application. There are three different data generators serving at similar purposes are used in FDS. The first and second ones generate 240.000 words each, which are used to generate the data to be transmitted in

Real Time Recording and Error states, while the last one produces 480.000 words that is equal to the size of an error message (960.000 bytes).

MAIN UNIT SIMULATOR: A process designed to model the behaviour of the Main Unit which the FDS will be integrated into. It is responsible for producing and sending the system clock signal. In addition, it sends the commands and statuses such as real time record and error status to simulate an error case for test.

The block diagram for the final error test scenario is shown in Figure 5.10.

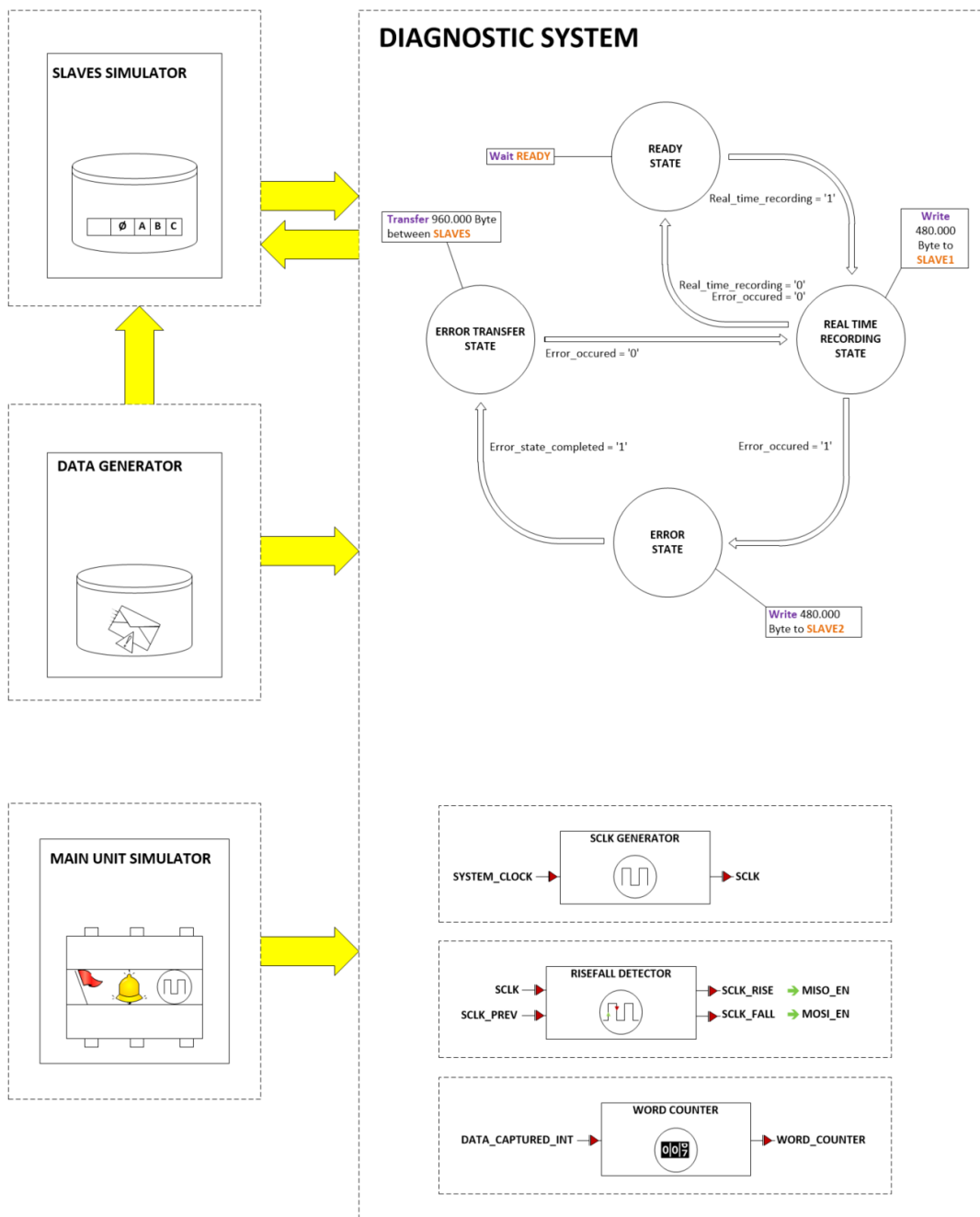


Figure 5.10 : Block diagram of the final test environment.

5.2.2 The test results

The error scenario detailed in microprocessor-based design sections was built for FPGA-based design and run repeatedly. The system flow was examined in behavioural simulation on the VIVADO Design Suite environment. The related scenario is given shortly below:

- The diagnostic system was activated through the Main Unit Simulator.
- A dummy sensor traffic was generated by using the Data Generator Unit.
- When the first 480.000 byte signal part (2-second sensor data before the error occurs) was recorded, the error flag was triggered by Main Unit.
- Another dummy sensor traffic (2-second data after the error) was produced by using the Data Generator Unit again.

The starting and ending points for the test scenario are determined in software. The tests that start with initialization of the system, terminated automatically after the failure scenario is completed. The period between the starting and ending points is given in Figure 5.11.



Figure 5.11 : General view of the test result (diagnostic system).

As shown in the general view given in Figure 5.11, all of the buses have data in the transmission states as expected, the flag statuses, selection of the slave units and the transitions between states are compatible with the scenario.

However, since the related process includes transfers of big data and takes a large time interval, it shall not be reliable to perform detailed checks on the general view. For this reason, the relevant transfer process has been sampled and examined in detail for each state by analysing the simulations results extensively, starting from the initial moment, and the determinations and findings are given below.

Before the transmission (before enabling the Real Time Recording by Main Unit):

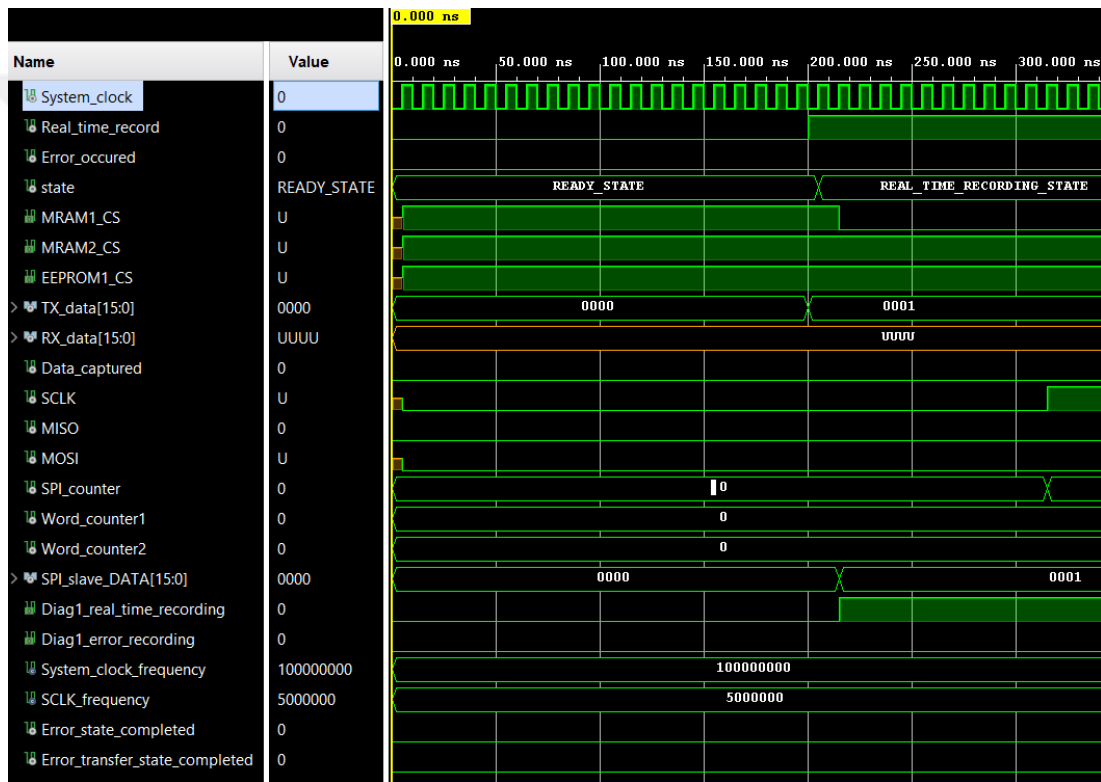


Figure 5.12 : Ready state.

As shown in Figure 5.12, the software began working in the Ready State since the Real Time Record command was not triggered. In the ready state, all buffers and counters had zero value (initial value) and there was no data transactions on MOSI/MISO buses.

In addition, it is clearly seen that the system kept the chip select output in the disabled position for all of the Slave Units and there was no clock pulse in the SPI clock output in order to prevent mistakenly write/read operations. Lastly, it is observed that the system sent status indications accurately.

Real Time Recording State:

For Real Time Recording State, waveforms of all inputs/outputs, expressions, counters and buffers etc. have been examined and detailed explanations are given below.

- With the Real Time Record order given by the Main System which the FDS is integrated, the system switched to RTC State.
- Transmission of the SPI clock was started with the transition to RTC state.
- The related slave unit MRAM1 was activated while MRAM2 and EEPROM devices were kept disabled.
- The dummy sensor data sent from the Main System was retrieved to the TX_data buffer and sent via MOSI bus properly.
- The data received from the MRAM1 was properly read in the MISO bus and registered into the RX_data buffer.
- It has been observed that all the data produced by the Data Generator, starting from 1 and increasing by one up to 240000, are received and sent correctly.

***NOTE:** The modulo operation ($\text{mod}65536$) should be performed to find the values which are greater than 65535 because the data size is 16-bit. For instance, the transfer in the Real Time Recording State ends with A980 which is equal to 43392 in decimal. 43392 is 240000 modulo 65536.*

- Data_captured signal was triggered after all successful word transfers.
- Word Counters were activated and every successful word transfer was followed and counted properly.
- System status indications were sent correctly throughout the state. (Diag1_real_time_recording = 1, Diag1_error_recording = 0)
- The transition to the Error State has been observed with the Error occurred input having logic value “HIGH”.

The waveforms of the beginning and end for the Real Time Recording State are given in Figures 5.13 and 5.14.



Figure 5.13 : Beginning of real time recording state.



Figure 5.14 : End of real time recording state.

Error State:

Waveforms of all inputs/outputs, expressions, counters and buffers etc. in Error State have been examined and detailed explanations are given below.

- With logical "HIGH" value of the Error_occurred flag, FDS switched to Error State correctly.
- The transmission of the SPI clock resumed uninterruptedly without any clock displacement or misclock.
- The related slave unit MRAM2 was activated while MRAM1 was deactivated again. EEPROM device were still disabled.
- The dummy sensor data sent from the Main System was retrieved to the TX_data buffer and sent via MOSI bus properly.
- The data received from the MRAM2 was properly read in the MISO bus and registered into the RX_data buffer.
- It has been observed that all the data produced by the Data Generator, starting from 240001 and increasing by one up to 480000, are received and sent correctly. (According to the modulo principle)

***NOTE:** The transfer in the Error State starts with A981 which is equal to 43393 in decimal ($240001 \bmod 65536$) and ends with 5300 which is equal to 21248 in decimal ($480000 \bmod 65536$).*

- Data_captured signal was triggered after all successful word transfers.
- Word Counter was reset on the transition to Error State. Each successful word transfer was followed and counted by starting at zero properly.
- System status indications were sent correctly throughout the state. (Diag1_real_time_recording = 1, Diag1_error_recording = 0)
- The transition to the Error Transfer State performed automatically and accurately by using the Word Counter.

The waveforms of the beginning and end for the Error State are given in Figures 5.15 and 5.16.

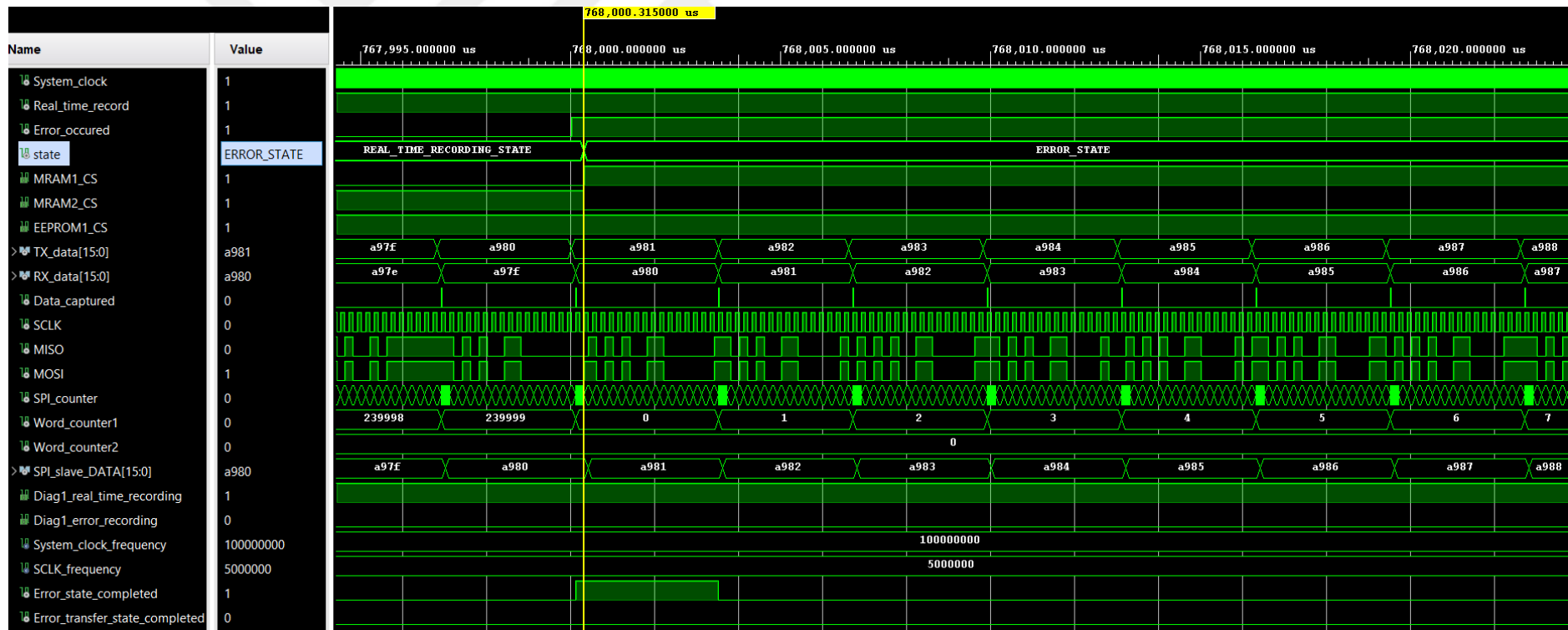


Figure 5.15 : Beginning of error state.

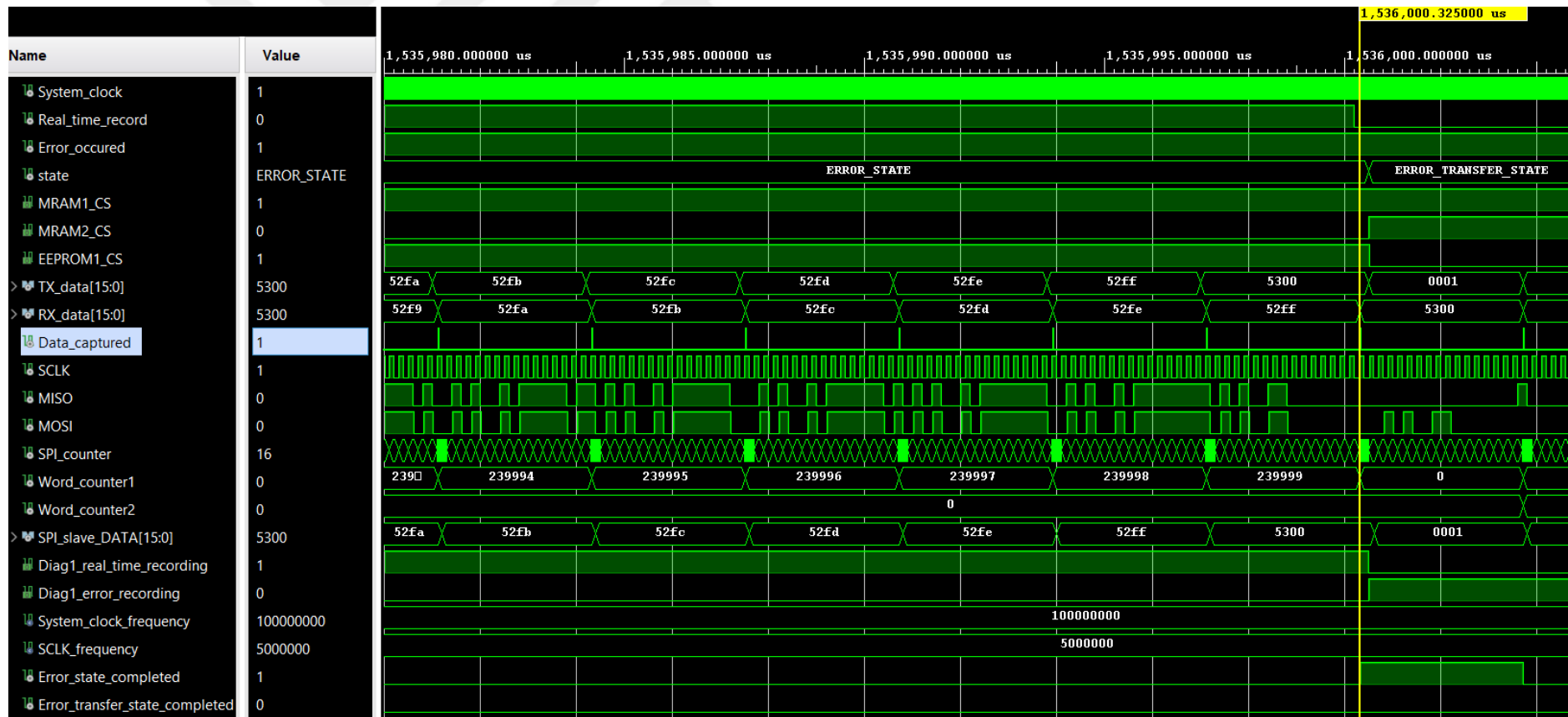


Figure 5.16 : End of error state.

Error Transfer State:

For Error Transfer State, waveforms of all inputs/outputs, expressions, counters and buffers etc. have been examined and detailed explanations are given below.

- With the data counter value reached 240.000 again, the system detected the 4-second error message written to the MRAMs and switched to RTC State.
- The transmission of the SPI clock resumed uninterruptedly without any clock displacement or misclock.
- The related slave unit EEPROM was activated while MRAMs was disabled again.
- The dummy error message was retrieved to the TX_data buffer and sent via MOSI bus properly.
- The data received from the EEPROM was properly read in the MISO bus and registered into the RX_data buffer.
- It has been observed that all the data produced by the Data Generator, starting from 0 and increasing by one up to 480000, are received and sent correctly. (According to the modulo principle)

NOTE: *The transfer in the Error Transfer State starts with 0 and ends with 5300 which is equal to 21248 in decimal (480000 modulo 65536).*

- Data_captured signal was triggered after all successful word transfers.
- Word Counter was reset on the transition to Error Transfer State. Each successful word transfer was followed and counted by starting at zero properly.
- System status indications were sent correctly throughout the state. (Diag1_real_time_recording = 0, Diag1_error_recording = 1)
- The transition to the Real Time Recording State performed automatically and accurately by using the 2nd Word Counter.

The waveforms of the beginning and end for the Error Transfer State are given in Figures 5.17 and 5.18.

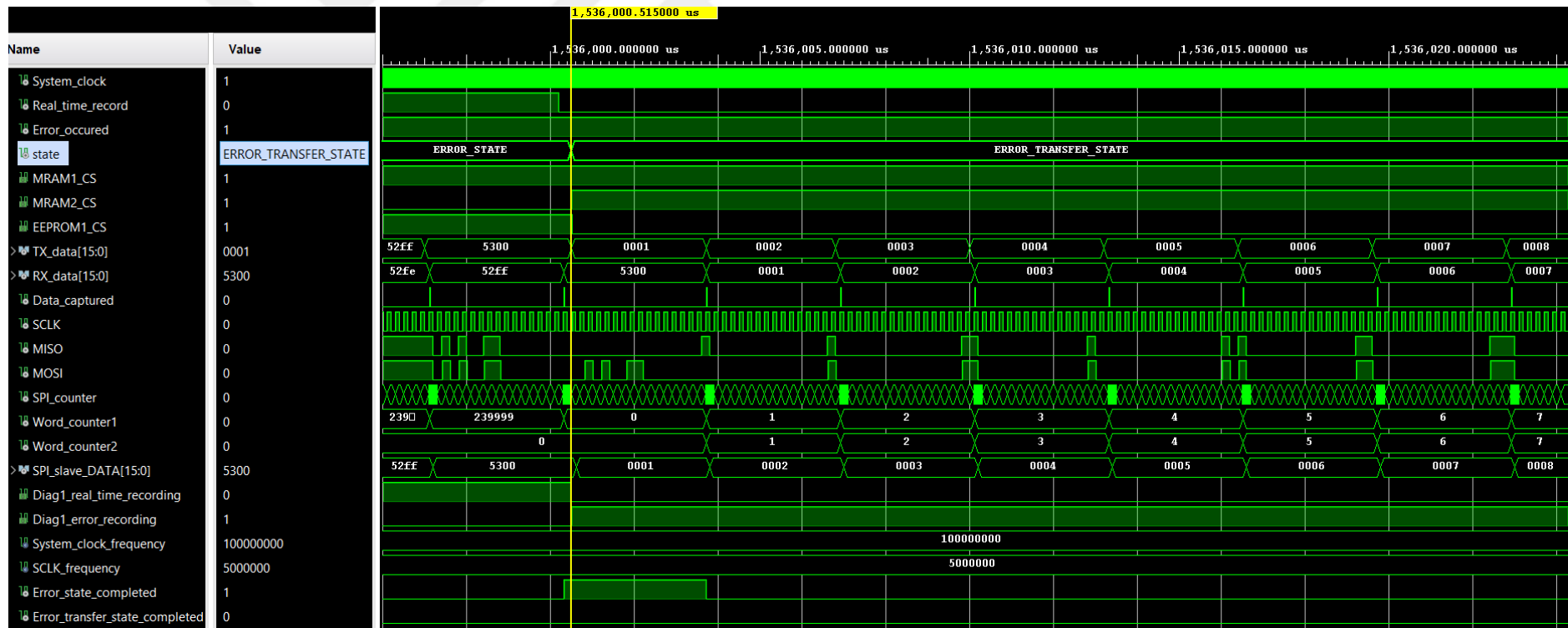


Figure 5.17 : Beginning of error transfer state.

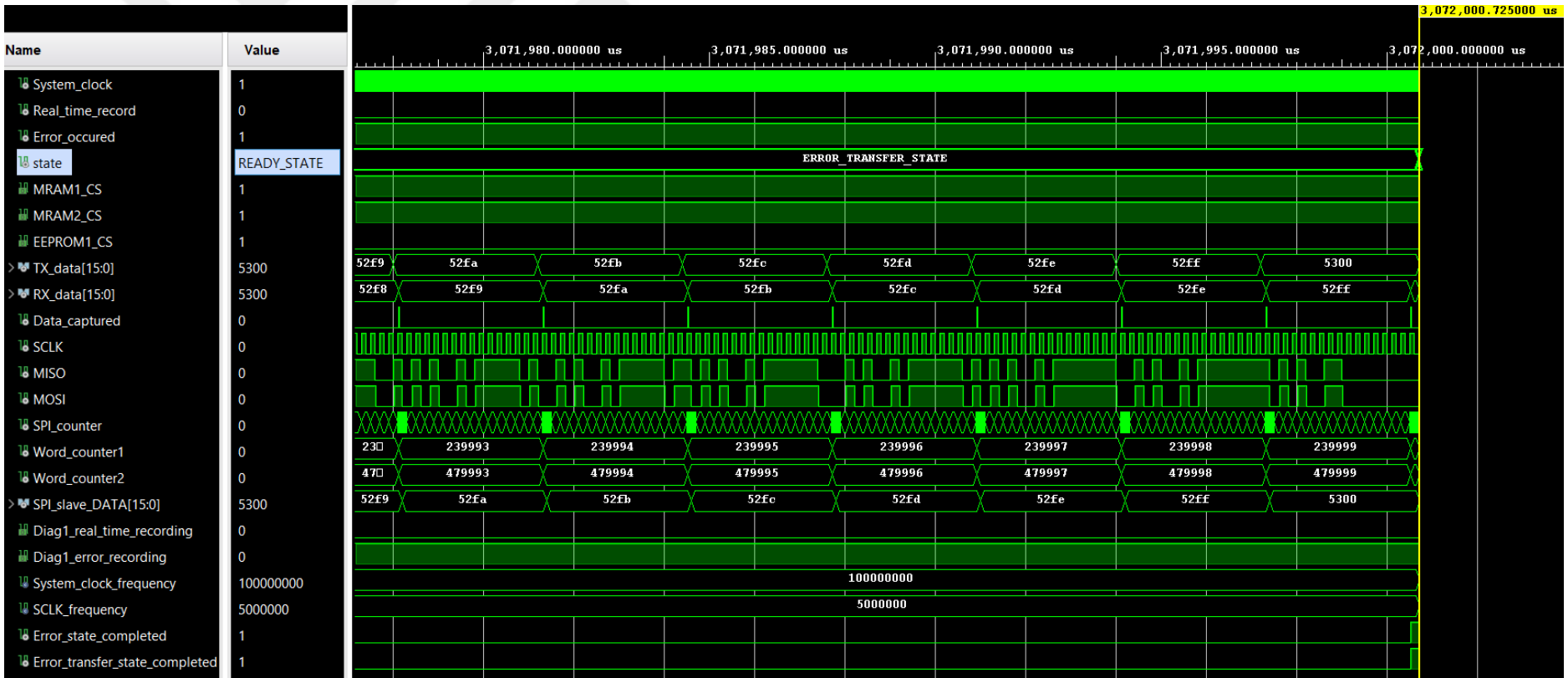


Figure 5.18 : End of error transfer state.

Within the scope of the tests, it has been observed that the main and utility (or auxiliary) softwares work in accordance with their purposes in the system designed as FPGA-based, and the FDS has operated in conformity with the error scenario according to the commands and indications sent by the Main System. In addition, it was observed that there has not been any corruption, delay in the data or any missed data during the communication managed by the system.

As a result, it has been observed that the structure designed as FPGA-based can successfully perform the recording operations made by the microprocessor-based FDS, and the error recording scenario has been run in the same manner.

5.3 The Final Error Scenario Tests for MFDS

In this section, the redundancy tests for the MFDS performed through behavioural simulation on the VIVADO Design Suite environment. A dummy sensor traffic was generated and the same scenario was created by taking the scenario followed in the microprocessor-based design processes as reference. The first 480.000-byte signal part (2-second sensor data before the error occurs) was recorded, the error flag was triggered. Thus and so, the sensor recording process, which would continue for 2-seconds more after the error, was launched. After these steps, the error transfer state was performed and the whole error message consists of 960.000 bytes were transferred. Differently from the other systems designed in scope of the thesis, the sensor traffic was continued and it was checked that whether the recording was still accomplished without any interruption.

As a first step of the test, the external module needs were determined according to the test scenario and designed. In the second step, tests were performed and results were analysed in detail.

5.3.1 The test results of error recording decision module

In this section, the test environment (or simulation model) were set and different test scenarios for Error Recording Decision Module which creation steps were described in section 4.3.1 were built and was run in behavioural simulation on the VIVADO Design Suite environment. Before building the test scenarios, the behaviour of the external units (or modules) that Error Recording Decider will operate with modeled, and were designed as it was in previous sections. While designing test modules, only

the functions which are needed in scope of the test were created such as sending indications, commanding, and the verification and performance of the Error Recording Decider were tested.

The block diagram for the final error test scenario is shown in Figure 5.19.

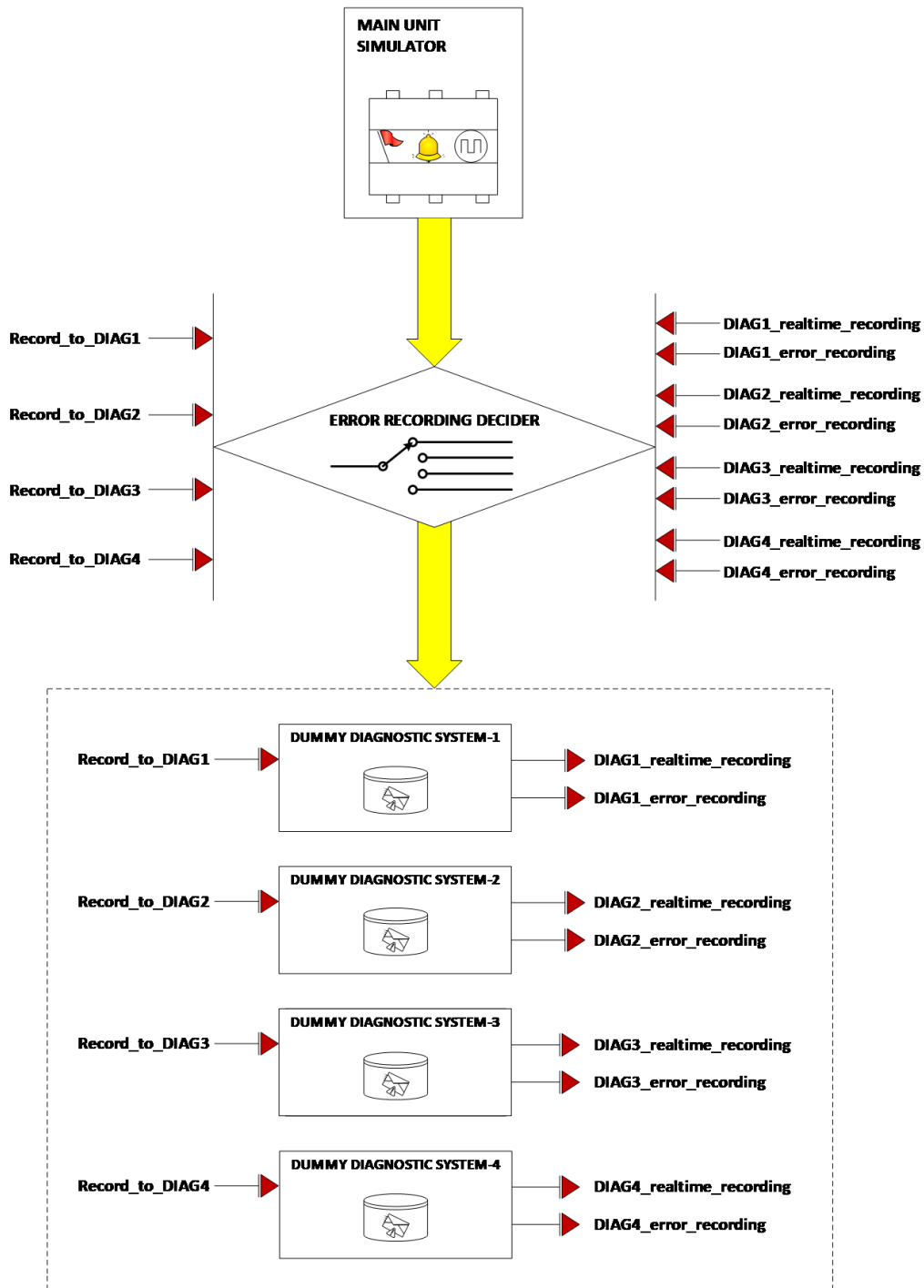


Figure 5.19 : Block diagram of the error recording decider test environment.

Tests were performed repeatedly, one of the results for three different scenarios are given below with the test descriptions.

No error scenario:

Test scenario for normal operation. The MFDS is enabled with the Real Time Record command and no error occurs. The result of the related test is given in Figure 5.20:

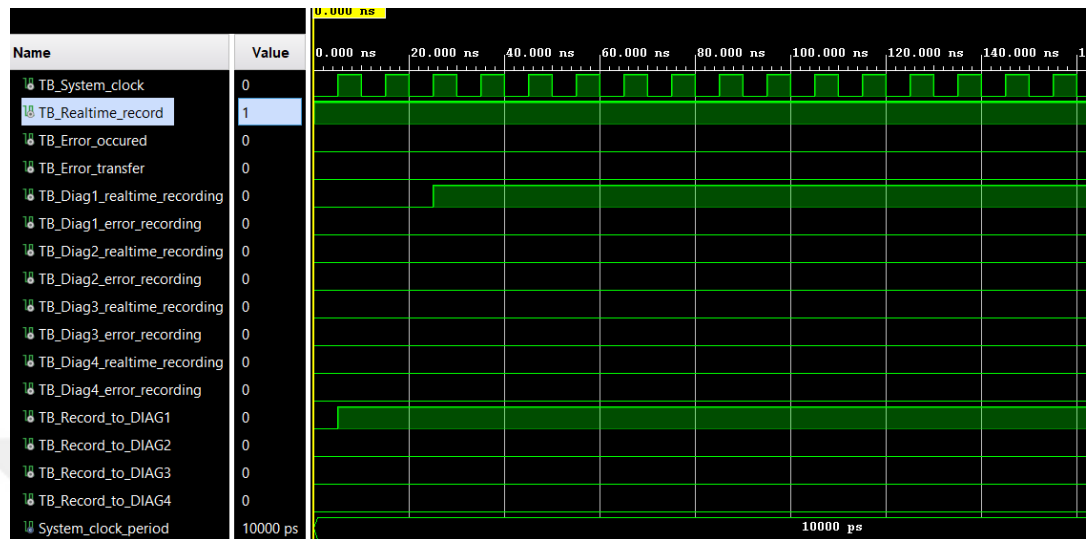


Figure 5.20 : No error scenario test result.

While system clock and real time record command were received, no error occurred indication sent from the Main Unit as seen in Figure 5.20. In this case, 1st Diagnostic System was enabled according to the order and the others were disabled, since all of the Diagnostic Systems were available after the start-up.

Single error scenario:

Test scenario for single error case. An error occurs, while the MFDS is real time recording. The result of the related test is given in Figure 5.21:

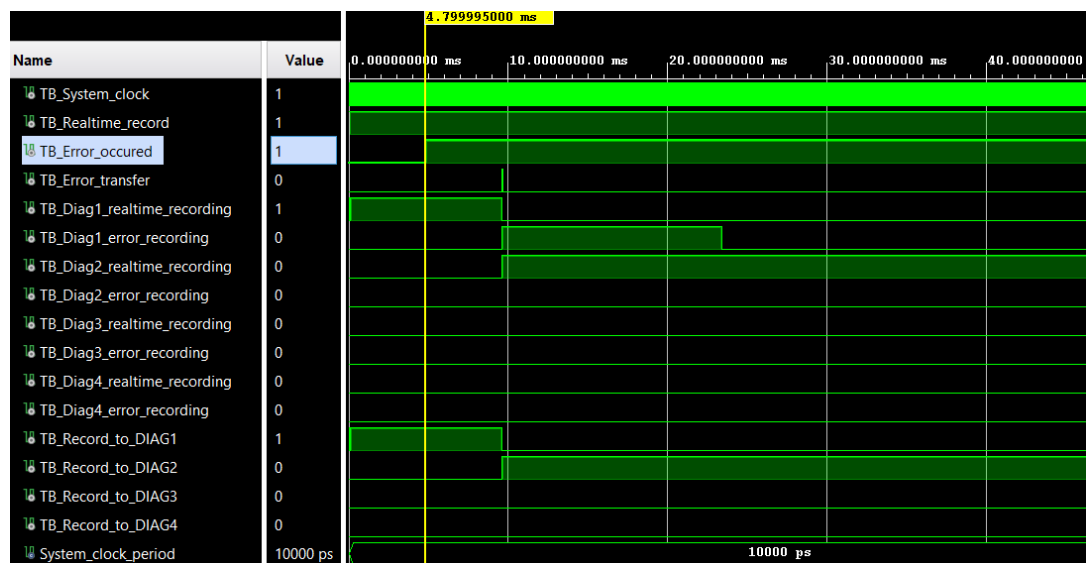


Figure 5.21 : Single error scenario test result.

As seen in Figure 5.21, system started operating with 1st Diagnostic System as was in the case with no error. Then, 1st Diagnostic System started error transferring when the 4-second error message was ready for transmission. The system appoint the time that error transfer process should start automatically by taking the moment the Error Flag had logical “HIGH” value as reference. The main point should be emphasized here is that; activation of the 2nd Diagnostic System and the start of 1st Diagnostic System's error transfer happens at the same time. In other words, the transition of the 1st diagnostic system to the error transmission state and the 2nd diagnostic system's transition to the real-time recording state occurred simultaneously. Thus and so, recording continued uninterrupted.

Worst case scenario:

More than three consecutive errors occurs, while the MFDS is real time recording. The result of the related test is given in Figure 5.22:



Figure 5.22 : Worst-case scenario test result.

The test image shown in Figure 5.22 is belong to the result of worst-case scenario. As explained before, the MFDS was designed on the purpose of capturing all error messages under all conditions. When the system has been designed, the needed Diagnostic System number was determined as four according to the principle of full redundancy. In accordance with the worst-case scenario five error created sequentially by the Main System Simulator and results were observed. The flow of the test is also given step by step below:

- The system started operating with 1st Diagnostic System as no error was at the beginning.
- With an error occurred indication, the corresponding system transitioned to the error transfer state, and the next diagnostic system started real time recording at one and the same time.
- The transition from real time recording to the error transfer state for a diagnostic system always performed with starting real time recording of the next diagnostic system.
- The most important point is that, the MFDS continued operating without any interruption. Even if consecutive errors occurred, the first diagnostic system got ready for real time recording before the fourth diagnostic system started error transferring.

In consequence, it has been observed that operation continued cyclically in the consecutive errors case, since the first diagnostic system is always available again before the last diagnostic system switches to the error transfer state. It has been observed that system works properly and not missing any error message even in the worst-case scenario based on the test results.

5.3.2 The test environment

As explained before, the FPGA-based development process is only a behavioural proposal for future, and the hardware processes for FPGA are out of the thesis' scope. So, modeling the behaviour of the external units that diagnostic system will operate with or integrated into is also needed for the final tests, in contrast to the system based on microprocessor.

While creating the new test scenario and modules, the error scenario defined in scope of the thesis was taken as reference, and the simulation model were set. The external process blocks (or modules) that needed for tests serve the same purpose as the modules designed in scope of the FDS tests, and for this reason, they are quite similar. They were just updated as to run in keeping with the uninterrupted operation. While the updated simulation data generator unit creates and transmits the data from the sensors even in the case of an error, the main unit simulator was updated to send the

commands and statuses according to the new continuous duty principle. The block diagram of the test environment is given in Figure 5.23.

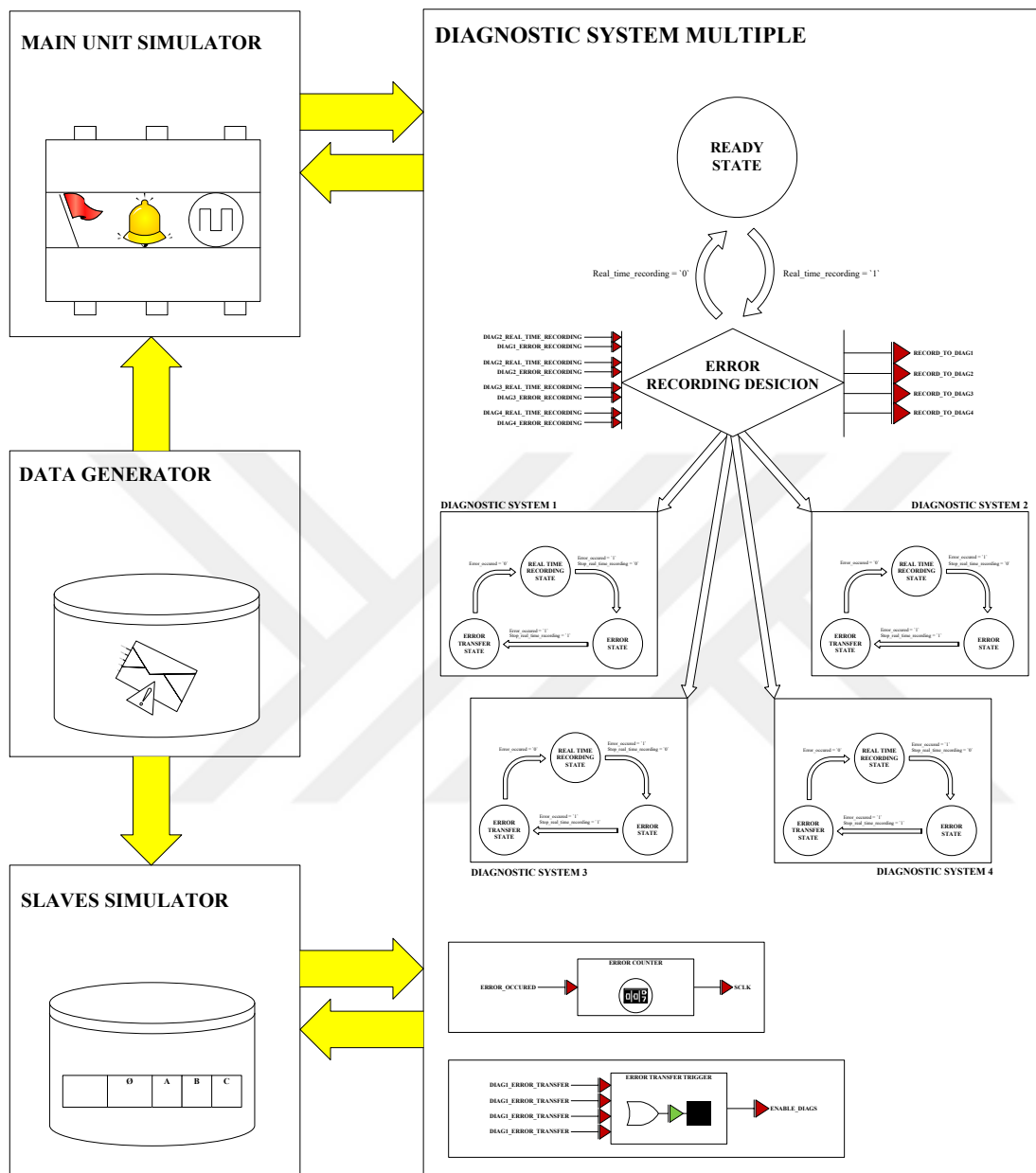


Figure 5.23 : Block diagram of the final test environment.

5.3.3 The test results

The error scenario detailed in microprocessor-based design sections was run by using the test model in section 5.3.2 and run repeatedly.

The starting and ending points for the test scenario are determined in software. The tests that start with initialization of the system, terminated automatically after the

scenario is completed. The results given in this section contains the time period between these starting and ending points. Tests were performed in several scenarios repeatedly on the VIVADO Design Suite behavioural simulation environment, and the results for three different scenarios are analyzed in this section. First the test descriptions and the results on the simulation test bench are given below respectively. The expected result is that the system successfully records the sensor data without loss.

No error scenario:

Test scenario for normal operation. The MFDS is enabled with the Real Time Record command and no error occurs during the test. The expected result is that the system successfully records the sensor data without loss. The related results of the no error test scenario is given in Figure 5.24 and 5.25 respectively.

Single error scenario:

Test scenario for single error case. An error occurs, while MFDS is real time recording. The expected result is that the first diagnostic subsystem continues the recording process until the error message is generated, and goes into the error transfer mode with the generation of the error message containing the 4-second sensor data. When the related diagnostic system switches to error transmission mode, the second diagnostic system should transition to real time recording mode and undertake the recording task for new sensor data. The related results of the no error test scenario is given in Figure 5.26 and 5.27 respectively. Lastly, the active fault diagnostic system changed over only once in single fault scenarios as shown in Figure 5.28. This is because the number of errors is one.

The main motivation of the design of an FPGA-based MFDS, which has the ability of parallel operation, is to provide a system having the ability of continuing recording without any data loss, even in sequential error scenarios. In this context, the results for the worst case scenario are examined in detail. In this scenario, it is aimed to see that changeovers occur after each error. Thus, uninterrupted recording will be performed without any loss even in sequential errors.

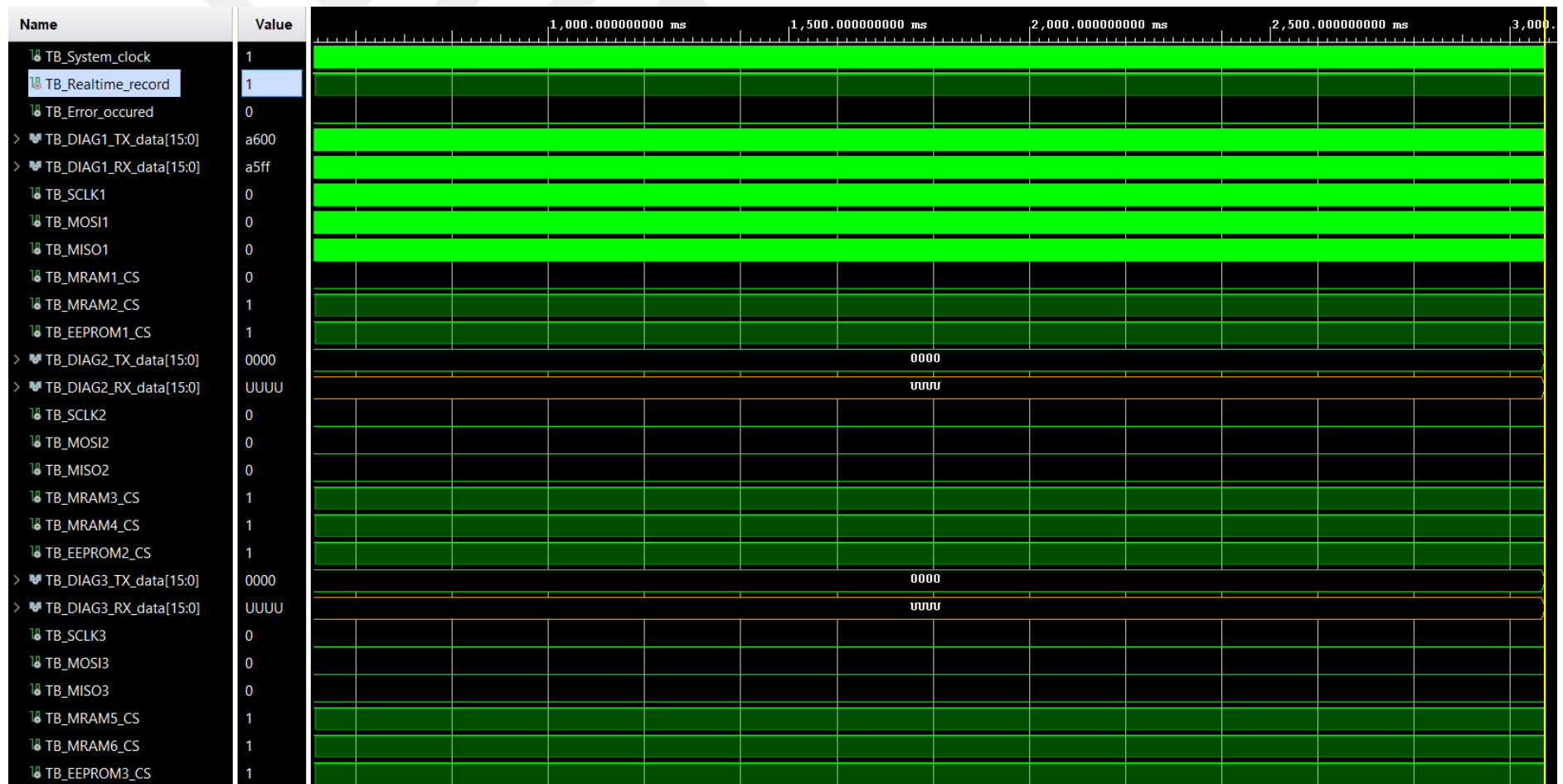


Figure 5.24 : No error scenario test result part-1.

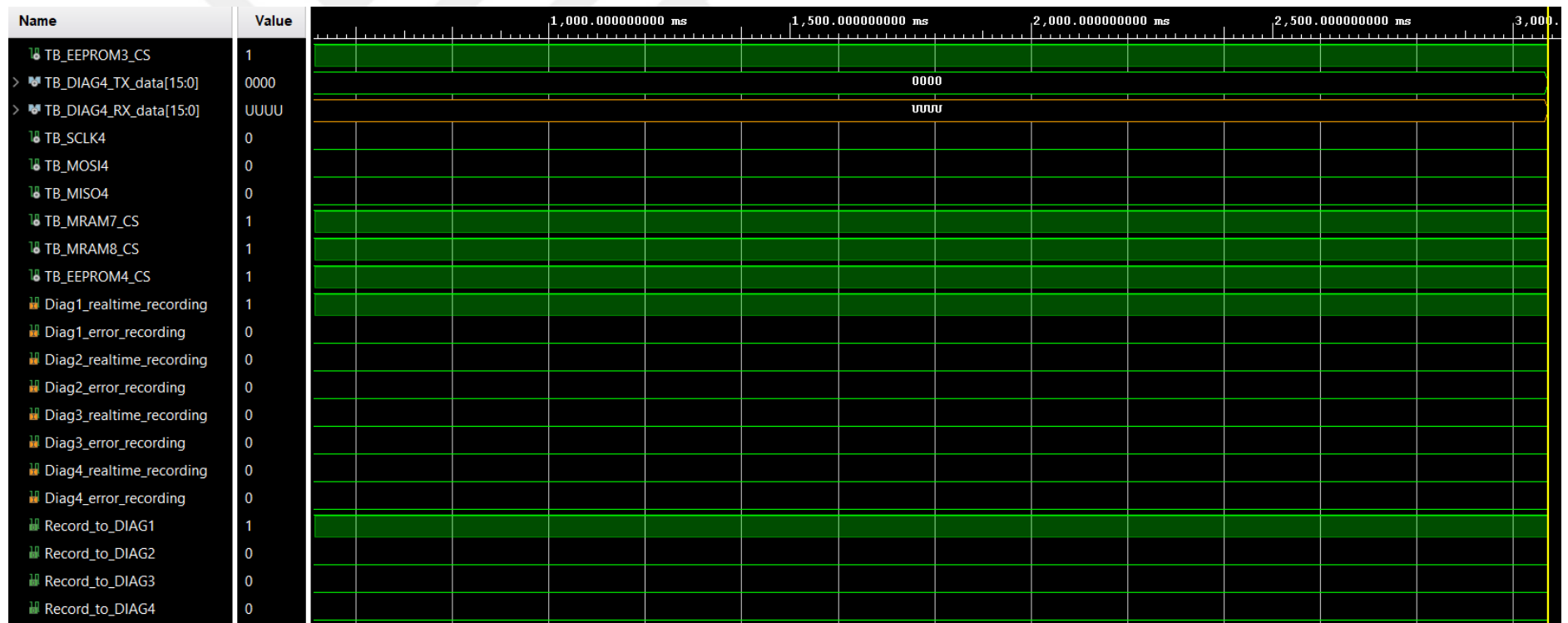


Figure 5.25 : No error scenario test result part-2.

Hereby the critical point is that the second diagnostic system starts real time recording at the same time with the first one starting the error transfer, and by doing so, no loss was observed in the sensor data.

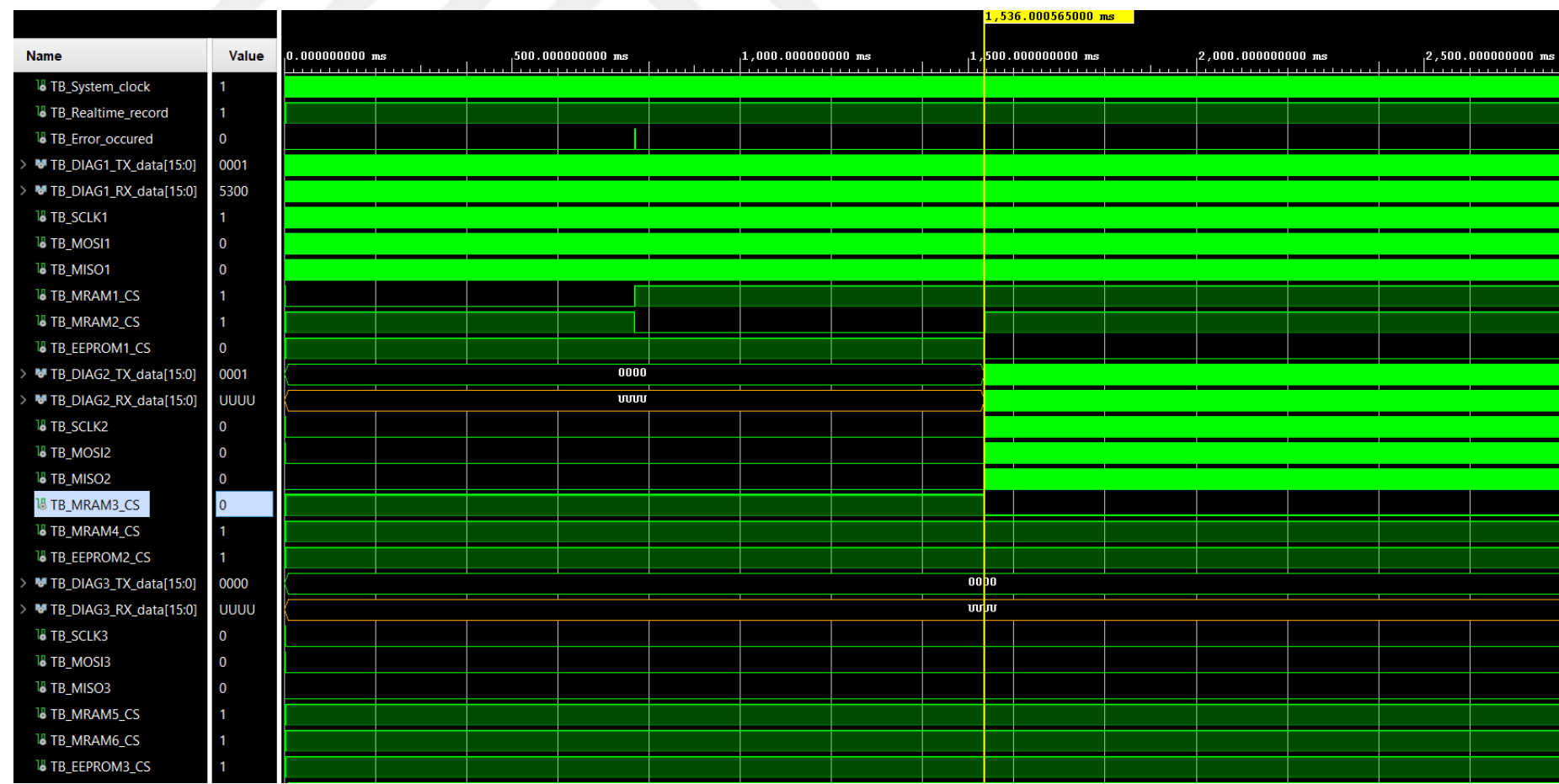


Figure 5.26 : Single error scenario test result part-1.

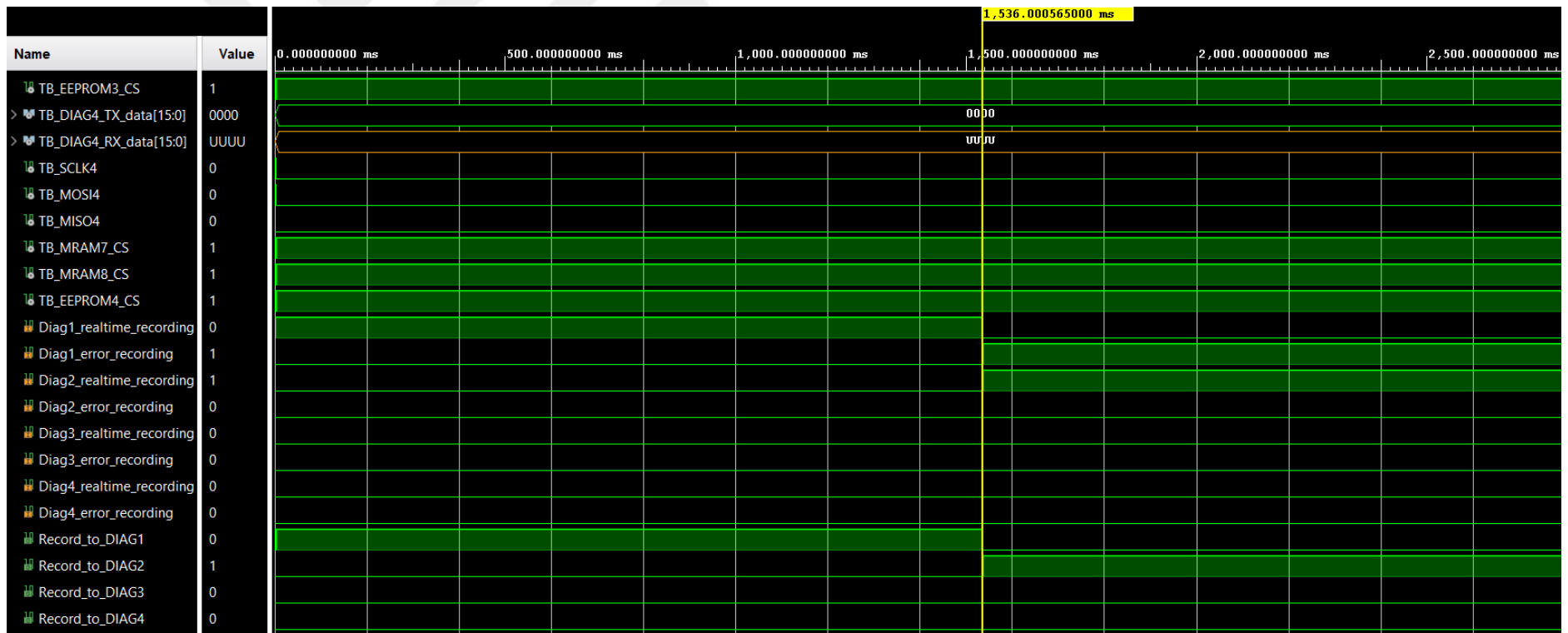


Figure 5.27 : Single error scenario test result part-2.

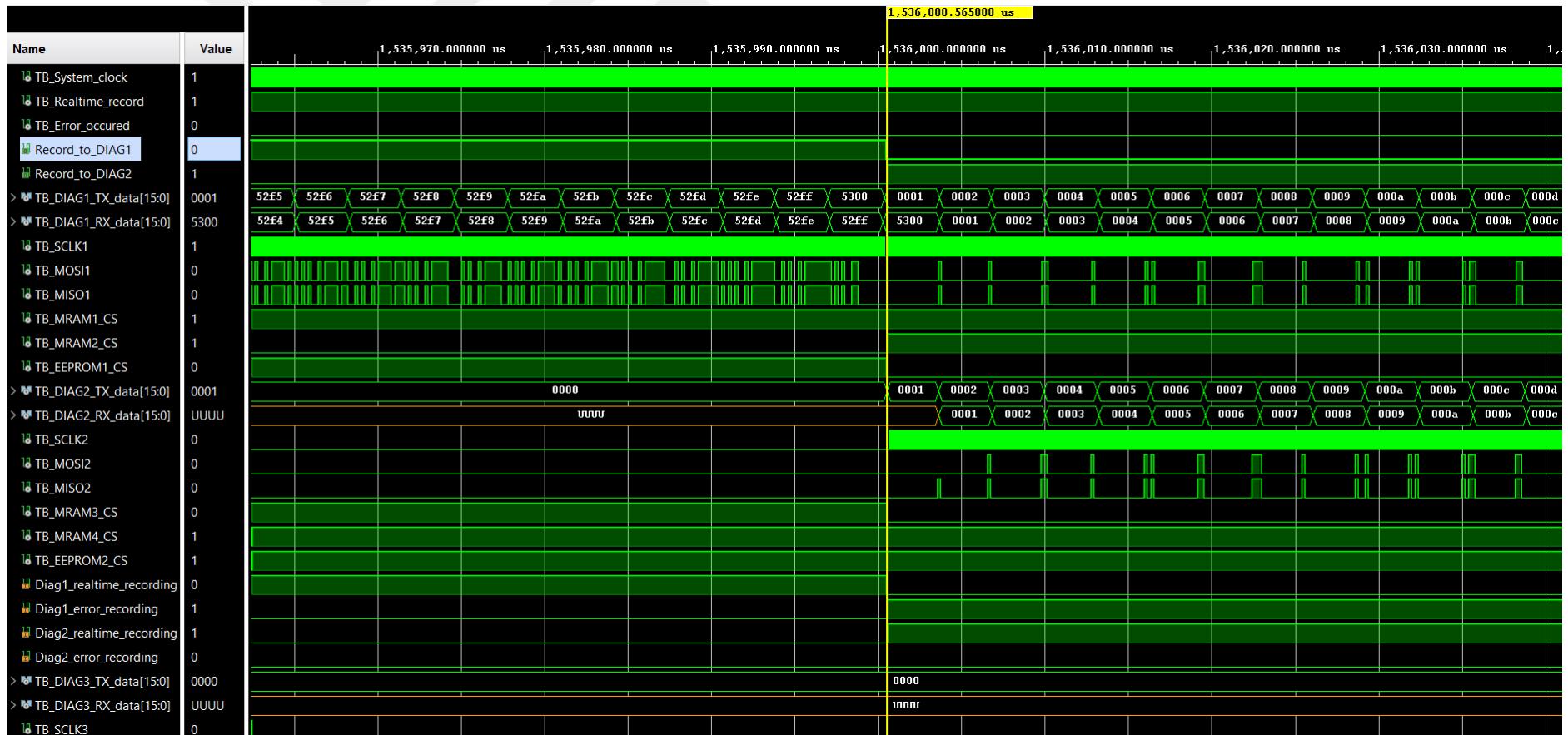


Figure 5.28 : The changeover moment of active diagnostic system.

Worst case scenario:

Test scenario for worst-case, five consecutive errors occur, while the MFDS is real time recording. The expected result is that first diagnostic system starts real time recording and continues the recording operation until an error condition. It should go into the error transmission mode with the generation of the error message containing 4-second the sensor data. When the related diagnostic system switches to error transmission mode, the next diagnostic system should transition to real time recording mode and undertake the recording task for new sensor data. This scenario should happen in the same way for all error situations and the next diagnostic system should start real time recording at the same time with the previous diagnostic system starting the error transfer. In conclusion, it should be seen that the MFDS is capable of uninterrupted logging even for the worst-case. No loss of data or time interval that measurements cannot be performed should be observed in the sensor data. The general view of the results for the related test are given in Figures 5.29 and 5.30.

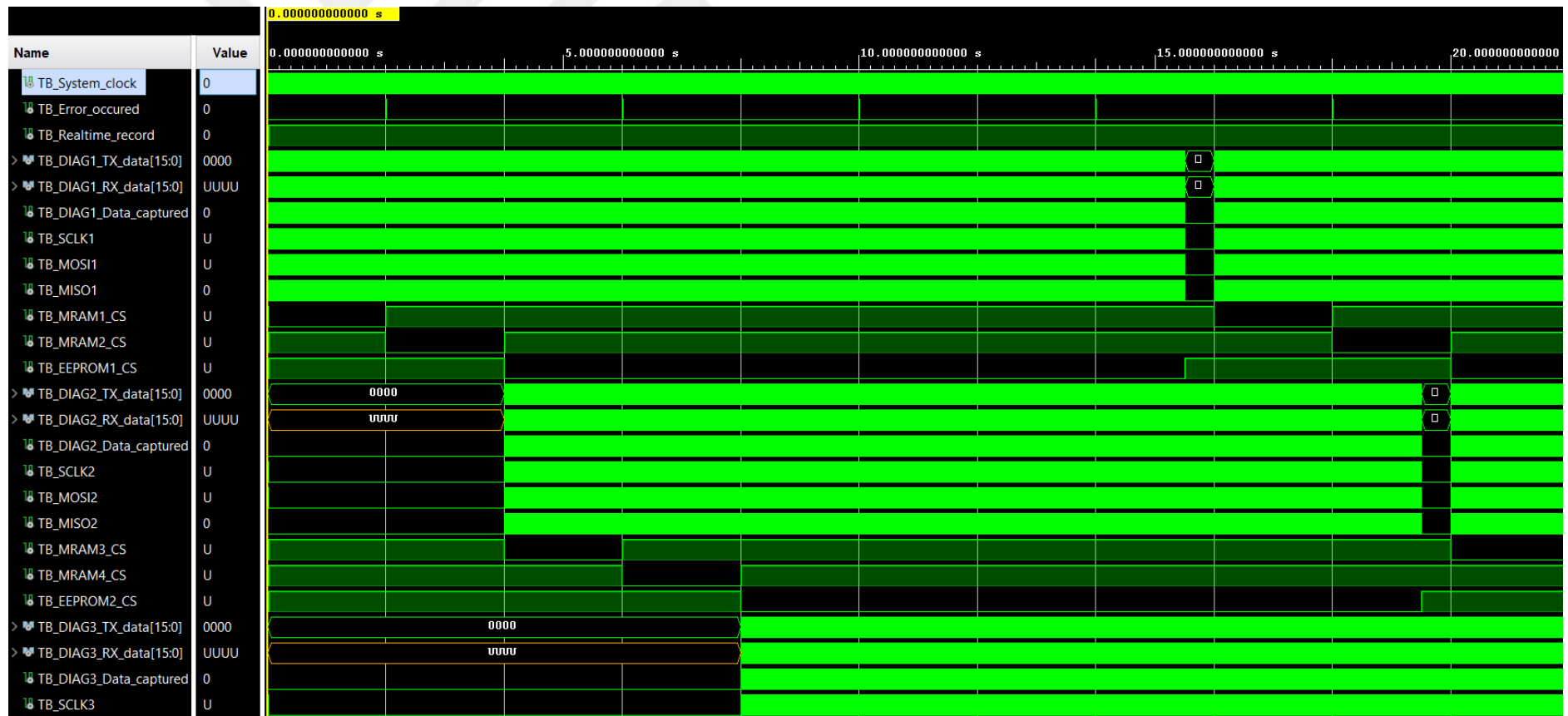


Figure 5.29 : Worst case scenario test result part-1.

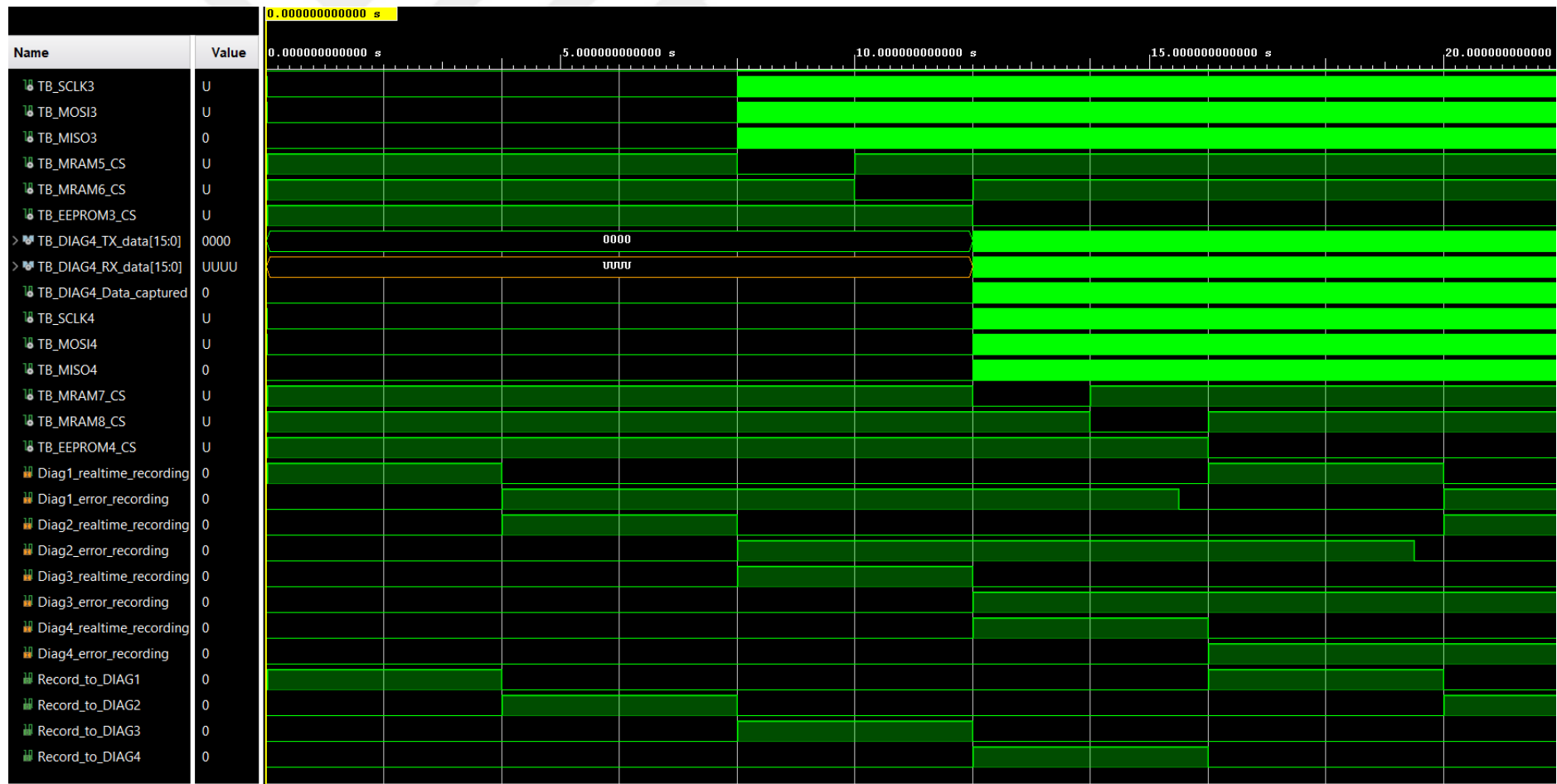


Figure 5.30 : Worst case scenario test result part-2.

The results given in Figure 5.29 and 5.30 are the view of the process performed under a condition including five sequential errors. As mentioned before, the main motivation of the design of an FPGA-based MFDS is to provide a system having the ability of continuing the recording without any data loss under all circumstances. Therefore, the important parts are highlighted with red colour in Figure 5.31 and their explanations are given according to the numbers below:

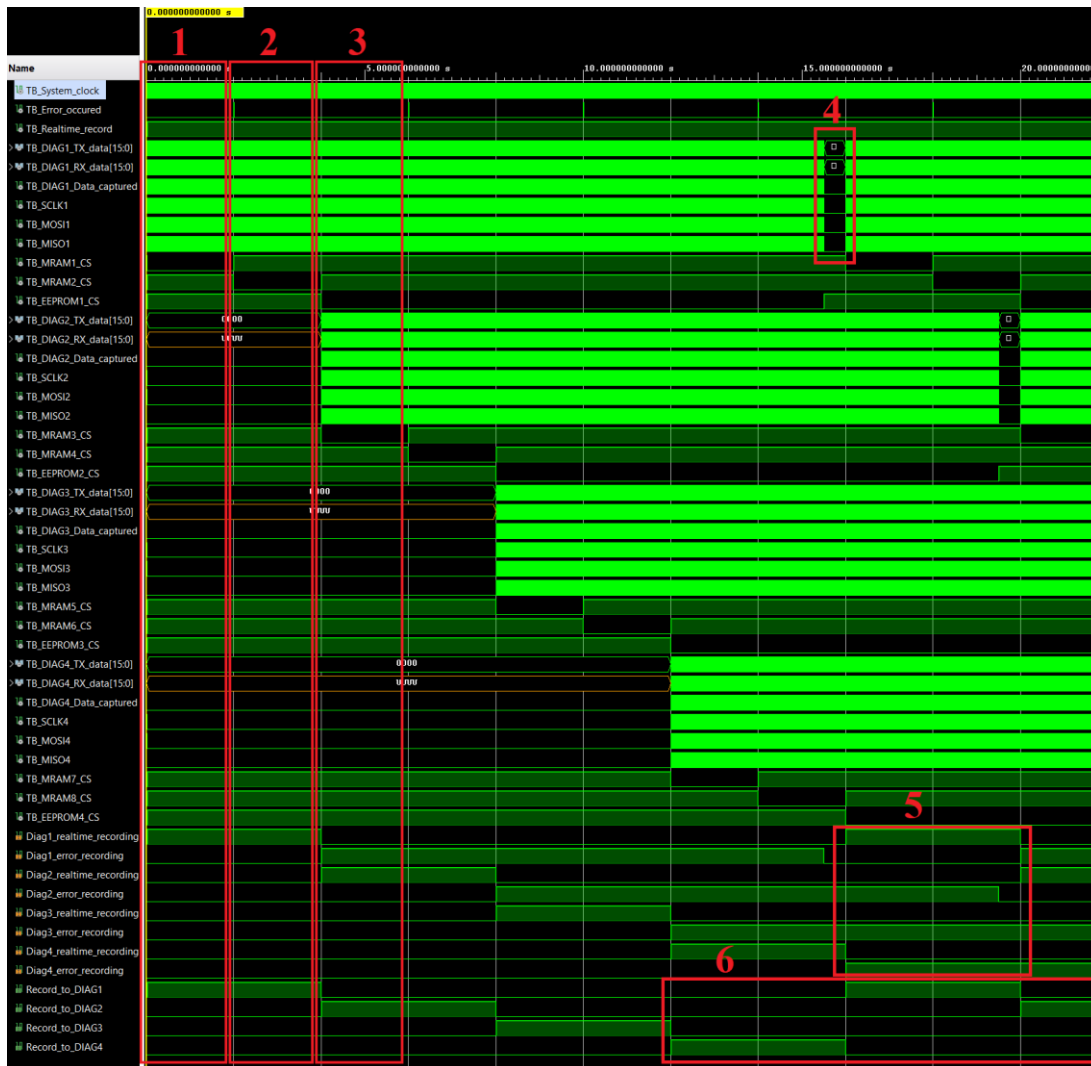


Figure 5.31 : Emphasized parts of the worst case scenario test result.

1. With the Real Time Record order given by the Main System which the MFDS is integrated, the system start real time recording. The corresponding recording operation was performed by the first diagnostic system as arranged in the order, since all diagnostic systems were available. Therefore, only the first diagnostic system were switched to RTC State. Transmission of the SPI1 clock was started with the transition to RTC state, while the others were disabled. The

related slave unit MRAM1 was activated while the other devices were kept disabled. The dummy sensor data sent from the Main System was retrieved to the DIAG1_TX_data buffer and sent via MOSI bus properly. The data received from the MRAM1 was properly read in the MISO bus and registered into the DIAG1_RX_data buffer. DIAG1_Data_captured signal was triggered after all successful word transfers. System status indications were sent correctly throughout the state. (Diag1_real_time_recording = 1, Others = 0)

2. In this part, it has been seen that the related slave unit MRAM2 was activated while MRAM1 was deactivated again with logical "HIGH" value of the Error_occurred flag. The other devices were still disabled. The transmission of the SPI1 clock resumed uninterruptedly without any clock displacement or miscalculation. The traffic was also sent and received properly. The transition to the Error Transfer State performed automatically and accurately when the 2-second sensor message containing the data after the error moment was created.
3. This part is the section that first diagnostic system transitioned to the error transfer state while the second diagnostic system started real time recording. The transmission of the SPI2 clock was started without any clock displacement or miscalculation on SPI1 clock transmission. The related slave units EEPROM1 and MRAM3 were activated accurately. The traffics were observed on both MOSI1, MISO1 and MOSI2, MISO2 buses properly. System status indications were sent correctly throughout the state. (Diag1_error_recording = 1, Diag2_real_time_recording = 1, Others = 0) Thus, the two diagnostic systems worked properly as responsible for different operations in parallel and without affecting each other.
4. As seen in the Figure 5.31, the processes explained in 1, 2, 3 were performed by every diagnostic system based on error occurrences. One of the key features of the MFDS is the approach of 4 diagnostic systems operating in a cyclic manner to record the relevant sensor data and error messages that will occur under all circumstances. It is provided by the rule that first diagnostic subsystem switched to the ready state before the last one transitioned to the error transfer state. This is a principle based on the theoretical calculations made while the system was designed. As seen in the part in 4, the first diagnostic system already transitioned to the ready state before the fourth diagnostic

system started the transfer of the error message. In related section, it has been seen that there was no clock pulse, no data transactions on MOSI/MISO buses and correspondingly no written or read message, since the first Diagnostic System was not activated for real time recording by the Error Recording Decider. However, the system clock was sent to the corresponding system as per the design of the MFDS and it was provided the 1. Diagnostic System to wait ready (or standby).

5. Part 5 can be considered a verification of the working mechanism described in part 4. As shown in part 5, the first Diagnostic System sent the real-time record and error record statuses as zero, because it was in the ready state before the beginning of fourth Diagnostic System error transferring. With the transition of fourth Diagnostic System to the error transfer state, it started recording and sending the real-time recording status as logic HIGH.
6. In the last highlighted part, the results for the role assignments commands ensuring the synchronization between the parallel systems were emphasized. As seen in the waveforms, the enabling (or activation) commands were assigned to the Diagnostic system one by one according to their occupancies and the cyclical operating principle as new error occurred notifications arise. However, the first diagnostic system returns to the ready state before the last one must stop real time recording, even if consecutive errors occur as shown in the Part 6. In that part, the first Diagnostic System was re-triggered for real time recording at exactly the same time with the fourth Diagnostic System's transition from the error state to the error transfer state.

As detailed before, MFDS keeps recording of sensor data for 2 seconds more after the moment fault occurred by using the current system that has performed the real time recording. Additionally, the system also determines synchronously the new diagnostic system that will take over the real time recording task when the relevant system starts the error transfer. In case the 4-second data including the waveforms of sensors is recorded, the relevant Diagnostic System switches to the error transfer state, while the next Diagnostic System takes over the recording of new sensor data task from the former. The changeover moments of the active system observed within the scope of the test scenario are given in Figures 5.32, 5.33, 5.34, 5.35 and 5.36 respectively for all Diagnostic Systems in the error situations.



Figure 5.32 : The first changeover of active diagnostic system.



Figure 5.33 : The second changeover of active diagnostic system.



Figure 5.34 : The third changeover of active diagnostic system.

The most important point in results is that, there is no clock pulse, no data transactions on MOSI/MISO buses before the fourth error moment as shown in the Figure 5.35. It is due to the fact that the first Diagnostic System was already completed recording of the first error message, before the fourth diagnostic system started the transfer of the fourth. The system clock was sent to the corresponding system as per the design of the MFDS and it was provided the 1. Diagnostic System to wait ready (or standby).

This mechanism, which is called multitasking or parallel processing, ensures that the system has the ability to record the sensor data continuously without any loss. The corresponding results are seen in Figure 5.35 and 5.36 by including the statuses of all Diagnostic Systems in the error situations within the scope of the test scenario.



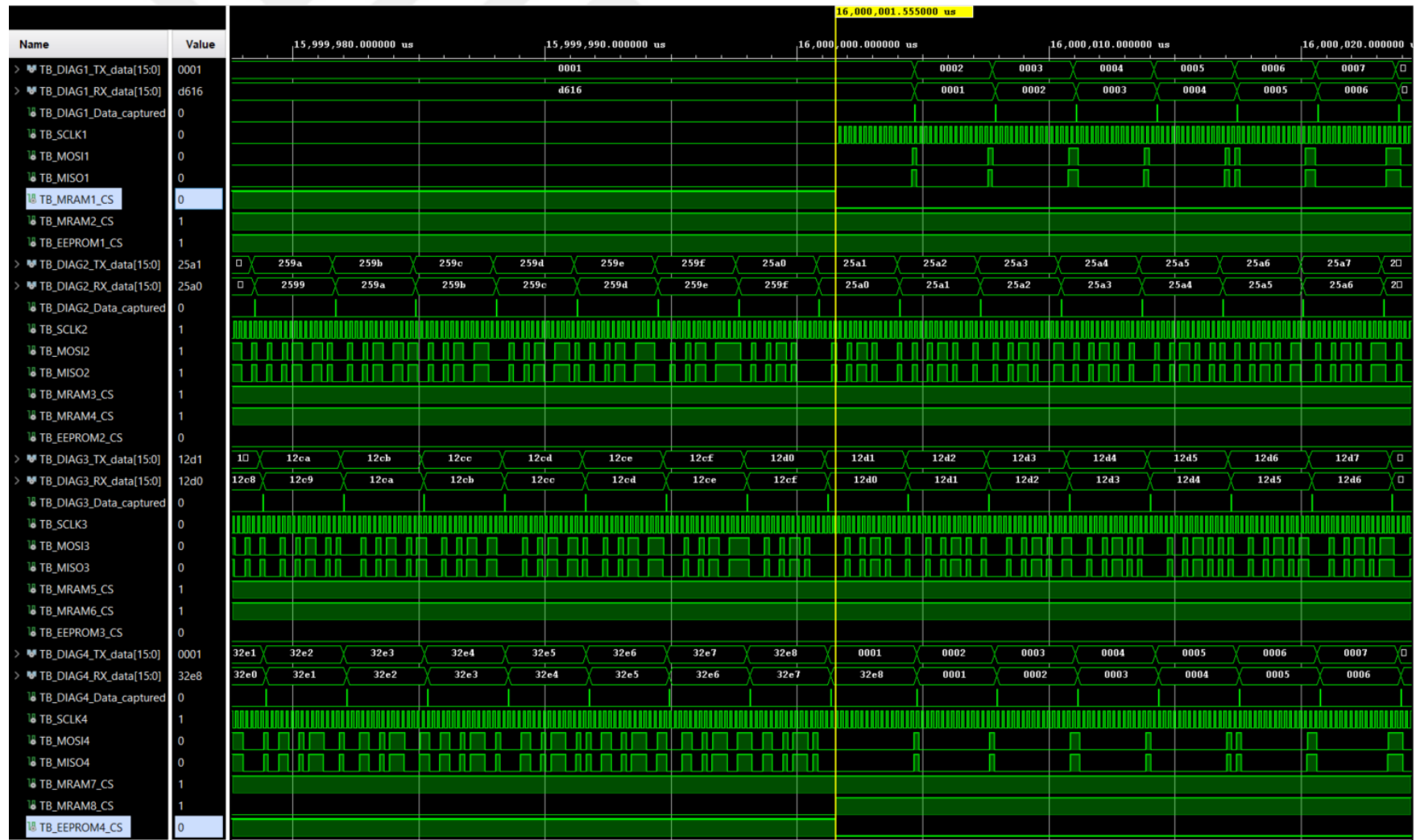


Figure 5.35 : The fourth changeover of active diagnostic system.

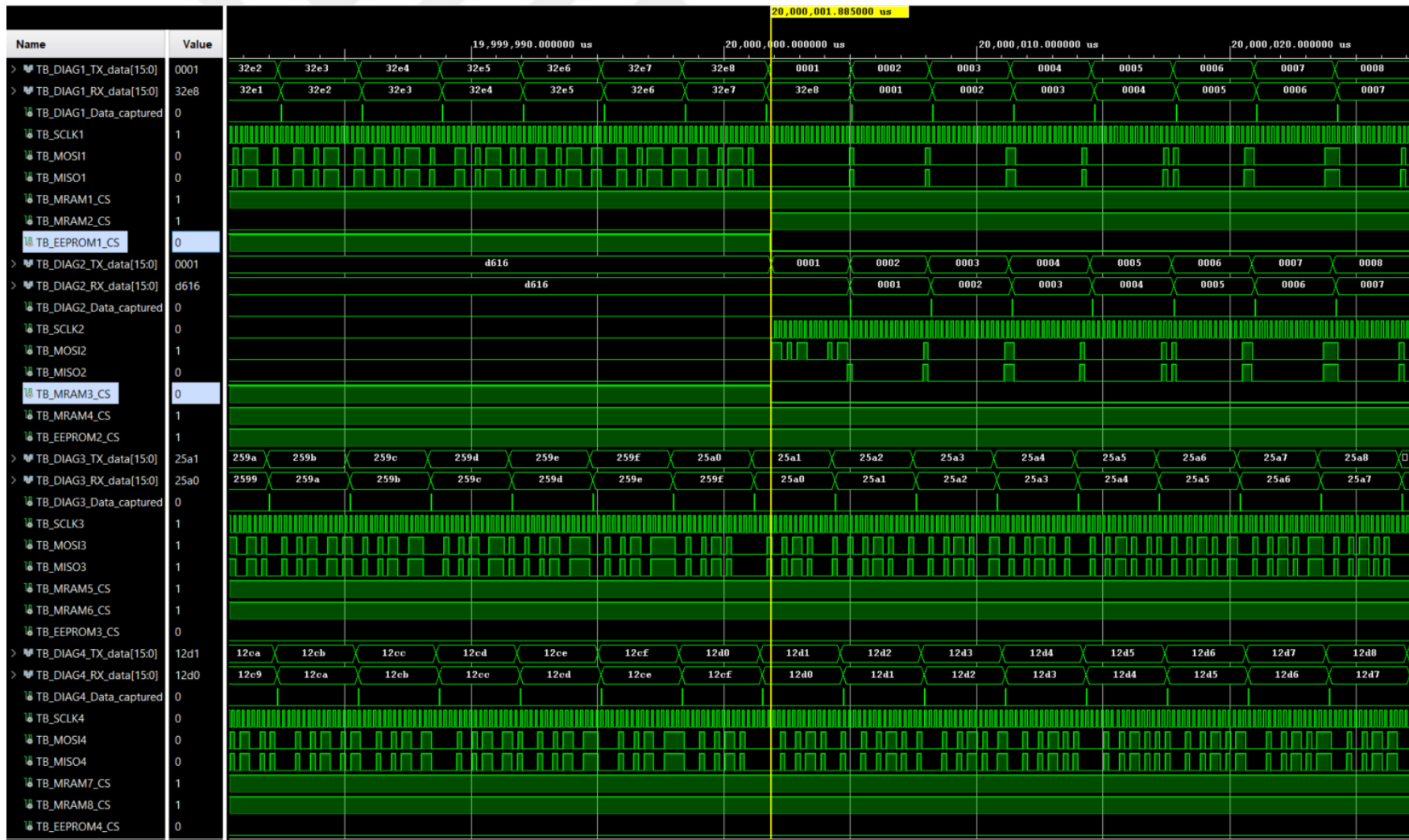


Figure 5.36 : The fifth changeover of active diagnostic system.

6. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a Microprocessor-based Fault Diagnostic System, which operates in conformity with or integrated into main units such as control, automation, power units to be used in numerous fields, was designed including the software and hardware. This FDS is fundamentally a logger system that continuously records sensor data to be sent by the main system, creates the “Error Messages” by using the waveforms measured by the external sensors in case of a special situation determined as an error, and stores the error messages even in the case of a power cut, as error messages may contain electrical anomalies including also power failure. The related system is able to provide different operating modes such as real-time recording, error state, and error transfer mode according to the technical requirements emphasized related sections.

Additionally, an FPGA based technical proposal which is named Multitasking Diagnostic System has been submitted in scope of the thesis. A new system model providing uninterrupted operation was created by considering the advantages of using FPGA for embedded applications such as parallel processing, speed, flexibility, extendibility.

In the following sections, the systems are examined, and a detailed comparison containing their advantages and disadvantages over each other according to the requirements and circumstances, are given.

6.1 Contributions of This Study

In line with this purpose, firstly a microprocessor-based embedded system was designed whose hardware designed in Altium PCB Design Software & Tools, and software in CCS by using C programming language by starting from the requirements, conceptual and detailed system design phases.

After the hardware and software development processes were completed, the accuracy and performance tests were performed repeatedly on the microprocessor-based ICU setup for the FDS. As the result of the tests, there has been no corruption between the

source and destination, no bad data or missing packets. It has been concluded that the FDS is designed successfully within the scope of the thesis, and works faultlessly.

However, as mentioned in the related sections, an error message created by using the 24 sensors' data sampled at 5kHz for 4 seconds is transferred from temporary memory structures to the permanent in case of an error. The size of an error message has been calculated as 980.000 bytes. Hence, it is has seen that this error transfer process takes approximately 11 seconds on average for the microprocessor-based system, which means that new errors cannot be recorded in the meantime. An example oscilloscope screen of the relevant test is given in Figure 6.1.

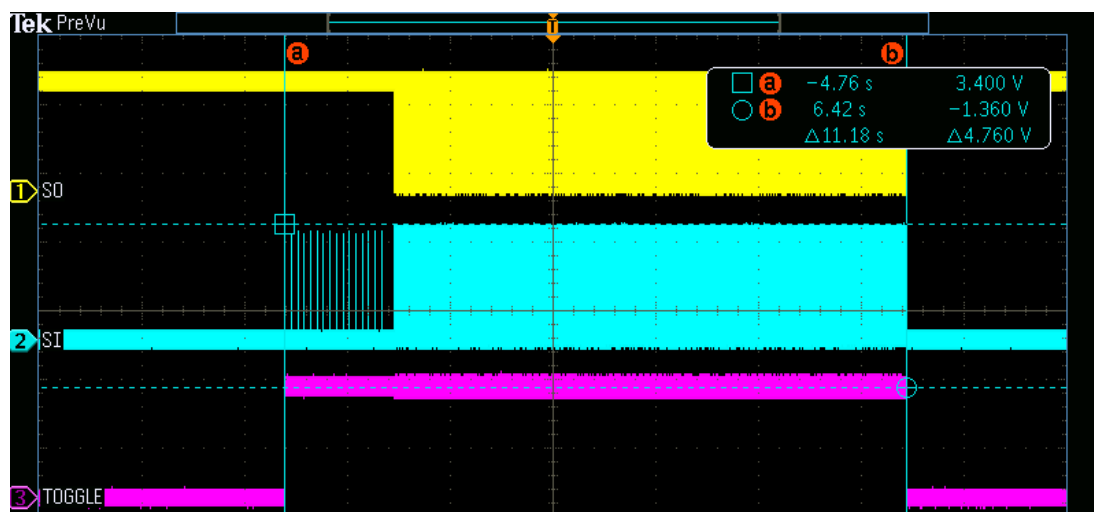


Figure 6.1 : The time interval new sensor measurements cannot be recorded.

Nevertheless, it also should be highlighted that this time is only valid for the system designed according to the request and needs of the project in thesis' scope. The time could be shortened by mitigating the decisions or circumstances. For instance, sampling frequency could be decreased, or time duration of an error message also might be shortened. Of course, these decisions are also trade-offs, and easing conditions will have performance-reducing consequences on the results as well. For instance, lowering the sampling frequency results in a rough recreation of the error waveforms, and shortening the error recording period means more limited information about before and after the error moment and could make debugging more difficult.

For this reason, also an FPGA based technical proposal which is named Multitasking Diagnostic System has been submitted in scope of the thesis. The related system is developed starting from zero by using VHDL in the VIVADO Design Suite environment as high-level. A new system model providing uninterrupted operation is

created by considering the advantages of using FPGA for embedded applications such as parallel processing, speed, flexibility, extendibility.

Accuracy and performance tests were performed also for the FPGA-based system through behavioural simulation on the VIVADO Design Suite environment. The system was retested again and again by simulating the real environment, and it has been seen that the MFDS is capable of uninterrupted logging even for the worst-case as shown in Figure 6.2



Figure 6.2 : Uninterrupted logging (MFDS).

As a result of the studies carried out within the scope of the thesis, a microprocessor-based diagnostic system that has been designed, verified and validated as both hardware and software. The related system is actively used for different applications, currently operates in the field. In addition, a utility model that could be applied in further studies and projects with more difficult requirements and conditions has been created with the FPGA-based technical proposal created.

6.2 Comparisons

In this section, a detailed analysis and comparisons are given for the systems designed within the scope of the thesis, including the current situations, the features, advantages and disadvantages.

First of all, it should be highlighted that both systems have been working faultlessly, because it has been clearly seen that the data read and written into both systems are accurate and correct. There has been no corruption between the source and destination, bad data or missing packets as the result of the tests.

When the systems are analyzed in terms of performance, it can be said that the FPGA-based system promises more success. The two also differ in the methods used to process instructions. Microcontrollers read through each line of the program sequentially; the commands are also processed in sequence. FPGAs can process orders simultaneously and can execute numerous lines of codes at a given moment. As mentioned earlier, the error transfer process takes approximately 11 seconds on average for the microprocessor-based system, which means that new errors cannot be recorded in the meantime. Even if this time is shortened by mitigating the decisions or circumstances, there will be performance-reducing effects on the results as well. This time also could be optimized by switching the processor from one code to another to achieve parallelism. However, this also will require effort, time and expertise.

From the viewpoint of flexibility, FPGA-based system is more versatile and flexible as expected due to FPGA's nature. The microprocessor based system only allows for reprogramming of firmware, which greatly limits the options for any modifications. For example, only a finite number of I/Os are recommended to be used for SPI communication in the microprocessor-based systems. Other general-purpose input/outputs could be used also, but it will take more resources and may not work on all cases. For example, SPI master can easily be bit-banged but SPI slave at 10MHz is probably impossible on a slow micro, and even at much lower speeds interrupts must be blocked. The specified pins are connected to dedicated hardware inside which makes it easier to use and it generally will work better, faster and will use less resources. It is suggested reading the datasheets to figure out which pins should be used and if there are alternatives available.

One of the thesis' aim is designing a general-purpose embedded FDS used in applications in different fields. All requirements and parameters, from the number of sensors to be recorded to the sampling frequency, from the communication protocols to the recording time, may change according to the corresponding project basis. In this context, both systems are designed generically so that they can be used in various fields and in different projects. Reprogramming is possible in firmware for both of the systems. But, the microprocessor-based system cannot be reprogrammed as hardware, it should be used as is. In this sense, the FPGA-based system may be more advantageous in some specific projects, since the logic gates can be rewired as often as required to change the program and carry out a different task due to the nature of FPGA.

Another important issue is development efforts. It takes time and is needed more than a basic understanding of coding language to create an FPGA function, as all the codes are compiled from scratch and then converted into machine language. The designer should have knowledge and understanding of both programming and hardware circuits. On the other hand, microprocessors are easy to use by anyone with a basic understanding of coding can program a microcontroller. Lastly, ready-made packages for microcontrollers to perform specific tasks are widespread and easily accessible.

Microprocessor-based system is simple to configure and can handle high-speed sequential data, specific to the purposed application in thesis' scope. However, they lack some of the features of FPGAs. Microcontrollers are more flexible when it comes to programming, but cannot be customized accurately through hardware.

When the systems are examined in terms of trouble-shooting and bugfix, it is seen that debugging and corrective works are faster and easier in the microprocessor-based system. Debugging and monitoring are easier for microprocessor due to their sequential logic. On the contrary, a steep learning curve is faced when dealing with FPGA programming. It may be necessary to use trial and error method in FPGA studies or external systems such as microcontrollers in development studies.

In terms of power consumption, it is foreseen that FPGA-based system will consume more power than microcontroller-based system for various reasons. Due to the fact that FPGAs have been used for emulation and prototyping, but they are not as efficient as other alternatives. The unused transistors, inefficient clock trees and extended signal

paths also contribute to increased power consumption. On the other hand, microcontroller-based systems consume much less power compared to FPGAs.

The last drawback of FPGAs is that they tend to be more expensive than microcontrollers. As explained in hardware design studies, the microcontroller, used in the first system, is TMS320F28377D manufactured by Texas Instruments. While the cost of this microprocessor is between \$ 20-30, it has been seen that the possible FPGA System On Chip (SOC) ICs to be used are in the range of \$ 100-150 according to market investigations. In this sense, it anticipated that the FPGA-based system will bring higher upfront costs in comparison to microcontrollers. A comparison table also given in Table 6.1 to summarize the results:

Table 6.1 : Comparison table.

SYSTEM CRITERIA	Microprocessor-based	FPGA-based
Accuracy	Excellent	Excellent
Performance	High	Excellent
Flexibility	Reprogramming is possible in firmware only	Hardware and firmware reprogrammable. Superior customization.
Multipurposeness	High	Excelent
Instruction set	Fixed instruction set	Does not have a fixed instruction set.
Configuration	Complex to configure an FPGA	Easy to configure
Power consumption	Much less power compared to FPGAs	More power consumption
Programming	Easy to Program	Specialized knowledge requires
Debugging	Easy	Hard
Processing	Sequential processing	Parallel processing
Hardware Design	Simple	Comparatively complex
Architecture	Processor based	LUT based
Tools	Open source tools available	No portability across tools
Cost	Cost effective	Costly

6.3 Suggestions for Future Work

In conclusion, both the microprocessor-based and FPGA-based embedded FDSs to be used for different applications have been designed, verified and validated. Yet, it should be emphasized that the microprocessor-based diagnostic system has been implemented and realized, while the FPGA-based system is only a proposal. Even though tests were performed also for the FPGA-based system, no hardware design or tests on physical setup (equipment) have been made. The progresses of present studies for the related systems are also summarized in Table 6.2.

Table 6.2 : Systems' progress table.

SYSTEM PHASE	Microprocessor-based	FPGA-based
Conceptual system design	COMPLETED	COMPLETED
Detailed system design	COMPLETED	COMPLETED
Hardware design	COMPLETED	NOT DONE
Firmware design	COMPLETED	COMPLETED
Tests	COMPLETED	COMPLETED
Implementation	COMPLETED	NOT DONE
Realization	COMPLETED	NOT DONE
Verification	COMPLETED	NOT DONE
Validation	COMPLETED	COMPLETED

Therefore, the FPGA-based system should be implemented and realized like the microprocessor-based system to be sure about the results. This means that the development studies for the FPGA-based system have not been completed, and the system still requires effort for realization and implementation in the current situation. In the context of future studies, it is planned to synthesize and implement the FPGA-based design, and repeat the tests on the physical hardware.

Lastly, it is recommended to evaluate the use of a hybrid system, which integrates the software programmability of a processor with the hardware programmability of an FPGA, enabling key analytics and hardware acceleration on a single device, having the advantages of both systems.



7. REFERENCES

- [1] **Url-1** <https://en.wikipedia.org/wiki/Diagnostic_program>, date retrieved 22.03.2020.
- [2] **M. Soleimani, F. Campean and D. Neagu.** (2021). Diagnostics and prognostics for complex systems: A review of methods and challenges. *Quality and Reliability Engineering International*. 37(8): 3746-3778, 2021 John Wiley & Sons Ltd.
- [3] **C. Hua Lin, K. Chen and Y. Lee.** (2019). Design and Implementation of a Diagnostic System for Lithium Battery Module based on Sinusoidal Loading Technique. In *10th International Conference on Power Electronics and ECCE Asia (ICPE 2019 - ECCE Asia)* (pp. 1898-1903). IEEE.
- [4] **T. I. Buldakova, A. V. Lantsberg and S. I. Suyatinov.** (2019). Multi-Agent Architecture for Medical Diagnostic Systems. In *1st International Conference on Control Systems, Mathematical Modelling, Automation and Energy Efficiency (SUMMA)* (pp. 344-348). IEEE.
- [5] **A. M. Boronakhin, D. Y. Larionov, L. N. Podgornaya and R. V. Shalymov.** (2020). The Use of Embedded Diagnostic Systems for Ensuring Safe and Energy-efficient Driving of a Locomotive. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (pp. 1165-1170). IEEE.
- [6] **Y. I. Zharkov, Y. G. Semenov, D. V. Kolosov and J. Otýpka.** (2015). The principals of operation of the automated current collection diagnostic system for electrified railways. In *2015 16th International Scientific Conference on Electric Power Engineering (EPE)* (pp. 760-763). IEEE.
- [7] **O. Mareček.** (2016). Monitoring and diagnostic system of power plant electrical equipment. In *2016 Conference on Diagnostics in Electrical Engineering (Diagnostika)* (pp. 1-4). IEEE.
- [8] **A. Shi-qi, Q. An-ning and Z. Yu-wei.** (2011). Design and realization of SPI interface in lithiumion battery voltage measuring system. In *2011 6th International Conference on Computer Science & Education (ICCSE)* (pp. 83-87). IEEE.
- [9] **S. Kim, D. Kim and D. Kim.** (2018). Event log analysis software design for naval combat system using smart platform. In *2018 International Conference on Electronics, Information, and Communication (ICEIC)* (pp. 1-2). IEEE.
- [10] **C. K. Fong and E.M.-K. Lai.** (1994). AILAN: a local area network diagnostic expert system. In *Proceedings of International Conference on Expert Systems for Development* (pp. 238-243). IEEE.

- [11] **P. R. Burje, K. J. Karande and A. B. Jagadale.** (2014). Embedded on-board diagnostics system using CAN network. In *2014 IEEE Global Conference on Wireless Computing & Networking (GCWCN)* (pp. 31-35). IEEE.
- [12] **Url-2** < <https://www.embedded.com/a-lightweight-logger-for-c/>>, date retrieved 14.03.2020.
- [13] **Url-3** < https://en.wikipedia.org/wiki/Read-only_memory>, date retrieved 04.12.2021.
- [14] **Url-4** < <https://quicklearncomputer.com/ram-and-rom-advantages-disadvantages-characteristics-features/#t-1628311611921>>, date retrieved 04.12.2021.
- [15] **Url-5** < <https://www.seeedstudio.com/blog/2019/11/22/spi-introduction-to-serial-peripheral-interface/>>, date retrieved 04.12.2021.
- [16] **Micron Technology.** (2014). *Micron Serial NOR Flash Memory MT25QL128ABA*, CCMTD-1725822587-10223 mt25q_qlhs_L_128_ABA_xxT.pdf - Rev. J EN, May 18, 2014.
- [17] **Everspin Technologies.** (2018). *MR20H40 / MR25H40*, Revision 12.4, March 23, 2018.
- [18] **Texas Instruments Incorporated.** (2016). *TMS320F2837xD Dual-Core Delfino Microcontrollers Technical Reference Manual*, Literature Number: SPRUHM8F, December 2013–Revised December 2016.
- [19] **Xilinx.** (2018). *Vivado Design Suite Tutorial*, Xilinx UG937 (v2018.1), April 4, 2018.

8. CURRICULUM VITAE

Name Surname : Onur BEKAR

EDUCATION :

- **B.Sc.** : 2018, Yıldız Technical University, Faculty of Electrical and Electronics Engineering, Department of Electronics and Communication Engineering (GPA 3.65/4).

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2020 – Present: System Design Engineer at ASELSAN.
- 2018 – 2020: Electronic Design Engineer at ASELSAN.
- 2018: Second-Highest-Ranking Graduate of Department Award, Yıldız Technical University.
- 2017 – 2017: Microsoft Systems Support Officer at KoçSistem.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Bekar O. and Gunes E. O.** 2022: Design of an Embedded Fault Diagnostic System. INTERNATIONAL GRADUATE RESEARCH SYMPOSIUM - IGRS'22, June 1-3, 2022 Istanbul, Turkey.

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Durgeci G., **Bekar O.** and Unverdi N.O. “Kablolu ve Kablosuz Optik Haberleşme Sistemlerinde Performans Analizi”, Poster presentation, *FOTONİK 2018 20. Ulusal Optik, Elektro-Optik ve Fotonik Çalıştayı*, Sep. 2018, Ankara. (p. 51).