T.R.

GEBZE TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

GENERIC FRAMEWORK DESIGN FOR AN IOT-ORIENTED DATA ACQUISITION SYSTEM

ERDEM ÇETİN

A THESIS SUBMITTED FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF ELECTRONICS ENGINEERING

GEBZE

2022

T.R.

GEBZE TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

GENERIC FRAMEWORK DESIGN FOR AN IOT-ORIENTED DATA ACQUISITION SYSTEM

ERDEM ÇETİN

A THESIS SUBMITTED FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF ELECTRONICS ENGINEERING

THESIS SUPERVISOR

ASSIST. PROF. DR. ÖNDER ŞUVAK

GEBZE

2022

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

IOT AĞLARINDAKİ VERİ TOPLAMA SİSTEMİ İÇİN TASARLANMIŞ GENEL BİR ANAYAPI

ERDEM ÇETİN

YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DR. ÖĞR. ÜYESI ÖNDER ŞUVAK

GEBZE
2022

GEBZE TEKNİK ÜNİVERSİTESİ	**YÜKSEK LİSANS JÜRİ ONAY FORMU**

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 16/06/2022 tarih ve 2022/29 sayılı kararıyla oluşturulan jüri tarafından 20/06/2022 tarihinde tez savunma sınavı yapılan ERDEM ÇETİN'in tez çalışması Elektronik Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (TEZ DANIŞMANI)	: Dr. Öğr. Üyesi Önder ŞUVAK
ÜYE	: Prof. Dr. Koray KAYABOL
ÜYE	: Prof. Dr. Alper DEMİR

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ...…/……/…… tarih ve ……/…..... sayılı kararı.

İMZA/MÜHÜR

SUMMARY

IoT (Internet of Things) networks for sensor data acquisition applications require facilities such as master/slave interchangeability and reconfigurability. In this thesis work, we propose a hardware/software codesigned framework possessing the stated features and several more in addition. With the stated framework, virtually all IoT devices can be reconfigured with ease via predefined reconfiguration schemes. There must be a library for each such device utilizable by the PC application that comes with the proposed package; consequently master/slave roles in operation can be assigned for the managed devices to participate in a data acquisition application. In contemporary IoT networks, devices are accessed through various communication interfaces such as Wifi/BT, Ethernet, PCIe, LVDS, CAN, UART, SPI, I2C, etc. The proposed framework is virtually capable of being extended to capture boards with many of the stated interfaces accommodating processors such as Raspberry Pi, Orange Pi, NXP/Freescale, STM, Sitara, Intel, Xilinx, and Microchip; and we claim to be able to configure and initialize each of the IoT devices in a network, managed by our framework, to have master or slave behavior through specifically programmed files stored in their external flash memories.

Key Words: Internet of Things, FPGA, Microcontroller, Dynamical Reconfiguration, Bootloader, Data Acquisition.

ÖZET

Sensör veri alıcısı odaklı uygulamalarda IoT (Nesnelerin İnterneti) tabanlı ağlar, efendi/köle olarak görev yapan aygıtların rol değiştirebilmelerinden ve tekrar ayarlanabilme özelliklerinden faydalanmak durumunda olabilirler. Bu tez çalışmasında, belirtilen özellikleri ve daha fazlasını da içeren, donanım/yazılım birlikteliği temeline göre tasarlanmış bir ana yapı sunulmaktadır. Belirtilen ana yapı ile, önceden tanımlı konfigürasyon seçimlerine göre neredeyse tüm IoT aygıtlarının ilgili şekilde ayarlanması mümkün olabilmektedir. Önerilen paket ile birlikte gelen PC uygulaması tarafından kullanılabilen her bir aygıt için bir kütüphane bulunmalıdır; böylece işletim sırasında efendi/köle rollerine bürünmesi istenen aygıtların veri alıcısı odaklı uygulamalarda kullanılmak üzere ayarlanabilmesi sağlanmaktadır. Çağımızdaki IoT ağlarında, aygıtlara Wifi/BT, Ethernet, PCIe, LVDS, CAN, UART, SPI, I2C, vs. gibi iletişim arayüzleri aracılığıyla erişilebilmektedir. Sunulan ana yapı, Raspberry Pi, Orange Pi, NXP/Freescale, STM, Sitara, Intel, Xilinx ve Microchip gibi işlemcileri üzerinde bulunduran ve bahsi geçen arayüzlerin birçoğunu da özellik olarak sunan kartları kullanıma alabilecek şekilde genişletilebilme kabiliyetine sahiptir; böyle IoT ağındaki aygıtların harici taşınır belleklerinde saklanacak önceden programlanmış dosyalar aracılığıyla efendi veya köle olacak şekilde ayarlanabilmesini bahsedilen ana yapının sağlayabileceği belirtilmektedir.

Anahtar Kelimeler: İnternet Tabanlı Bileşenler, FPGA, Mikrodenetleyici, Dinamik Programlama, Önyükleyici, Veri Aktarımı.

ACKNOWLEDGEMENTS

I would like to express my special thanks of gratitude to my thesis advisor Asst. Prof. Önder ŞUVAK, who gave me the golden opportunity to do this wonderful project on the topic Generic Framework Design For an IoT Oriented Data Acquisition System. He shared his valuable knowledge and experience from the beginning to the end of my graduate education. He not only shared his profound scientific knowledge with me but also taught me great lessons in life. His support, suggestions, and encouragement gave me the drive and will to complete this work. I came to know about so many new things I am really thankful to him.

I would like to thank everyone and especially my friends for helping me to overcome the difficulties I faced during my studies at Gebze Technical University.

Finally, I would like to thank my father Ahmet Refik ÇETİN, my mother Emine ÇETİN, my wife Feyza ÇETİN and my daughter Miray ÇETİN for valuable supports in this challenging process. I am grateful to my family for their love and support.

TABLE of CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

Abbreviations and Acronyms		**Explanations**
FPGA	:	Field Programmable Gate Array
GPIO	:	General Purpose Input Output
IC	:	Integrated Circuit
ICAP	:	Internal Configuration Access Port
I2C	:	Inter IC Bus
KB	:	KiloByte
MB	:	MegaByte
MCU	:	Microcontroller Unit
QSPI	:	Quad Serial Peripheral Interface
SPI	:	Serial Peripheral Interface
UART	:	Universal Asynchronous Receiver Transmitter
WBSTAR	:	Warm Boot Start Address

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Nowadays, embedded systems can contain different processor types to accomplish functions defined in each sector. These processors could be general purpose processors, FPGAs or ASICs. These types come with different advantages and disadvantages while doing the system design. Designer should consider different requirement sets defined as functional, performance, physical, data interface, electrical interface, cooling interface, safety, reliability, maintainability, testability, supportability, security and environmental. Because in radar applications, automotive or avionics there are very strict differences with time critical, safety critical, mission critical or flight critical functions and processing systems could have different strengths, weaknesses to be used on those functions. To exemplify this, parallel computing could achieve better performance for time critical tasks and system could be obliged to use FPGA series[1]. If high computational need of different tasks with lot of resources in a computer cluster network, system could use Intel Xeon or Stratix processor series with high floating point processing capability[2].

1.1. Scope of the Thesis

In this work, it is aimed to examine a general framework to distribute tasks and workforce between IoT devices dynamically for resource management. For this project, different type of processors will be used to imitate the behavior of different IoT devices. Thus 7 series Xilinx FPGA and STM series Microcontroller has been selected since all of them can be used effectively for dynamic reconfiguration purposes nowadays. Also there will be a sensor block to sense the motion and data will be read in SPI communication interface. In the first step, system will boot with initial configuration files with specific behaviors of applications. There will be a master, slave relationship between IoT devices and one of them will send sensor data to the main controller for data acquisition purposes. In the second step, when user triggers different assignment of the tasks, system will send commands via communication interface and both IoT devices' tasks will be switched on runtime. Then master module will be changed back to slave module properties and remaining module will start transferring sensor block data within short time interval on runtime.

In order to reflect a system which includes vast amounts of accelerated data processing via parallel computing while retaining time-critical properties, I have chosen FPGA as my first IoT system.

In the next generation FPGA systems, there were partial and dynamic reconfiguration concepts introduced[3]. FPGAs can be effectively acquire changes dynamically in execution cycles which means while it is operational. For our purpose, I have chosen Xilinx Artix-7 FPGA with Cmod-A7 development board with external QSPI Flash memory of 4 MB size. To distribute tasks and changing the behavior of FPGA, I have used multiboot property of the FPGA via ICAPE2 primitive. Thus I was able to design an IoT device with run-time full reconfigurable embedded platform. Basically, Artix-7 FPGA Multiboot feature enables switching between two or more configuration files, during normal operation, from an external SPI Flash memory[4]. When an error or an interruption is detected during the Multiboot configuration process, the FPGA triggers fallback feature that ensures the configuration with a golden “safe” image file. That’s why I have created two image files both will be saved to the QSPI Flash memory before start-up. Then system could be able to jump between two image files to reconfigure FPGA on the fly[5]-[6].

For the second IoT system, I have selected STM32 series MCU to reflect low-power, low cost easily implemented systems. Since those kinds of processors execute tasks sequentially, it is not possible to maintain time critical functions via these processors easily. The major factor to choose those systems could be that development and life cycle costs are low comparing to FPGAs. Also it is easier and faster for developers to implement various kinds of communication interfaces with embedded C language and specific set of instructions. For my thesis work, it was possible to use internal bootloader and custom bootloader application with this specific microcontroller family of ST Microelectronics[7].

Mixture of different processor families, it is aimed to show interoperability and scalability while having a generic sustainable framework which does not have to change with newer devices connected to the system.

1.2. Motivation

Dynamical Reconfiguration of IoT Devices that has been done by other researchers can be summarized like this:

- Using reinforcement learning or cognitive frameworks to trigger reconfigurations to achieve failover schemes, fault recovery in runtime [9]-[12].
- Changing the data traffic to failed network equipments to continue running a specified function or graceful degradation of that function[13]-[14].

Differences and contributions of my thesis work from previous researches explained below:

- Master - Slave operations are assigned at each timeframe via PC Application to each IoT device. This shows the implementation's interchangeability feature. Dr.Wiswanath Karad's work proposes an high speed data acquisition framework which collects data from other devices which have dedicated firmware/functions without the capability of reconfiguration[15]. Our work surpasses that approach by assigning different tasks between devices which have MCU, FPGA, SOC architectures.
- There will be a library generated for each device and different device types can be connected to our main CPU/PC. This shows the implementation's easy adaptability capability[15]. Dr.Wiswanath Karad's work also has the capability of the different device types' connection. Our work has an upside of connection capability to all industry standard communication protocols such as Ethernet, USB, Wifi/BT etc.
- All different device types can be connected to main CPU/PC via USB hub. Instead of connecting, one STM32 and one Artix-7, we can connect two stm32 device or two Artix-7 devices. This shows the implementation's portability. Dr.Wiswanath Karad's work have IoT framework connected to Internet via Wifi for high speed data acquisition purposes[15]. Our work has an upside of connection capability to all industry standard communication protocols such as Ethernet, USB, Wifi/BT etc.

- Also we can change the master IoT device with another IoT device with the same functionality as in our implementation of reading ADXL345 data. That means if master IoT device malfunctions, we can continue the sensor reading function via another IoT device with same functionality. This shows implementation's high availability. K.S.Rekha's proposed work have a solution for resolving unstable data traffics in the Network which have dedicated devices connected to Wireless Sensor Network[16]. Our generic framework will have more than only Wifi connection with the rest of the features mentioned in other bullets.
- This framework can be applied to different communication interfaces of IoT devices such as Wifi/BT, Ethernet, Pcie, Uart, SPI, CAN, I2c etc. This shows the generic framework implementation's rich heterogeneous capabilities. We can plug each IoT device with this application via all those mentioned communication interfaces. Asad Javed's work has proposed a IoT Network to come up with an architecture which has heterogeneity, interoperability, discovery and scalability[17]. Apart from these features, our work has high availability, portability, interchangeability and upgradeability features.
- We can assign two master applications simultaneously to two or more different IoT devices connected to our main CPU/PC. This shows our implemetation's redundancy feature. Since if one IoT device is broken we cannot lose the function because other IoT device will still keep running. Conrado Pilotto's work has triple modular redundancy by using partial reconfiguration framework for FPGA's[8]. As an example, our proposed work can have redundancy in overall system via assigning master-slave role and tasks between each IoT device. Generic framework will have redundancy feature for all IoT devices connected to network by assigning the same function to different FPGA, MCU, SOC types.
- We can plug next generation of IoT equipments with improved version of the processors to usb hub of CPU/PC which runs our GUI App. We can pre-load master/slave programming files to those improved boards external flash memories. Then this shows our implementation's scalability feature. Asad Javed's work has proposed a IoT Network to come up with an architecture which has heterogeneity, interoperability, discovery and scalability[17]. Apart

from these features, our work has high availability, portability, interchangeability and upgradeability features.

- In this framework, we can use full resources of each IoT device FPGA and MCU type, since we are doing full reconfiguration via multiboot for FPGA or bootloader for STM32. This shows we can achieve high performance in each IoT device. Because at any time, there will be only one application running inside IoT devices. Furthermore, we are not running two application embedded codes at the same time in one IoT device. As an example, our framework have advantage compared to Damian Wanta's work only includes partial reconfiguration of FPGA's for specific applications[18].
- Within our CPU/PC App, device types and lists can be enlargened via adding different MCU types bootloader applications. Then we can connect devices differ from STM32 or Xilinx FPGA such as processors of ATMEGA, Texas Instruments, Arduino etc. This shows the upgradeability feature of generic framework. As an example, our framework have advantage compared to Amir Masoud Gharehbaghi's work as reconfigurable architecture can be applied to only FPGA's[19].

1.3. Thesis Structure

This thesis work contains background and related works for concepts of reconfigurable computing and architectures of both Xilinx-7 FPGA series and STM32 series in second chapter. Behavioral model of the generic framework have been explained in separate parts for IoT devices and central controller in third chapter. Implementation of the systems and generic framework solution explained in detail with the working system results, analysis have been stated in fourth chapter. In Chapter 5, conclusion and future work have been explained. Furthermore, I have explained two ongoing projects to reflect the improved version of the generic framework in IoT networks.

2. BACKGROUND AND RELATED WORK

Reconfigurable devices like FPGAs and MCUs are being used in many sectors such as automotive, avionics, military, scientific computing and enterprise computing. The main ideas behind those devices were to have high reliability, high availability, interoperability and scalability of the hardware infrastructure. Thus there are concepts such as static, dynamic reconfiguration of the FPGAs and internal bootloaders for the MCUs have been introduced and used rapidly with ever-growing industries defined above.

2.1. Concepts of Reconfigurable Computing

2.1.1. Architecture of Xilinx 7-Series FPGAs

In the FPGA architecture, it contains configurable logic blocks and all of these logic blocks are connected each other via internal connections inside the IC. There are also block RAMs and DSP blocks between configurable logic blocks as seen in figure below. All of the input output pins are located near to the configurable logic which enables user to write the code to use all I/Os and logic blocks for their application specific properties[20].

The well-known FPGA families in the world produced by Xilinx and Altera companies. Thus they produce also application specific FPGAs which could serve for different needs of the companies work in automotive, military, enterprise, space industries. Depending upon the design of the system architecture, FPGA's could include different number and type of logic elements, different custom block core for peripherals, hardened blocks such as DSPs or hardened ARM processors inside logic blocks via AXI interconnects[21]. To illustrate this, figure 2.1 demonstrates RAM/DSP blocks, processor subsystem, peripherals, PCIe/Memory controller's interconnection.

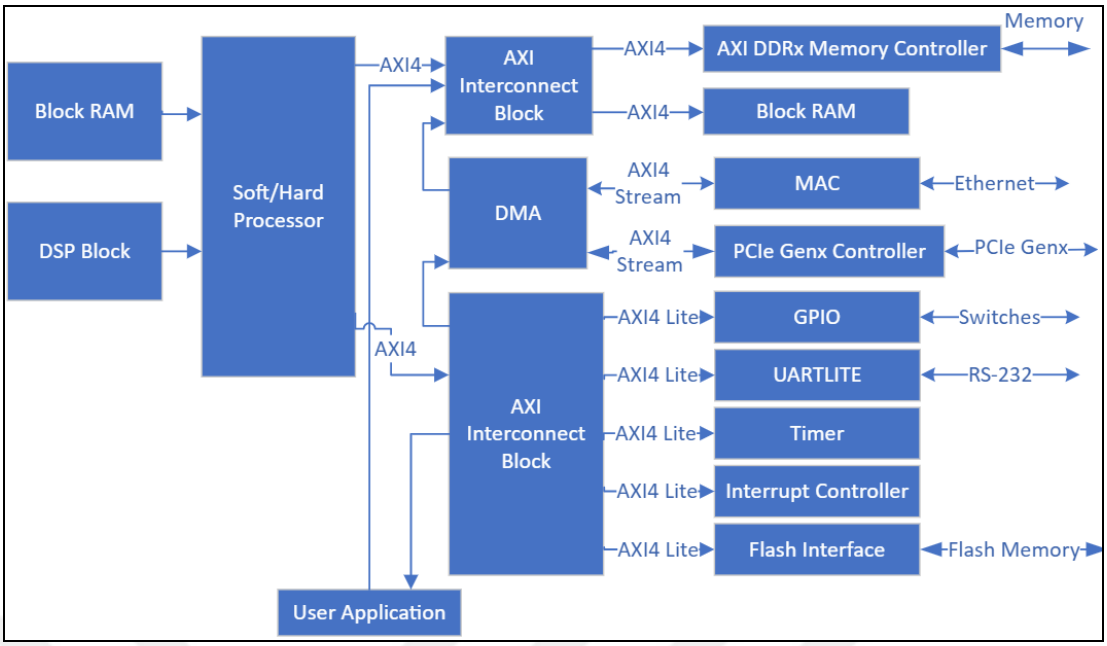

Figure 2.1: Reference fpga architecture.

Xilinx 7 Series FPGAs have high performance FPGA logic which includes 6-input look up table (LUT) as memory element. These smart units include 36 Kbyte block RAM for data buffering via integrated FIFO logic. Optimized high throughput with low power consumption has been acquired via these Xilinx FPGAs. There are 4 different FPGA families which are defined as Spartan-7, Artix-7, Kintex-7, Virtex-7 to meet the demanding requirements of high-tech industries. There are some advantages and drawbacks between different 7 series FPGA families as seen in the following table[22]. Therefore those processors should be selected via the application specific requirements of the customer.

Table 2.1: 7 Series fpga families' comparison.

Spartan-7 Family	Artix-7 Family	Kintex-7 Family	Virtex-7 Family
Optimized for low cost, lowest power, and high I/O performance.	Optimized for low power applications requiring serial transceivers and high DSP and logic throughput.	Optimized for best price-performance with a 2X improvement compared to previous generation	Optimized for highest system performance and capacity with a 2X improvement in system performance.
Available in low-cost, very small form-factor packaging for smallest PCB footprint	Provides the lowest total bill of materials cost for high-throughput, cost-sensitive applications.	With its own features, enables a new class of FPGAs.	Highest capability devices enabled by stacked silicon interconnect (SSI) technology.

In my thesis work, I have chosen Artix-7 35t FPGA which is part of 7 series FPGAs since it has its own flash memory of size 4 MB located on the development board. Also it could be fully configured on the fly with image files located in flash. Furthermore, the logic block resources was enough for my applications as it has 33280 logic cell number and it has 50% less power consumption compared to the previous generation[22].

2.1.1.1. Static and Dynamic Configuration

There are some configuration concepts in reconfigurable hardware. A reconfigurable device like FPGA can be configured in two ways; static and dynamic configurations. When the device is configured just after power-up, it is called static configuration. To reconfigure a running device statically, the user must stop the execution cycle and reconfigure it again. Dynamic configuration means changing the configuration while the device is running. The user does not have to reset the device or stop the execution to enable dynamical change of configuration image files[23].

Full reconfiguration of 7 series FPGAs is possible via multiboot property without interrupting execution cycles or reset power of the FPGA.

In the 7 series FPGAs, there are two features known as multiboot and fallback to makes it possible to update firmware inside logic blocks while system is

operational. For developers, this concept is also known as on-the-fly programming to dynamically change bitstream images. Also there is a protection mechanism inside fpga which is known as fallback feature. In the thesis work multiboot feature will be used to change application firmwares inside fpga dynamically. But if there is a corrupted file as application firmware, system will abort the process and load initial working bitstream to the FPGA. Therefore, system will continue running and prevent the unexpected errors to happen via corrupted firmware[4]. It is possible to quickly boot whole configuration dynamically via QSPI flash 4 MB memory which is shown below as SPI x4 Configuration interface[5].

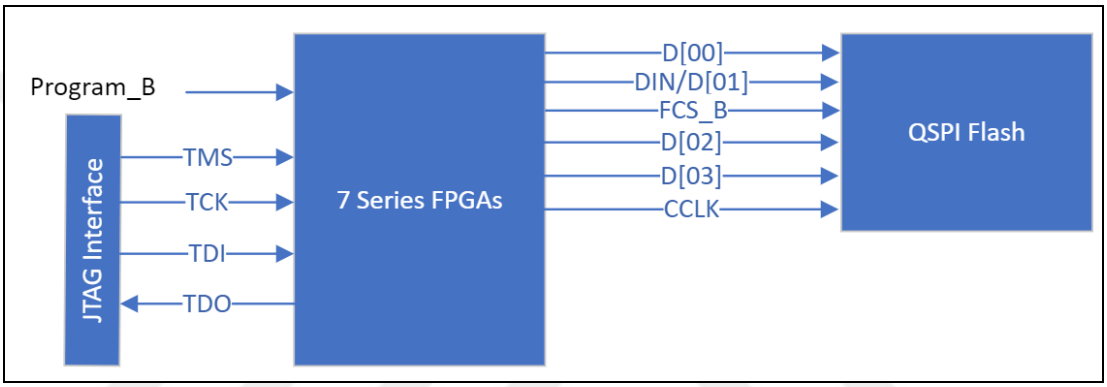

Figure 2.2: 7 Series fpga spix4 configuration interface.

Multiboot feature works in two ways:

Firstly, WBSTAR-warm boot address and internal PROGRAM-IPROG command could be embedded in the bitstream. if FPGA system acknowledges the internal PROGRAM command. System first starts from flash address 0 which contains initial bitstream which is also indentified as golden bitstream. The second firmware application is stored at flash address which is defined in the WBSTAR register to show next configuration address. Normally WBSTAR value is default value. The IPROG is automatically embedded in the bitstream when WBSTAR takes over another value then default. Configuration logic will execute initial-golden bitstream from flash address 0 at start. When system reads IPROG command then configuration logic will jump to the flash address specified in WBSTAR register to update the firmware to change the field inside FPGA. If there is an error condition to load the firmware inside FPGA then fallback mechanism will be enabled. Configuration logic pulls INIT_B and DONE to low state and clears the

configuration memory. Then system will convert to the initial golden image file and ignores the WBSTAR and IPROG commands in fallback process. By doing so, system will restart the configuration process to run initial golden image file which is in good condition[4].

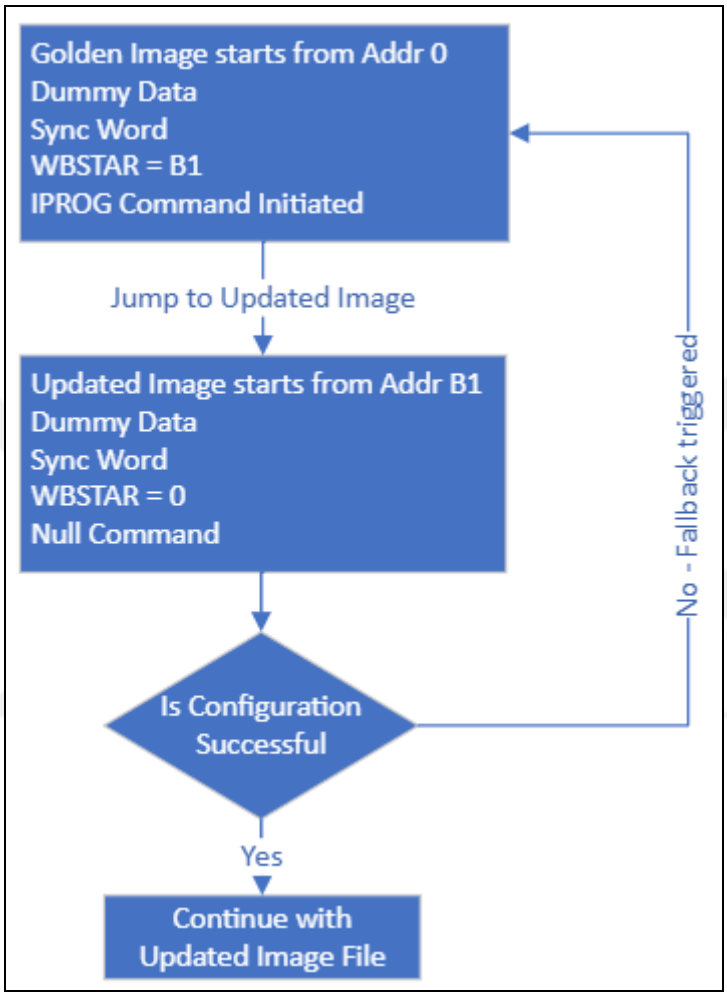

Figure 2.3: Multiboot fallback flash memory components and configuration steps.

Secondly, in 7 series Xilinx FPGAs, there is also another way of initiating multiboot property via ICAPE2 primitive. This primitive enables users to trigger multiboot via written VHDL code in VIVADO programme. In this thesis work, I have used ICAPE2 primitive to control multiboot property via communication interface and handle the reconfiguration mechanism independently to the embedded bitstream values[24].

ICAPE2 primitive is a design element to reach internal configuration logic of the FPGA. By using the primitive, it is possible to send data and commands to

configuration logic. Developers can use this primitive by just instantiation in VHDL code rather than choosing from IP catalog, inference or macro support. The port definitions of the ICAPE2 primitive is defined in the table below.

Table 2.2: Port descriptions for icape-2 primitive.

Port	Direction	Width	Function
CLK	Input	1	Clock Input
CSIB	Input	1	Active Low ICAP Enable
I<31:0>	Input	32	Configuration data input bus
O<31:0>	Output	32	Configuration data output bus
RDWRB	Input	1	Read/Write Select Input

In order to accomplish multiboot feature, the user determines the start address to the updated firmware located in flash memory and sends the synchronization word first. Then user writes the updated firmware's flash address value to WBSTAR register. Then user initiates this reconfiguration process by sending the IPROG command by ICAPE2 primitive[5].

When configuration logic receives IPROG command, FPGA resets itself but lets the reconfiguration logic to run. It controlls INIT_B and DONE pins to become low state. Then FPGA clears all configuration memory and makes INIT_B to high state. Then system will be configured by updated bitstream address which is located in WBSTAR register[5]. Also configuration mode decides which pins will be controlled via WBSTAR. It can be seen in below figure for all configuration modes that is possible. In this thesis work, I will be using SPI interface to field upgrade from 4 MB Flash memory inside Cmod-A7 board.

Configuration Mode	Pins Controlled by WBSTAR
Master Serial	RS[1:0]
Master SPI	START_ADDR[23:0] is sent to SPI device serially
Master BPI	RS[1:0], A[28:00]
Master SelectMAP	RS[1:0]
JTAG	RS[1:0]
Slave SelectMAP	RS[1:0]
Slave Serial	RS[1:0]

Figure 2.4: WBSTAR controlls for configuration mode.

2.1.2. Architecture of STM32F103 Series MCUs

The STM32F103 is a microcontroller unit with ARM Cortex-M3 32-bit RISC core which uses 72 MHz frequency. MCU contains flash memory up to 128 Kbytes of size and SRAM up to 20 Kbytes of size. It has general purpose input and outputs, two ADCs, timers plus PWM timer and several peripherals which can be used as communication interfaces known as I2C,SPI,UART,USB and CAN[25]. It can be seen in below table in STM32F103Cx columns.

Table 2.3: STM32F103xx medium-density device features and peripheral counts.

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	General Purpose	3		3		3		3	
	Advanced-Control	1		1		1		1	
Communication	SPI	1		2		2		2	
	I2C	1		2		2		2	
	USART	2		3		3		3	
	USB	1		1		1		1	
	CAN	1		1		1		1	
GPIOS		26		37		51		80	
12 bit synchronized ADC Number of Channels		2 10 Channels		2 10 Channels		2 16 Channels		2 16 Channels	

2.1.2.1. Bootloader of STM32 Series

STM32F103 series have different boot modes and those modes could be selected via boot pins on the board. Through the boot pins, it is possible to adjust the STM32 configuration file to be used from User Flash, System Memory and embedded SRAM[7].

STM32 internal bootloader is embedded inside System Memory and it can be used to program flash memory via UART interface[25].

Table 2.4: STM32f10xx configuration in System memory boot mode.

Bootloader Feature/Peripheral	**State**	**Description**
Clock Source	HIS Enabled	The system clock is equal to 24 MHz using PLL.
RAM	-	512 bytes starting from address 0x20000000 are used by the bootloader firmware.
System Memory	-	2 Kbytes starting from address 0x1FFFF000 contain the bootloader firmware.
IWDG	-	The independent watchdog prescaler is configured to its maximum value and is periodically refreshed to prevent reset.
USART1	Enabled	Once initialized, the USART1 configuration set.
USART1_RX pin	Input	PA10 pin
USART1_TX pin	Output push-pull	PA9 pin
SysTick timer	Enabled	Automatically detects the serial baud rate from host.

2.2. Software Tools for Programming of the Systems

2.2.1. Software Tool for Xilinx FPGAs

I have used VIVADO 2020 for programming Artix-7 FPGA with two image files. I have selected boot from QSPI Flash and programmed two image files to the QSPI flash to jump from golden file to the updated file.

2.2.2. Software Tool for STM32 MCUs

I have used STM32CubeIDE tool for writing and compiling the application codes. Also I have used STM32CubeProgrammer to load the bootloader firmware and two different application firmwares to the predefined Flash locations separately.

3. BEHAVIORAL MODEL OF THE GENERIC FRAMEWORK

Generic framework follows the flowchart described below.

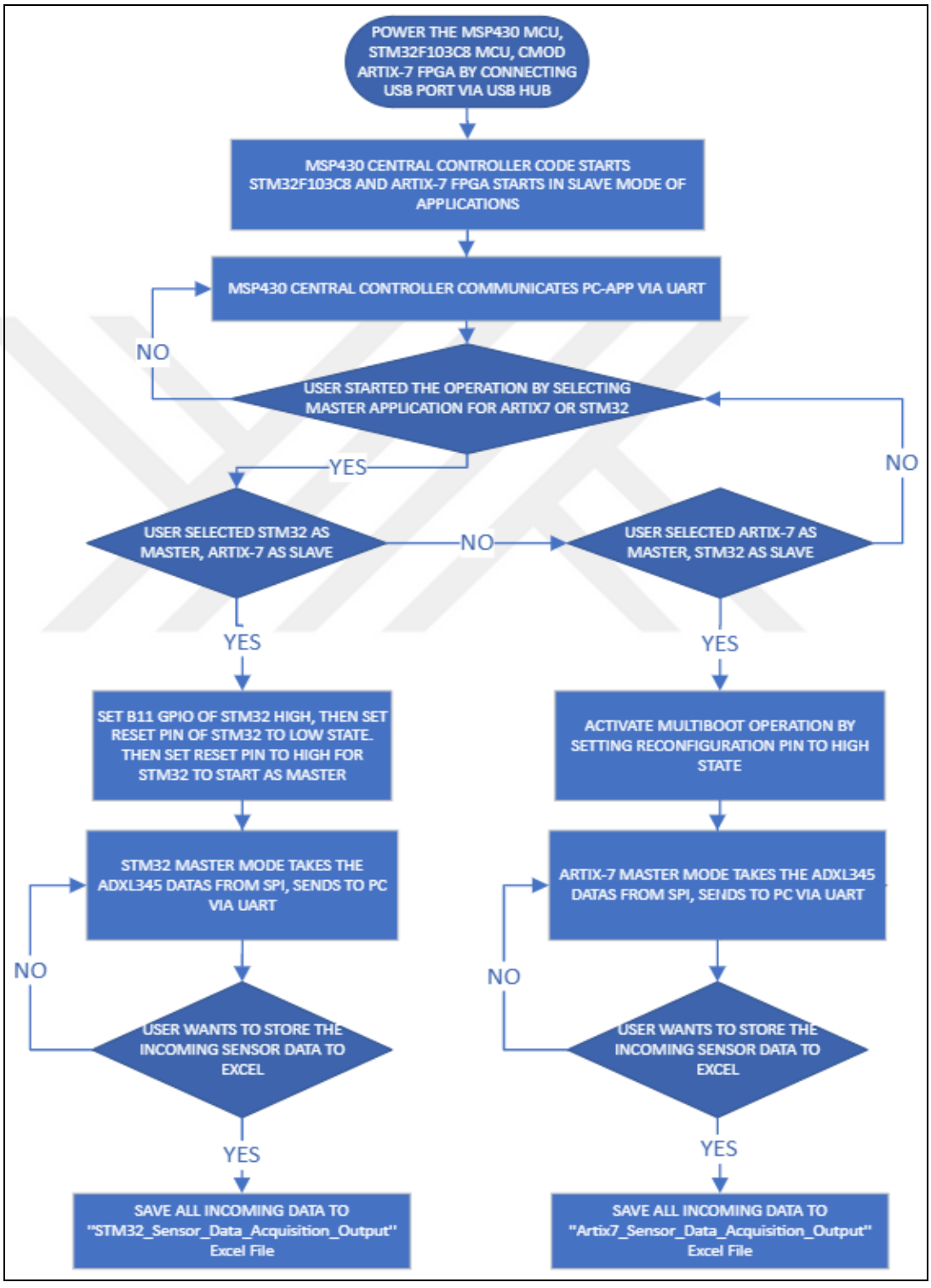

Figure 3.1: Flowchart of the generic framework implementation.

3.1. Data Acquisition System GUI Model

PC Application Software follows the below sequence after opened by user:

- It sends the user input for master – slave selection of STM32 device and Artix-7 device from GUI via PC UART connection.
- Monitors the incoming data from central controller unit.
- By users' activation, GUI will log the incoming data of sensor to the excel file.

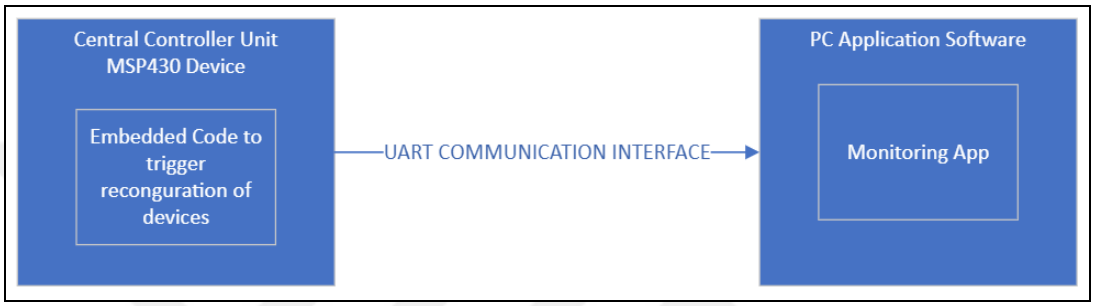

Figure 3.2: Relationship between central controller unit and gui.

3.2. Central Controller Model

MSP430 Central Controller Device follows the sequence below after power-up:

- It gets the user input for master – slave selection of STM32 device and Artix-7 device from GUI via PC UART connection.
- According to the user's selection input, system determines the functionality of the devices and uses appropriate triggers to handle bootloader mechanism for Artix-7 device and STM32 device as seen in figure 3.3.
- If the Artix-7 board is in master mode, MSP430 board gets the SPI interfaced ADXL345 data from UART interface.
- If the STM32 board is in master mode, MSP430 board gets the SPI interfaced ADXL345 data from UART interface.
- Then MSP430 board continuously sends the incoming data from the sensor to the PC via UART interface connection for data acquisition purposes.

- To determine the master modes of each device, MSP430 will send the sensor data with unique identification number which refers to the IoT device type. Thus it will be possible to log the data with the IoT device type.

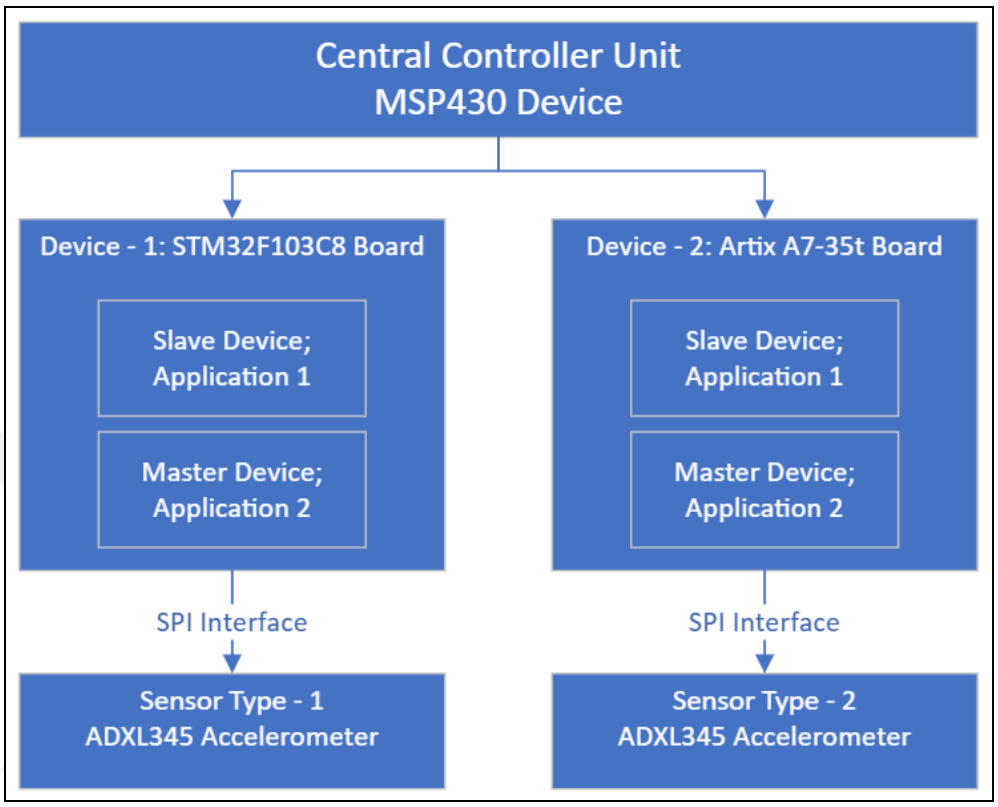

Figure 3.3: Relationship between msp430 central controller with iot devices.

3.3. IoT Device-1 Model with Artix-7 FPGA

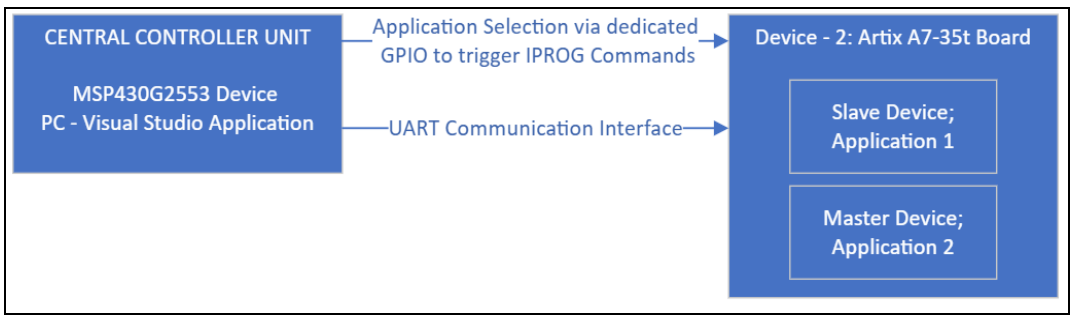

Figure 3.4: IoT Device 1 Model.

These properties are valid for master and slave modes of Artix-7 FPGA Board:

- Two different application image files have been compiled with VIVADO 2021.1 and those application image files have been saved in QSPI Flash memory.
- IPROG command will be used to trigger application flash memory address jump to run FPGA with updated image file.
- Application 1 will run as slave mode of the FPGA.
- Application 2 will run as master mode of FPGA to use SPI interface to communicate with ADXL 345 accelerometer sensor.
- Multiboot operation will be triggered by MSP430 board via dedicated I/O as reconfiguration pin to start sending the critical DataStream of ICAPE2.

3.3.1. IoT Device-1 Application-1 Model

In CMOD-A7 board, application-1 corresponds to the slave mode application where system do not access to the ADXL345 sensor datas and changes the brightness and color on the RGB in given time intervals and system will run with clock speed of 100 Mhz.

Artix-7 FPGA board follows the sequence below after power-up:

- System starts by changing the brightness and color of RGB led on CMOD A7 board by its own counter.
- System will continuously monitors the reconfiguration pin's state with a process inside VHDL module with 100 Mhz clock's each rising edge.
- If system recognizes the high state at reconfiguration pin, it will execute the reconfiguration mechanism. Then fpga will run with updated image file which contains master mode operation to get ADXL345 sensor data. Otherwise, fpga will continue with initial slave mode configuration.

3.3.2. IoT Device-1 Application-2 Model

In CMOD-A7 board, application-2 corresponds to the master mode application where system accesses to the ADXL345 sensor data.

Artix-7 FPGA board follows the sequence below after triggered reconfiguration:

- System starts running with updated image file which starts from flash memory address 0x002FFFFF.
- System reads the values in x axis, y axis and z axis and acknowledges the gravity acceleration. Since accelerometers should detect gravity acceleration as 1 g value even in steady motion.
- If system detects more than the threshold for gravity acceleration in z axis, it turns on the led and it turns led off otherwise.
- Then system sends these acquired sensor values to the MSP430 board via UART interface for data acquisition purposes.

3.4. IoT Device-2 Model with STM32F103C8

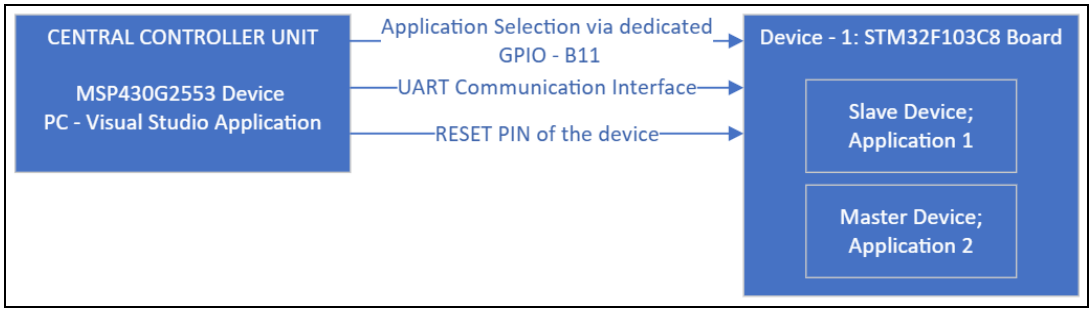

Figure 3.5: IoT Device-2 Model.

These properties are valid for master and slave modes of STM32 Board:

- Three different application image files have been compiled with STM32CudeIDE and those application image files have been saved in 64 Kbytes of Flash memory located on STM32 board.
- Those application image files are bootloader custom image, slave mode operation image file and master mode operation image file.

- After power-up or reset, system starts running from flash address 0x800000 and initially controls GPIO pin B11. This dedicated GPIO pin will be controlled by booting application.

  Firstly, if B11 is in low state system will jump to flash address 0x8005000 to continue working in slave mode namely application-1.

  Secondly, if B11 is in high state system will jump to flash address 0x800A800 to continue working in master mode namely application-2.
- Through flash linker file, system runs from flash address 0x8005000 or 0x800A800 within bootloader custom firmware.
- Reconfiguration will be triggered by MSP430 board via dedicated GPIO which is B11 pin and Reset pin as reconfiguration pins.

  Firstly, MSP430 board changes the B11 pin to make device ready for reconfiguration. This is done after user selects the master-slave choice for STM32 board from PC GUI Application.

  Secondly, MSP430 board changes the Reset pin from low state to high state for the system to enter into reset condition. Then MSP430 board releases Reset pin to low state for bootloader custom application to start.

3.4.1. IoT Device-2 Application-1 Model

In STM32f103C8 board, application-1 corresponds to the slave mode application where system do not access to the sensor data.

STM32F103C8 board follows the sequence below after bootloader executed for application-1:

- STM32 board will start its own counters for specific time delay.
- Then STM32 board will toggle the led on and off in given time intervals.

3.4.2. IoT Device-2 Application-2 Model

In STM32f103C8 board, application-2 corresponds to the master mode application where system accesses to the ADXL345 sensor data.

STM32F103C8 board follows the sequence below after bootloader executed for application-2:

- System reads the values in x axis, y axis and z axis and acknowledges the gravity acceleration.
- Since accelerometers should detect gravity acceleration as 1 g value even in steady motion. Whenever system detects an acceleration value which is above 0.5 g in z-axis of ADXL345, it will turn on the led. Otherwise it will turn off the led on the STM32 board with acceleration values below 0.5 g in z-axis of ADXL345.
- Then system sends these acquired sensor values to the MSP430 board via UART interface for data acquisition purposes.

4. IMPLEMENTATION

In my thesis work, there are one UART interface between MSP430G2553 board and PC with micro USB cable via FTDI FT232RL USB to TTL converter. MSP430G2553 board's three discrete pin output of port 2 have been used for reconfiguration trigger for Cmod A7 Artix-7 module board and STM32f103C8 development board. There are SPI interface between SDA, SDO, SCL, CS pins for Cmod A7 Artix-7 module and ADXL345 sensor. Also similar interface established for STM32f103C8 board's dedicated SPI1 interface which are A4 pin as chip select, A5 pin as SPI clock, A6 pin as master in slave out signal, A7 pin as master out slave in signal. Also FTDI converter has been used to power the Cmod A7 Artix-7 module and STM32f103C8 board since USB output of the computer can supply up to 500 mA of current. This power supply was enough as a source because maximum current that can be drawn for STM32 board is around 150 mA and Artix-7 FPGA board draws around 55 mA even in master mode configuration firmware loaded. As a total I needed a power supply which has 5 Volts in power rail and can supply more than 205 mA of current for the implementation. For the MSP430G2553 board, I have used other USB 3.0 port and the need was around 330 uA/MHz for active mode as stated in datasheet. Maximum amount of current need could be 23,76 mA for 72 MHz of running clock inside of the chip via frequency division of PLLs. Then USB port of the PC was more than enough to supply enough power even the prepared central controller firmware running in MSP430G2553.

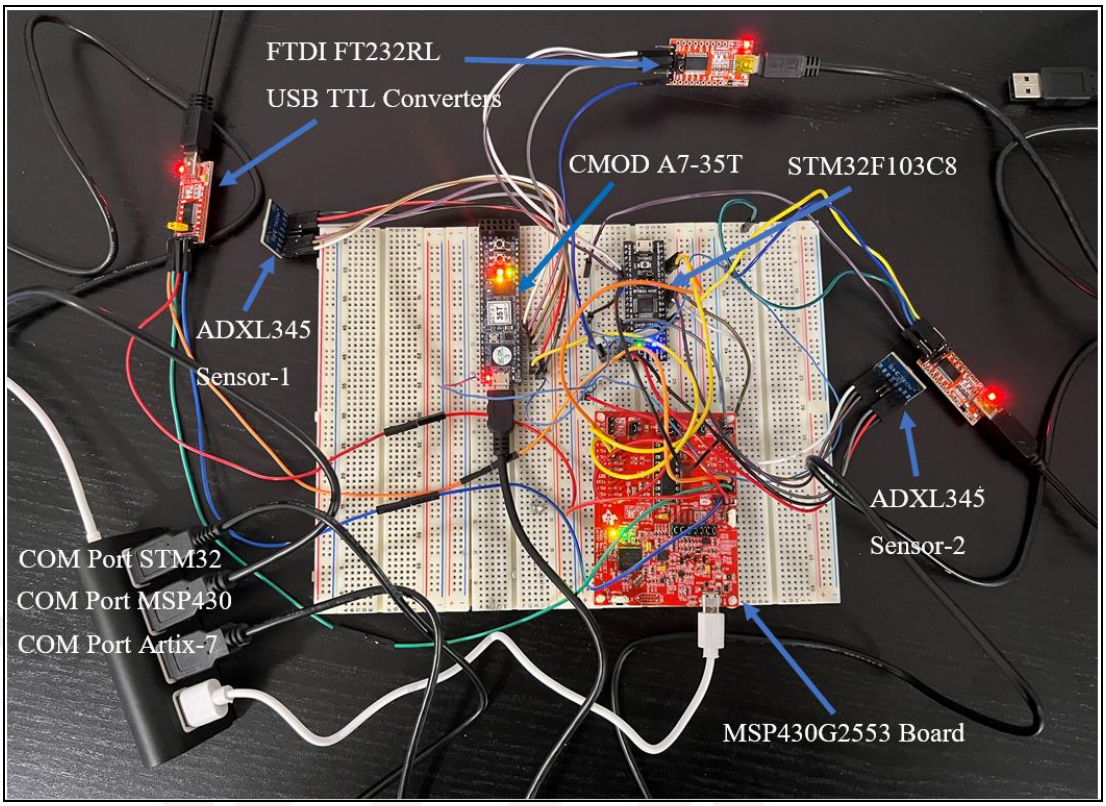

Figure 4.1: Implementation of the thesis project for data acquisition system.

4.1 GUI Implementation Results

For the central controller data acquisition purposes, I have developed a graphical user interface via Microsoft Visual Studio 2019 in C# programming language to communicate between MSP430 boards.

The purpose for the GUI is to configure the image files in the boards and capture the incoming sensor data via MSP430 and save those data to excel sheet in the name of the selected board which runs the master application. As it is seen in the below figure, there are three uart interface serial ports which are connected to the MSP430 board, Artix-7 board and STM32 board for data acquisition purposes. Also user can switch the master and slave modes for Artix-7 board and STM32 board to exchange the tasks between them.

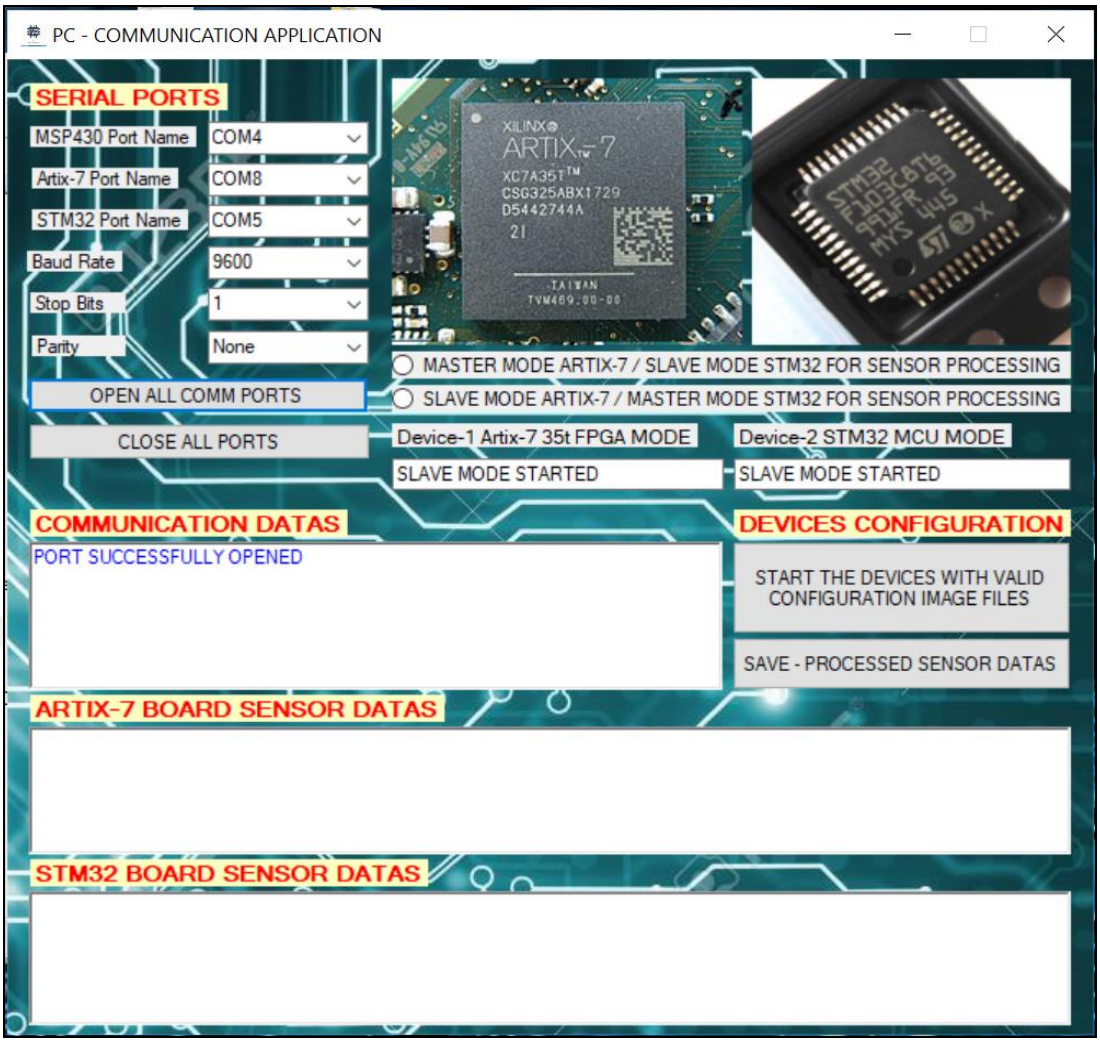

Figure 4.2: PC application gui for user to control master-slave functions of the devices.

When PC-APP executable object file opened, user must click open-port button to connect to the FTDI USB-TTL converter port of MSP430 board. After it opens the port there will be a message shows “PORT SUCCESSFULLY OPENED” and both IoT devices will start as slave mode for Artix-7 board and slave mode for STM32 board.

After user selects the Artix-7 board as master and STM-32 board as slave mode of operation and click the button which states “START THE DEVICES WITH VALID CONFIGURATION IMAGE FILES”. Then application communication log will show the exchanged datas between MSP430 and PC-App in the following figure.

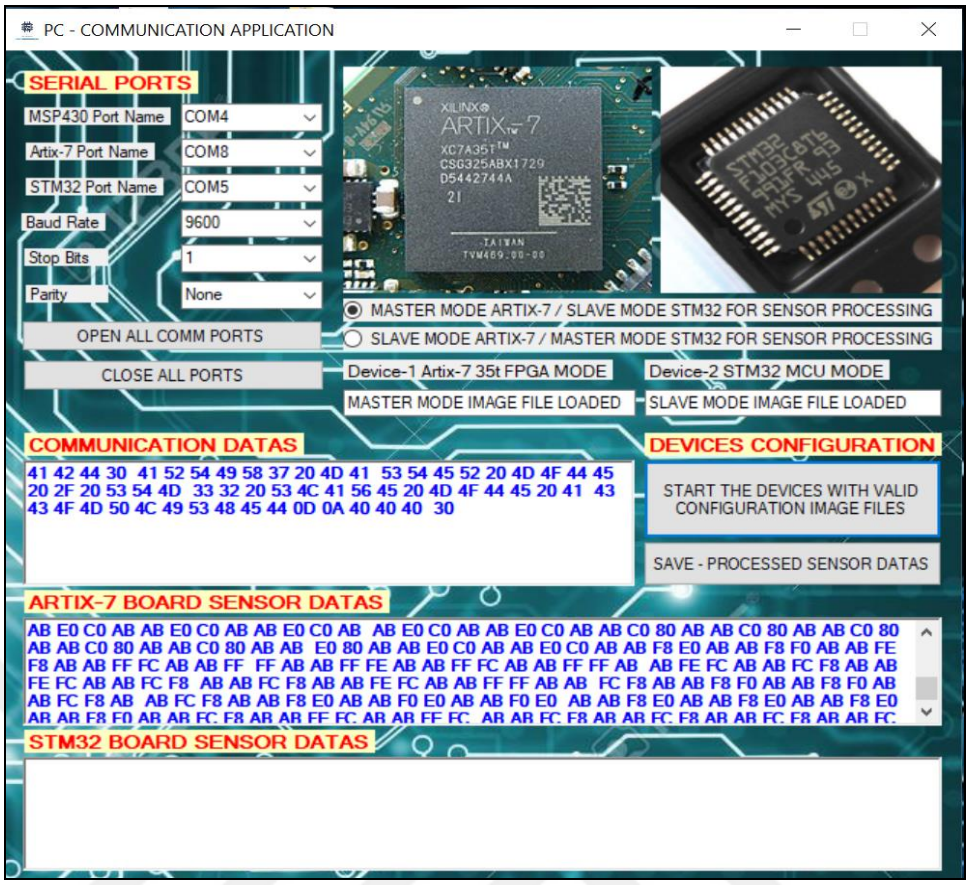

Figure 4.3: PC application when user start devices as artix-7 master, stm32 slave.

PC-Application will send the bytes 0x41, 0x42, 0x44, 0x30 to start operation and MSP430 board sends the rest of the datas to show Artix-7 master and STM32 slave mode operation started. In ASCII format, we can see the below screen for this specific transmission in RealTerm serial monitoring program in the figure below.

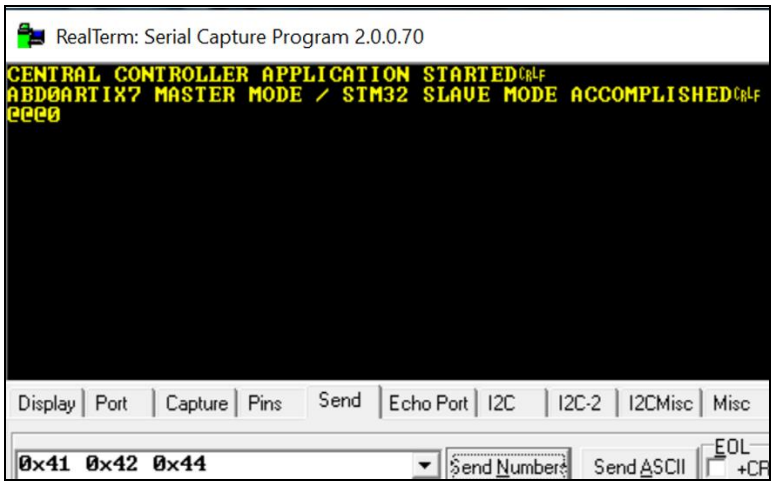

Figure 4.4: Realterm capture of the datastream as artix-7 master, stm32 slave.

After user selects the Artix-7 board as slave and STM-32 board as master mode of operation and click the button which states “START THE DEVICES WITH VALID CONFIGURATION IMAGE FILES”. Then application communication log will show the exchanged datas between MSP430 and PC-App in the following figure.

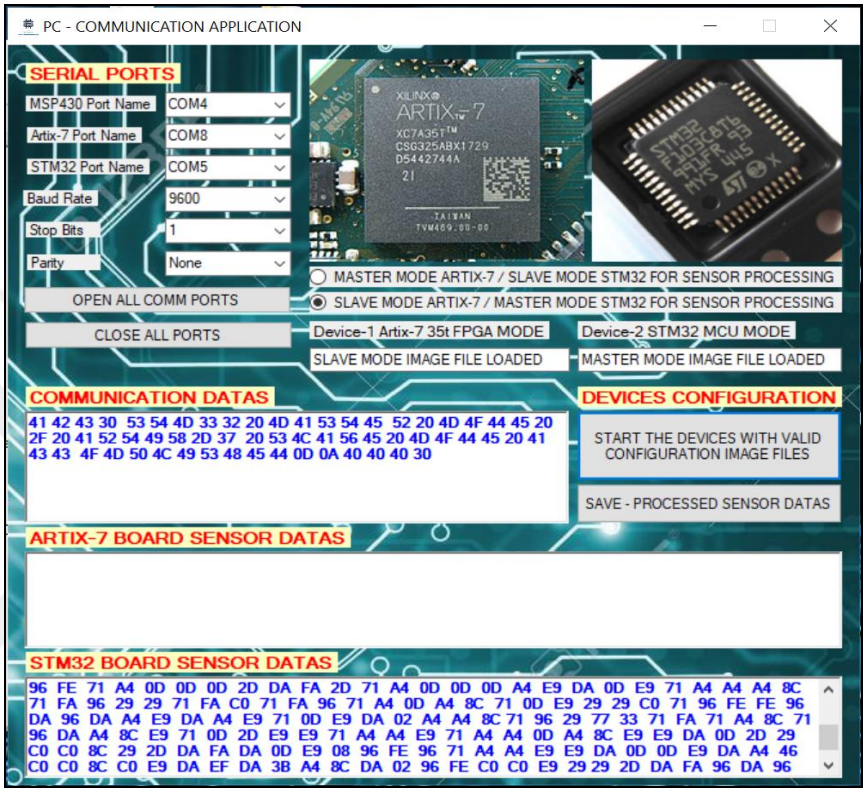

Figure 4.5: PC application when user start devices as artix-7 slave, stm32 master.

PC-Application will send the bytes 0x41, 0x42, 0x43, 0x30 to start operation and MSP430 board sends the rest of the datas to show Artix-7 slave and STM32 master mode operation started. In ASCII format, we can see the below screen for this specific transmission in RealTerm serial monitoring program in the figure below.

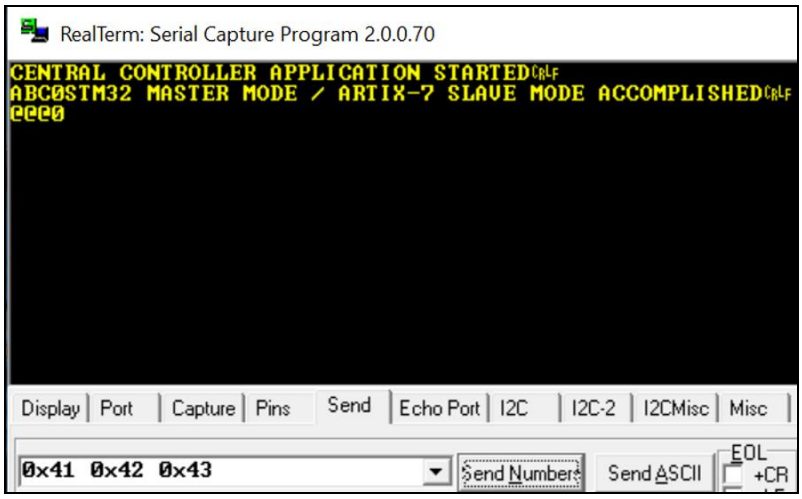

Figure 4.6: Realterm capture of the datastream as artix-7 slave, stm32 master.

As seen above, there are two data streams that can be sent from PC-Application.

- If PC-Application sends four bytes as 0x41 0x42 0x44 to MSP430 board, then MSP430 board reconfigures Artix-7 device as master and STM32 as slave mode of operation.
- If PC-Application sends four bytes as 0x41 0x42 0x43 to MSP430 board, then MSP430 board reconfigures Artix-7 device as slave and STM32 as master mode of operation.

Last but not least, PC-Application has final feature of recording the datas to excel file to save sensor data, if user selects “SAVE-PROCESSED SENSOR DATAS”.

- If STM32 MCU is in master mode, then Application creates “STM32_Sensor_Data_Acquisition_Output.xls” file under “.\\Documents” file section in the PC. In Visual Studio code, there is a Boolean which controls the user input choice of master-slave selection which is defined as “STM32master_ARTIX7slave” as seen in figure 16. This Boolean variable has been set to true for STM32 master mode of operation as seen in figure below. Then application will save the values to excel as seen in figure 4.7.

Figure 4.7: PC application when user selects to store sensor datas for stm32.

```
336         if (STM32master_ARTIX7slave)
337         {
338             Microsoft.Office.Interop.Excel.Application xlApp;
339             Microsoft.Office.Interop.Excel.Workbook xlWorkBook;
340             Microsoft.Office.Interop.Excel.Worksheet xlWorkSheet;
341             object misValue = System.Reflection.Missing.Value;
342
343             xlApp = new Microsoft.Office.Interop.Excel.Application();
344             xlWorkBook = xlApp.Workbooks.Add(misValue);
345
346             xlWorkSheet = (Microsoft.Office.Interop.Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
347             //StreamReader reader = new StreamReader("Array.txt");
348             String line = richTextBox3.Text;
349             string[] hexValuesSplit = line.Split(' ');
350
351             int storage = 1;
352             int alignment = 1;
353             //Code to save datas to excel file
354             foreach (String hex in hexValuesSplit)
355             {
356                 if (hex != ' '.ToString() & hex != "".ToString())
357                 {
358                     int value = Convert.ToInt32(hex, 16);
```

Figure 4.8: Code fragment which creates excel sheet for stm32 data.

1	STM32 Processed Sensor Datas																			
2	0	233	233	41	233	113	250	113	250	150	218	45	113	13	233	233	233	41	113	
3	218	2	250	250	218	150	218	13	13	70	140	218	13	70	218	150	41	218	164	
4	45	192	192	218	13	59	250	254	150	113	13	45	113	150	113	250	150	103	113	
5	164	140	218	13	13	233	192	218	164	164	164	13	233	113	13	164	140	140	87	
6	113	13	233	233	8	106	211	220	8	106	60	7	113	13	45	113	164	164	164	
7	140	119	111	111	119	101	101	106	69	218	164	70	113	150	45	233	140	113	164	
8	45	140	233	218	13	45	218	150	113	13	13	233	113	13	233	41	218	13	164	
9	140	113	164	211	45	41	218	250	218	2	13	45	233	113	150	45	192	218	150	
10	113	164	13	13	13	45	218	250	45	113	164	13	13	13	164	233	218	13	233	
11	164	164	164	140	113	250	150	41	41	113	250	192	113	250	150	113	164	13	164	
12	113	13	233	41	41	192	113	150	254	254	150	218	150	218	164	233	218	164	233	
13	13	233	218	2	164	164	140	113	150	41	119	51	113	250	113	164	140	113	150	
14	164	140	233	113	13	45	233	233	113	164	164	233	113	164	164	13	164	140	233	
15	218	13	45	41	192	192	140	41	45	218	250	218	13	233	8	150	254	150	113	
16	164	233	233	218	13	13	233	218	164	70	192	192	140	192	233	218	239	218	59	
17	140	218	2	150	254	192	192	233	41	41	45	218	250	150	218	150	192	113	250	
18	113	13	211	113	150	150	150	113	164	140	218	13	45	218	150	218	150	254	192	
19	250	250	192	233	140	218	2	150	150	150	150	113	45	233	140	113	13	45	233	
20	250	150	218	164	140	233	218	150	113	164	140	218	150	113	250	150	150	113	13	
21	218	150	254	192	41	233	218	13	13	45	218	150	150	218	164	13	233	192	113	

Figure 4.9: Saved excel sheet which contains stm32 sensor acquisition data.

- If Artix-7 FPGA is in master mode, then Application creates “Artix7_Sensor_Data_Acquisition_Output.xls” file under “.\\Documents” file section in the PC. In visual studio, else condition states the “STM32masterARTIX7slave” as false, thus ARTIX-7 has been selected by user as master mode of operation. Then application will get the values of Artix-7 communication port and stores these values to excel sheet as seen in Figure 4.10.

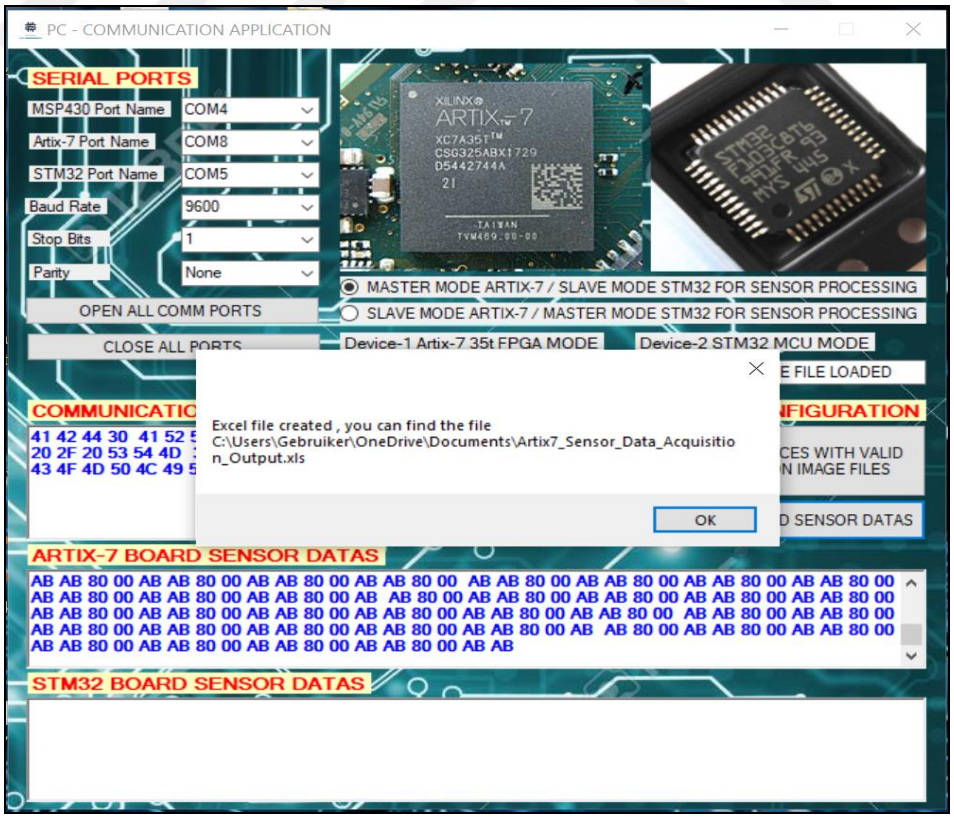

Figure 4.10: PC application when user selects to store sensor datas for artix-7.

```
385             else
386             {
387                 Microsoft.Office.Interop.Excel.Application xlApp;
388                 Microsoft.Office.Interop.Excel.Workbook xlWorkBook;
389                 Microsoft.Office.Interop.Excel.Worksheet xlWorkSheet;
390                 object misValue = System.Reflection.Missing.Value;
391
392                 xlApp = new Microsoft.Office.Interop.Excel.Application();
393                 xlWorkBook = xlApp.Workbooks.Add(misValue);
394
395                 xlWorkSheet = (Microsoft.Office.Interop.Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
396                 //StreamReader reader = new StreamReader("Array.txt");
397                 String line = richTextBox2.Text;
398                 string[] hexValuesSplit = line.Split(' ');
399
400                 int storage = 1;
401                 int alignment = 1;
402                 //Code to save datas to excel file
403                 foreach (String hex in hexValuesSplit)
404                 {
405                     if (hex != ' '.ToString() & hex != "".ToString())
406                     {
```

Figure 4.11: Code fragment which creates excel sheet for artix-7 data.

A1 — Artix-7 Processed Sensor Datas

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
571	224	224	224	192	128	192	192	192	192	224	192	224	192	224	192	254	254	248	248
572	240	248	240	240	224	252	248	240	224	255	248	255	252	224	192	255	252	224	192
573	224	240	224	192	192	224	224	224	192	192	192	224	192	252	252	192	128	240	224
574	252	252	255	254	254	252	252	248	240	248	240	254	252	248	240	252	248	252	248
575	240	248	240	248	240	248	240	248	248	248	240	240	240	240	240	240	224	254	252
576	248	248	240	248	240	248	240	248	240	240	240	240	240	240	248	240	224	240	224
577	224	248	240	240	240	224	224	224	224	224	192	224	192	224	192	224	192	192	192
578	128	128	128	192	128	192	128	192	128	192	128	192	128	192	192	192	192	192	192
579	192	192	128	192	128	192	128	192	128	192	128	192	128	192	192	192	128	192	128
580	192	192	192	192	192	192	192	192	192	192	192	192	128	192	192	192	192	192	192
581	192	192	192	192	192	192	128	192	128	192	128	192	192	192	192	192	192	192	128
582	192	192	192	192	128	192	192	192	192	192	128	192	192	192	192	192	128	192	192
583	192	192	192	192	192	224	192	192	128	192	192	192	192	192	192	192	192	192	192
584	192	192	192	192	192	240	224	248	240	248	240	240	240	224	192	240	224	224	224
585	192	224	192	224	192	224	192	224	192	224	192	224	224	224	224	255	254	254	254
586	254	248	248	255	255	254	255	248	240	240	240	240	240	248	240	248	240	255	248
587	252	248	240	252	248	248	240	248	240	240	224	240	240	248	240	248	240	255	248
588	240	240	240	248	240	240	240	248	240	224	224	224	224	224	224	240	224	240	240
589	240	240	240	248	248	248	240	252	248	255	254	254	252	255	252	252	248	254	254
590	252	252	248	252	248	248	248	252	248	254	252	255	255	252	248	252	248	252	252
591	252	252	252	254	252	240	240	255	255	254	248	254	252	255	254	252	248	255	255

Figure 4.12: Saved excel sheet which contains artix-7 sensor acquisition data.

4.2. Central Controller Implementation Results

In my thesis work, I have developed embedded C++ code for MSP430 board via Code Composer Studio program. After MSP430 system gets the master-slave configurations of the devices from PC Application via UART interface, MSP430 board handles the switching tasks between the boards via their reconfiguration GPIOs. In both boards, one discrete GPIO assigned to handle the full reconfiguration via jumping between image files in flash memories. As seen below figure, the central control application firmware takes the 198 Kbytes of memory out of 512 Kbytes of RAM. Also firmware uses 924 Kbytes of memory out of 16350 Kbytes of Flash memory located in MSP430G2553 board.

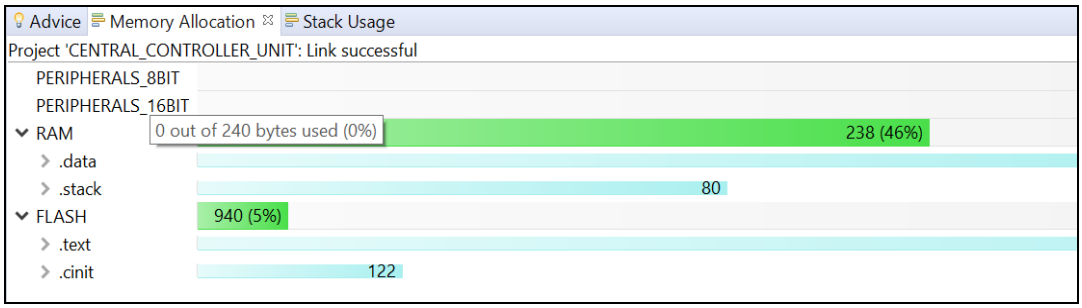

Figure 4.13: Code composer studio memory allocation for central controller unit project.

There are two working modes for MSP430 board for configuration purposes.

First working mode established like this:

- If PC-App sends datastream of 0x41 0x42 0x44 via UART interface, MSP430 board will detect the incoming messages via synchronization bytes of 0x41 0x42. Then mode of operation will be initiated via third byte which has the value 0x44.
- After this, MSP430 will reconfigure Artix-7 as master by pulling the RESET_RECONFIGURATION_PIN to high state and start multiboot operation of Artix-7 to start via updated bitstream in Flash memory. Since STM32 started as slave mode, MSP430 board will not change anything on STM32 device.

Second working mode established like this:

- If PC-App sends data stream of 0x41 0x42 0x43 via UART interface, MSP430 board will detect the incoming messages via synchronization bytes of 0x41 0x42. Then mode of operation will be initiated via third byte which has the value 0x43.
- After this, MSP430 will reconfigure STM32 as master by pulling the reset pin, B11 pin to high state. Then MSP430 board will change the reset pin state to low and start bootloader jump operation of STM32 to start with updated application file located in Flash memory. Since Artix-7 started as slave mode, MSP430 board will not change anything on Artix-7 device.

```
while(1){

    //SYNC BYTES CONTROL
    if((IFG2&UCA0RXIFG)){
        if(synccontrol == 2){
            synccontrol = 0;
            ReceivedData[2] = UCA0RXBUF;
            ser_output(ReceivedData);
        }
         if(UCA0RXBUF == 0x41 & synccontrol == 0){
             synccontrol = 1;
             ReceivedData[0] = 0x41;
         }
         if(UCA0RXBUF == 0x42 & synccontrol == 1){
             synccontrol = 2;
             ReceivedData[1] = 0x42;
         }
        // *ReceivedData = UCA0RXBUF;
         IFG2 &= ~UCA0RXIFG; // Clear RX flag
     }
```

Figure 4.14: Code fragment of msp430 board to control synchronization bytes and mode byte.

```
//------------ARTIX-7 OR STM32 CONFIGURATION ARRANGEMENT-------------------------//
//STM32 Reset and Pin5 set as high for ADXL345 Reading Master and Artix-7 FPGA as Slave
 if(ReceivedData[2] == 0x43){
            P2OUT = 0x28; //make it low for slave function   00101000
            __delay_cycles(32768);
            P2OUT = 0x20;
            __delay_cycles(32768);
            P2OUT = 0x28;
            ser_output(text3);
}
//Artix-7 FPGA as Master , STM32 as slave after reset
 if(ReceivedData[2] == 0x44){
            P2OUT = 0x18; //make it HIGH for master function of Artix-7
            __delay_cycles(32768);
            P2OUT = 0x10;
            __delay_cycles(32768);
            P2OUT = 0x18;
                ser_output(text2);
}
```

Figure 4.15: Code fragment of msp430 board for the management of artix-7, stm32 boards.

4.3. IoT Device-1 Implementation and Results

I have used VIDADO 2021.1 to write the VHDL code for two application firmwares and loading the firmware to the flash memory on CMOD A7 board.

For the initial golden VHDL code, I have declared the iprog_rst module inside main module as seen below:

```
component iprog_rst is
  generic (
    WBSTAR  : std_logic_vector(31 downto 0) := x"002fffff"     -- WBSTAR Address for updated file
  );
  port (
    RST     : in  std_logic;
    CLK     : in  std_logic;
    RIP     : out std_logic
  );
end component;
```

Figure 4.16: Golden vhdl file top module's iprog_rst declaration.

The ICAPE2 primitive is used and instantiated inside iprog_rst module as seen below:

```
ICAPE2_inst: ICAPE2

generic map (
      DEVICE_ID => X"362093",     -- Specifies the pre-programmed Device ID value to be used for simulation
                                  -- purposes.
      ICAP_WIDTH => "X32",        -- Specifies the input and output data width.
      SIM_CFG_FILE_NAME => "NONE" -- Specifies the Raw Bitstream (RBT) file to be parsed by the simulation
                                  -- model.
)
port map (
   O     => open,       -- 32-bit output: Configuration data output bus
   CLK   => CLK,        -- 1-bit input: Clock Input
   CSIB  => cs_l,       -- 1-bit input: Active-Low ICAP Enable
   I     => d_swapped,  -- 32-bit input: Configuration data input bus
   RDWRB => r_wx        -- 1-bit input: Read/Write Select input
);
```

Figure 4.17: ICAPE2 primitive declaration.

As seen above, ICAPE2 primitive contains the definitions which are DEVICE_ID, SIM_CFG_FILE_NAME. These definitions are for simulation purposes and configuration registers could be read from those. Input and output data width can be defined via ICAP_WIDTH variable. As it is defined as 32 bit values, I have used 32 bits as ICAP_WIDTH value. In the interface, we needed to define the read and write select and chip select to send with the datastream. In the datastream, it is defined in which series of data will be sent to do the multiboot operation properly according to the application note.

It is recommended to use the words like this for multiboot operation by Xilinx Application note[5]:

Configuration Data (hex)(1)	Explanation
FFFFFFFF	Dummy Word
AA995566	Sync Word
20000000	Type 1 NO OP
30020001	Type 1 Write 1 Words to WBSTAR
00000000	Warm Boot Start Address (Load the Desired Address)
30008001	Type 1 Write 1 Words to CMD
0000000F	IPROG Command
20000000	Type 1 NO OP

Figure 4.18: IPROG command through icape2 primitive.

Normally flash addresses contain ‘’FF’’ value, when configuration logic acknowledges the sync word, then system captures the datastream and processes it. Then system expects ‘’Type 1 NO OP’’ data to exist and continues operation. Then it

is needed to define the WBSTAR register value as 32 bit address. This value determines the updated image file location in flash memory and I have defined ‘’002fffff’’ as my start address in flash memory for my updated image file. Then I have to write IPROG command to the CMD register. This will trigger the reconfiguration mechanism inside FPGA but will hold the WBSTAR register value. The ordering of the bits for each byte in datastream is reversed and send to the datastream to allow 3 cycles for Artix-7 FPGA to function properly as seen below figure.

```
case count is
   when  0 => data <= x"FFFFFFFF";    -- Dummy Word
   when  1 => data <= x"FFFFFFFF";    -- Dummy Word
   when  2 => data <= x"FFFFFFFF";    -- Dummy Word
   when  3 => data <= x"FFFFFFFF";    -- Dummy Word
   when  4 => data <= x"FFFFFFFF";    -- Dummy Word
   when  5 => data <= x"20000000";    -- Type 1 NO OP
      cs_l  <= '0';
      r_wx  <= '0';
   when  6 => data <= x"AA995566";    -- Sync Word
   when  7 => data <= x"20000000";    -- Type 1 NO OP
   when  8 => data <= x"20000000";    -- Type 1 NO OP
   when  9 => data <= x"30020001";    -- Type 1 Write 1 Word to WBSTAR
   when 10 => data <= WBSTAR;         -- Warm Boot Start Address
   when 11 => data <= x"20000000";    -- Type 1 NO OP
   when 12 => data <= x"20000000";    -- Type 1 NO OP
   when 13 => data <= x"30008001";    -- Type 1 Write 1 Words to CMD
   when 14 => data <= x"0000000F";    -- IPROG Command
   when 15 => data <= x"20000000";    -- Type 1 NO OP
   when 16 => data <= x"20000000";    -- Type 1 NO OP
      cs_l     <= '1';
      r_wx     <= '1';
   when others =>
      cs_l     <= '1';
      r_wx     <= '1';
end case;
```

Figure 4.19: VHDL code fragment to send the icape2 datastream.

In the main module of application-1, I have declared input and outputs for the golden file as below figure:

```
entity Application1 is
    Port (
           BTN          : in  STD_LOGIC_VECTOR (1 downto 0);
           CLK          : in  STD_LOGIC;
           LED          : out  STD_LOGIC_VECTOR (1 downto 0);
           power3V : out STD_LOGIC;
           RESET_SIGNAL_RECONFIGURATION: in STD_LOGIC;
           UART_TXD     : out  STD_LOGIC;
           RGB0_Red     : out  STD_LOGIC;
           RGB0_Green    : out  STD_LOGIC;
           RGB0_Blue    : out  STD_LOGIC
              );
end Application1;
```

CMOD_A7_FPGA_PRO
OneDrive/Bureaublad/
GOLDEN_000000/CM

Figure 4.20: Main module input/output declaration of application1.

As seen above, golden image will be triggered via signal defined as “RESET_SIGNAL_RECONFIGURATION”. This signal is connected to the reset signal in the iprog_rst module. If this signal goes high, the count variable will start from zero to send the appropriate datastream to handle reconfiguration as seen below.

```
if(CLK'event and CLK='1') then

   if(RST = '1')  then
      run      <= '1';
   end if;

   if(run = '0')  then
      RIP      <= '0';
      cs_l     <= '1';
      r_wx     <= '1';
      count    <= 0;
      data     <= (others => '1');
   else
      RIP      <= '1';
      if(count /= CMAX)  then
         count <= count + 1;
      end if;
```

Figure 4.21: Code fragment in iprog_rst module to start datastream.

Inside of the process definition, if reset signal goes high, system will set signal ‘’run’’ to go high. Then system will send the datastream to determine WBSTAR memory address, send IPROG command to start reconfiguration mechanism to immediately run FPGA with updated bitstream.

In the application 1, 11 I/O pins, 164 Cells, 267 Nets have been used, It can be seen in schematic tab inside synthesized design of VIVADO 2021.1 The application 1 only contains RGB led brightness change, led lighting functions and uses 100 MHz clock frequency. Static power consumption is 0.072 Watt and dynamic power consumption is 0.111 Watt as a total chip draws 0.183 Watts of power for its own firmware. In the power consumption, clocks draw more power as 0.002 Watts because in this application system runs with 100 MHz frequency via frequency divider of PLLs. The VIVADO 2021 figure which contains this information can be seen below.

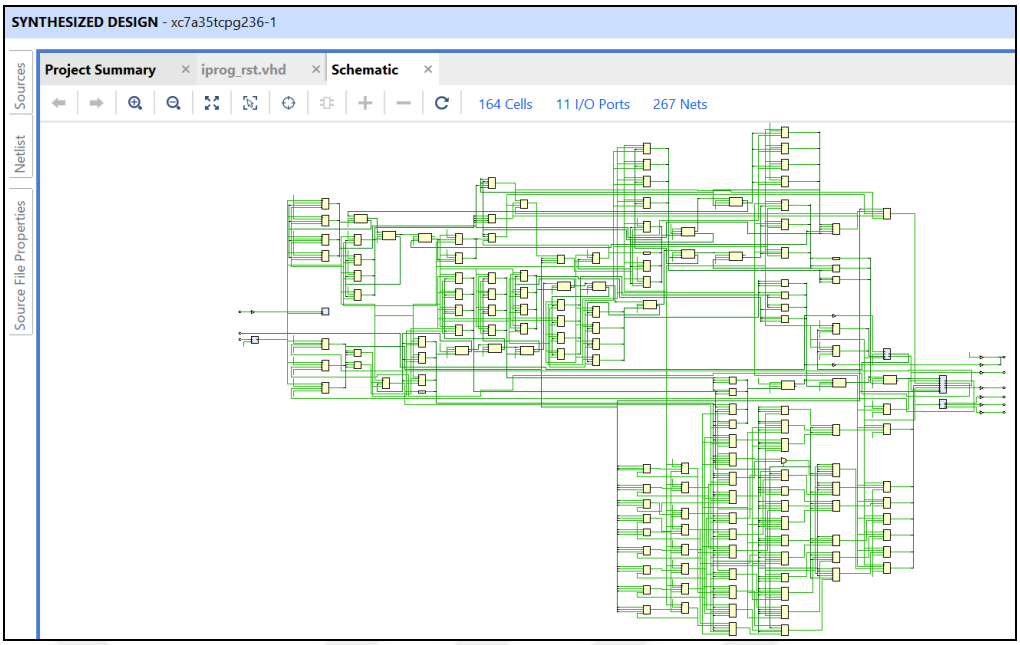

Figure 4.22: Schematic output of vivado 2021.1 for golden image file for application-1.

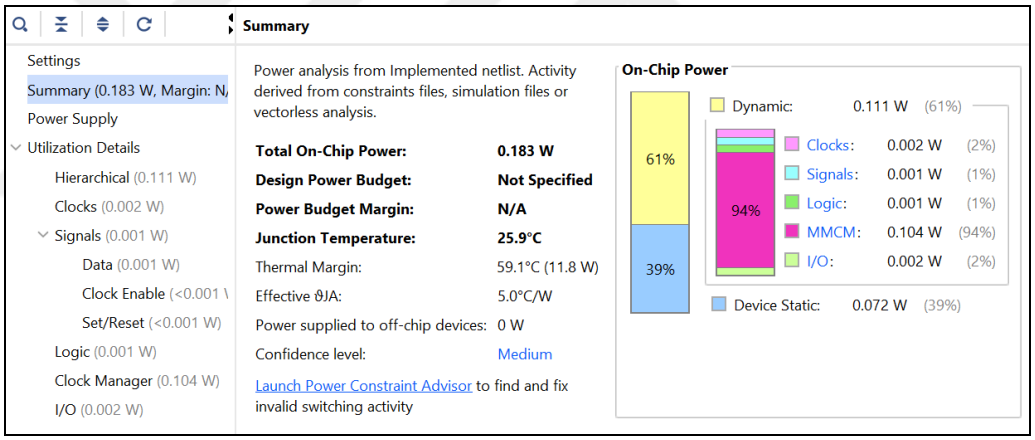

Figure 4.23: Power activity of application-1.

In the main module of application-2, I have declared input and outputs for the golden file as below figure:

```
entity Application2 is
    Port (
           SPI_SCLK     : out std_logic;
           SPI_CS       : out std_logic;
           SPI_MOSI     : out std_logic;
           SPI_MISO     : in std_logic;
           BTN          : in  STD_LOGIC_VECTOR (1 downto 0);
           CLK          : in  STD_LOGIC;
           LED          : out  STD_LOGIC_VECTOR (1 downto 0);
           power3V : out STD_LOGIC;
           GND : out STD_LOGIC;
           RESET_SIGNAL_RECONFIGURATION: in STD_LOGIC;
           UART_TXD     : out  STD_LOGIC;
           RGB0_Red     : out  STD_LOGIC;
           RGB0_Green    : out  STD_LOGIC;
           RGB0_Blue    : out  STD_LOGIC
              );
end Application2;
```

Figure 4.24: Main module input/output declaration of application2.

As seen above, updated image will have SPI pins declarations as input master in slave out pin namely SPI-MISO, output master out slave in pin namely SPI-MOSI, output SPI-clock and also output for UART interface. Also system runs with 10 MHz of clock frequency which is different from the golden initial files ‘clock which is 100 MHz.

In the application 2, 16 I/O ports, 208 Cells, 360 Nets have been used, It can be seen in schematic tab inside synthesized design of VIVADO 2021.1 The application 2 contains RGB led brightness change, led lighting functions and SPI interface with ADXL345 sensor and uses 10 MHz clock frequency. Also it sends the SPI interfaced ADXL345 data via UART interface in master mode. In Synthesized design, it is possible to see the power activity of the Artix-7 FPGA according to the master mode application. Static power consumption is 0.072 Watt and dynamic power consumption is 0.106 Watt as a total chip draws 0.178 Watts of power for its own firmware. In the power consumption, clocks draw less power as 0.001 Watts because in this application system runs with 10 MHz frequency. The VIVADO 2021 figures which contain this information can be seen below.

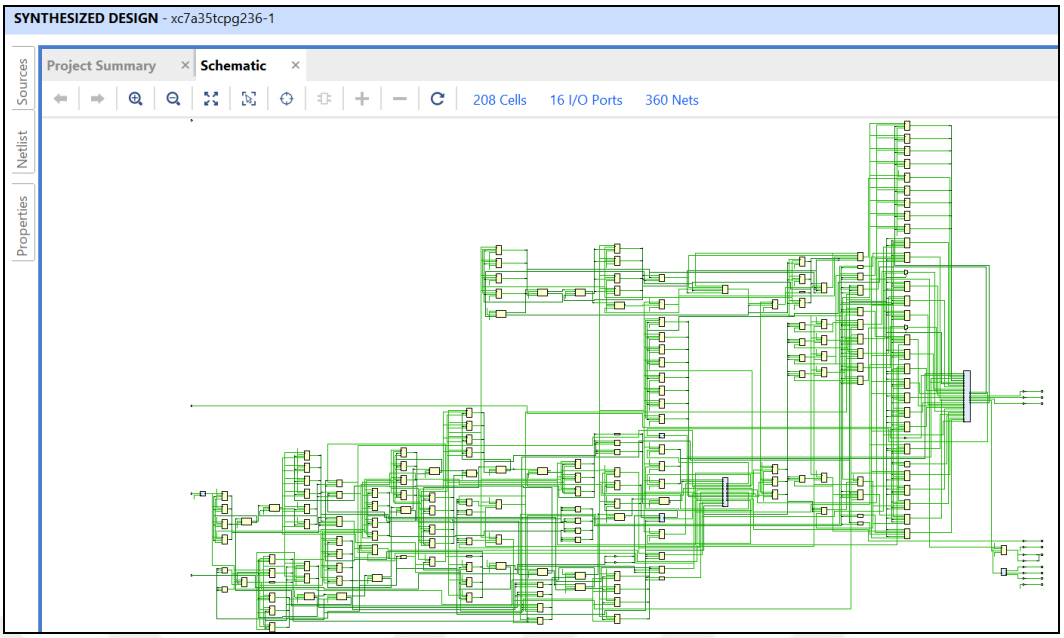

Figure 4.25: Schematic output of vivado 2021.1 for updated image file for application-2.

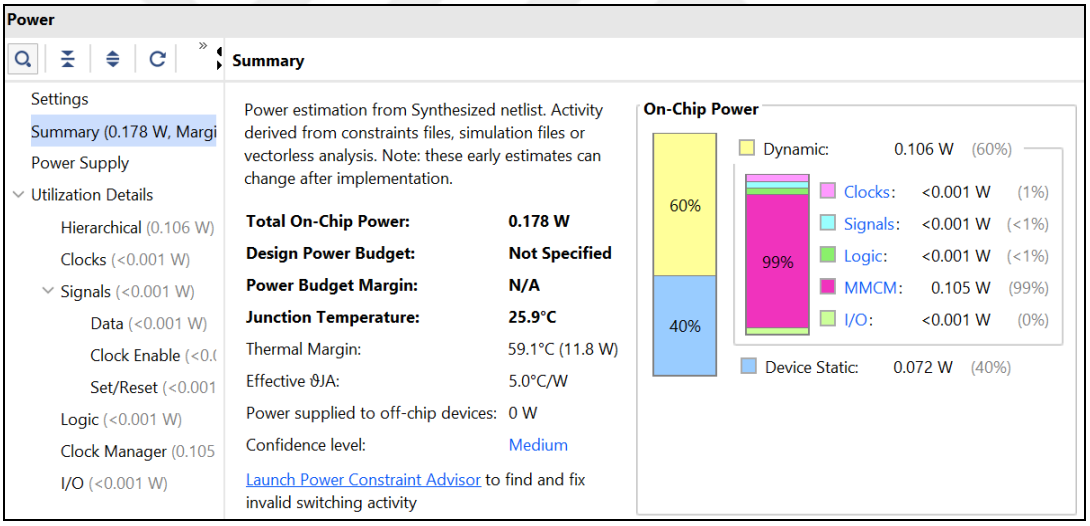

Figure 4.26: Power activity of application-2.

4.4. IoT Device-2 Implementation and Results

In the thesis work, I will be using bootloader custom file and two application files written for STM32 to embed three different firmwares inside flash memory via STM32CubeProgrammer tool. Custom bootloader code will be written to flash address starting from 0x08000000. Application-1 firmware code will be written from address A to address B. Application-2 firmware code will be written from address C to address D as seen in figure.

At the customized bootloader file, user should first select boot from flash mode to enable flash for programming files. The system will run with a customized bootloader file that's written inside flash memory. When system boots, it will check its own dedicated GPIO to see which application is wanted. After this control, system will jump to the specific application address to run the programming file allocated to it.

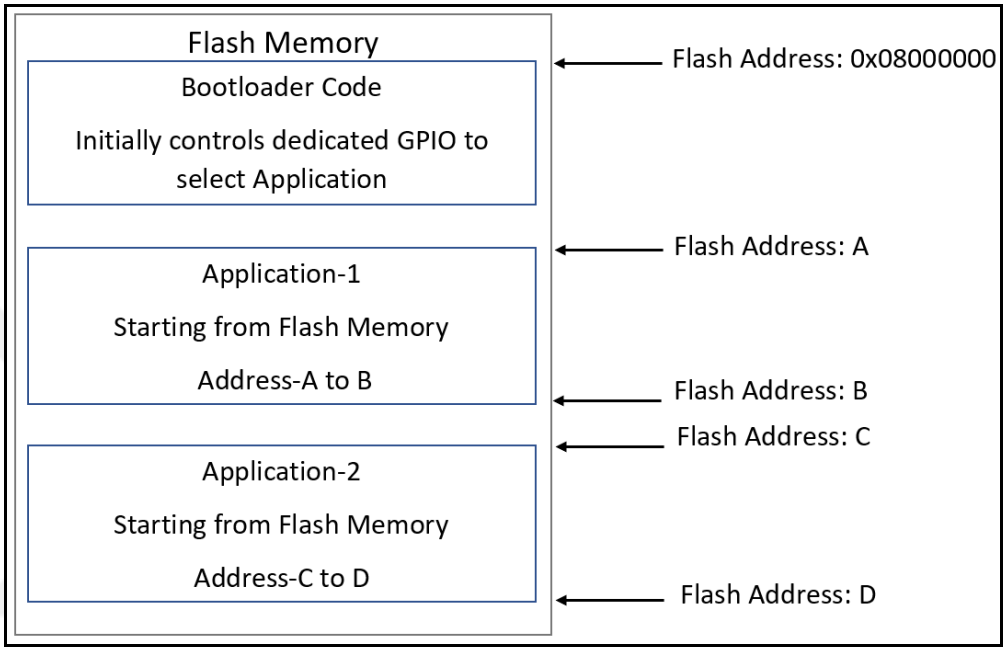

Figure 4.27: STM32 board flash memory for bootloader application.

After each reset, bootloader custom firmware inside flash memory will be activated to control the dedicated GPIO, if that GPIO set to high, system will jump to Application-2 memory address. Then system will continue working with Application-2 firmware. If user selects dedicated GPIO to go low, system will jump to Application-1 memory address. Then system will continue working with Application-1 memory address.

At this implementation of STM32F103C8 board, bootloader file is created to control the dedicated GPIO pin B11 to determine which application from flash to execute. In bootloader project, linker file and bootloader c code to jump between applications.

In linker files memory addresses and sizes are determined as 20 Kbytes for each application file since I had 64 Kbytes of flash memory on the board. For bootloader

custom file, 20 Kbytes of size have been given via flash line in memory section as it can be seen from below figure 4.28.

As seen below in figure x, I have address space from 0x8000000 to address A of flash memory for bootloader mode application. Thus I have separated address space starting from 0x8000000 to the bootloader file as it can be seen in figure below.

```
/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);    /* end of "RAM" Ram type memory */

_Min_Heap_Size = 0x200 ;    /* required amount of heap  */
_Min_Stack_Size = 0x400 ;   /* required amount of stack */

/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 20K
  FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 20K /*Flash Memory of 64 Kbyte */
                                                        /*20 Kbytes given to bootloader file */
}
```

Figure 4.28: STM32 bootloader flash linker file memory definition.

As seen below in figure 4.29, I have address space from A to address B of flash memory for slave mode application. Thus I have separated address space starting from 0x8005000 to the application 1 for slave mode as it can be seen in figure below.

```
/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);    /* end of "RAM" Ram type memory */

_Min_Heap_Size = 0x200 ;    /* required amount of heap  */
_Min_Stack_Size = 0x400 ;   /* required amount of stack */

/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 20K
  FLASH    (rx)    : ORIGIN = 0x8005000,   LENGTH = 22K /*Flash Memory of 64 Kbyte */
                                                        /*20 Kbytes given to slave mode application file */
}
```

Figure 4.29: STM32 application-1 flash linker file memory definition.

As seen below in figure x, I have address space from C to address D of flash memory for master mode application. Thus I have separated address space starting from 0x800A800 to the application 2 for master mode as it can be seen in figure below.

```
/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);    /* end of "RAM" Ram type memory */

_Min_Heap_Size = 0x200 ;    /* required amount of heap  */
_Min_Stack_Size = 0x400 ;   /* required amount of stack */

/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 20K
  FLASH    (rx)    : ORIGIN = 0x800A800,   LENGTH = 22K /*Flash Memory of 64 Kbyte */
                                                        /*22 Kbytes given to master mode application file */
}
```

Figure 4.30: STM32 application-2 flash linker memory definition.

In bootloader c code, dedicated GPIO state is being controlled, if the state goes the same as Reset pin, slave mode application will be chosen. Otherwise, system will run from master mode application since App2 is chosen in below figure. The main reason to assign this pins' state to reset pin, is to synchronously select system running application if reset pin is pushed or triggered by user.

```
if(HAL_GPIO_ReadPin(App_GPIO_Port, App_Pin) == GPIO_PIN_RESET)
    App = App1;
else
    App = App2;
```

Figure 4.31: Application selection via bootloader c code section.

In bootloader c file, there is a state defined as "JumpMode". If JumpMode is active, system will first control which application is selected via GPIO. Then it will control whether there are empty addresses or not to see the code exists in the memory section. APP1_START refers to the flash address where the slave mode application is located. Similarly, APP2_START belongs to the flash address where the master mode application is located.

```
if(bootloaderMode == JumpMode)
{
    if(App == App1)
    {
        //This control aims to understand if the code really exist
        uint8_t emptyCellCount = 0;
        for(uint8_t i=0; i<10; i++)
        {
            if(readWord(APP1_START + (i*4)) == -1)
                emptyCellCount++;
        }
        if(emptyCellCount != 10)
            jumpToApp(APP1_START);
        else
            errorBlink();
    }
    else
    {
        //This control aims to understand if the code really exist
        uint8_t emptyCellCount = 0;
        for(uint8_t i=0; i<10; i++)
        {
            if(readWord(APP2_START + (i*4)) == -1)
                emptyCellCount++;
        }
        if(emptyCellCount != 10)
            jumpToApp(APP2_START);
        else
            errorBlink();
    }
}
```

Figure 4.32: Bootloader jump feature at dedicated c file.

In STM32 Board, I have implemented the master mode application file to get sensor values via SPI interface. First I have written the functions to write, read and initialize the ADXL345 sensor device. For ADXL sensor initialization, I have determined the data format range to cover +4g acceleration to -4g acceleration by writing the 0x31 register to value 0x01. To reset all bits in the sensor, I have written 0x00 value to 0x2D register. Then for power control measurement, I have written value 0x08 to the 0x2D register. For ADXL sensor to read, multiple byte read mode enabled, CS pin state changed to low for enabling sensor. Then it is needed to send the address where we want to read the data from. In the address, I have read 6 bytes of data since multiple byte read is enabled. Then CS pin state changed to high for disabling the sensor. For ADXL sensor to write, multiple write modes are enabled. Similarly, CS pin should go low state to enable the sensor, and then it is possible to transmit address and data values. To end operation, CS pin should go high state to disable the sensor.

```c
void adxl_write (uint8_t address, uint8_t value)
{
    uint8_t data[2];
    data[0] = address|0x40;
    data[1] = value;
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
    HAL_SPI_Transmit (&hspi1, data, 2, 100);
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
}

void adxl_read (uint8_t address)
{
    address |= 0x80;
    address |= 0x40;
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_RESET);
    HAL_SPI_Transmit (&hspi1, &address, 1, 100);
    HAL_SPI_Receive (&hspi1, data_rec, 6, 100);
    HAL_GPIO_WritePin (GPIOB, GPIO_PIN_7, GPIO_PIN_SET);
}

void adxl_init (void)
{
    adxl_write (0x31, 0x01);
    adxl_write (0x2d, 0x00);
    adxl_write (0x2d, 0x08);
}
```

Figure 4.33: ADXL345 write, read, initialization functions in master mode main c file.

In master mode c file, ADXL345 sensor values will be continuosly read via SPI interface for x axis, y axis and z axis. If z Axis value goes higher than 0.5 g value, GPIO pin 13 which is connected to the LED on the board turns on. Otherwise, led will go to turn off state to show the gravity acceleration of 1 g is not detected.

```c
while (1)
{
    // READ DATA
        adxl_read (0x32);
        x = ((data_rec[1]<<8)|data_rec[0]);
        y = ((data_rec[3]<<8)|data_rec[2]);
        z = ((data_rec[5]<<8)|data_rec[4]);
        // Convert into 'g'
        xg = x*.0078;
        yg = y*.0078;
        zg = z*.0078;
        // Acceleration value control
        if(zg > 0.5)
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
        else
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);

  /* USER CODE BEGIN 3 */
        HAL_Delay (200);   // wait for 200 ms
}
```

Figure 4.34: Code fragment in application-2 to read spi interfaced sensor data.

For the STM32 board implementation, STM32CubeIDE creates files with the “ELF” extension. The bootloader project elf file contains 19.77 Kbyte memory starting from address 0x08000000 to address 0x08005000. The slave mode

application project elf file contains 3.84 Kbyte memory starting from address 0x08005000 to address 0x0800A800. The master mode application project elf file contains 12.91 Kbyte memory starting from address 0x0800A800 to address 0x08010000. The memory output details of the STM32CubeIDE can be seen in figure below:

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20005000	20 KB	13.64 KB	6.36 KB	31.78%
FLASH	0x08000000	0x08005000	20 KB	240 B	19.77 KB	98.83%

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20005000	20 KB	18.45 KB	1.55 KB	7.73%
FLASH	0x08005000	0x0800a800	22 KB	18.16 KB	3.84 KB	17.47%

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20005000	20 KB	18.19 KB	1.81 KB	9.06%
FLASH	0x0800a800	0x08010000	22 KB	9.09 KB	12.91 KB	58.66%

Figure 4.35: Memory regions for bootloader firmware's elf file, slave mode firmware's elf file and master mode firmware's elf file.

5. CONCLUSION AND FUTURE WORK

5.1. Conclusion

Reconfigurable models for IoT devices are chosen in this thesis work to establish a system which is capable of switching tasks on the fly with full reconfiguration. For this purpose, I have chosen widely used processor families in automotive, military, aerospace, enterprise, healthcare industries which are Xilinx FPGA, STM, MSP series MCUs. Since all boards communicate via serial communication interfaces with each other and the sensor block, I have come up with a GUI to communicate and configure IoT devices via user inputs. In a network full of IoT devices, it is possible to reconfigure those devices firmware without unplugging them to stop operation or cut the power in each of them. This increases the flexibility and development with the complex infrastructure to reduce the life cycle costs and maintainability efforts.

5.2. Future Work

System runs completely bare-metal and that creates a need to use additional PC for GUI development. In the future there could be an improvement to use embedded Linux, Linux or Windows running on multicore CPU board to control the MSP, STM series processors. Then GUI could be developed and run within that CPU's operating system. This implementation is finished and demonstrated in section 5.3.1.

After this, next step is to assign different tasks between cores in the CPU and external MSP, STM series processors. Then generic framework could be applied and extended to heterogeneous tasks for diverse applications easily with CPUs, FPGAs and various MCU boards. This work can continue to accomplish a library with all these different MCUs, CPUs implementations with mixture of communication interfaces such as Ethernet, PCI Express, LVDS, Bluetooth, Wifi, USB, CAN, SPI, I2C, UART. Then it will be possible to use this library in our central controller multi-core CPU's embedded operating system where any IoT component can be plugged to be configured easily via different communication interfaces. This implementation nearly finished and can be seen in section 5.3.2.

5.3. Ongoing Projects

5.3.1. Generic Framework Implementation with Multicore Application Processor

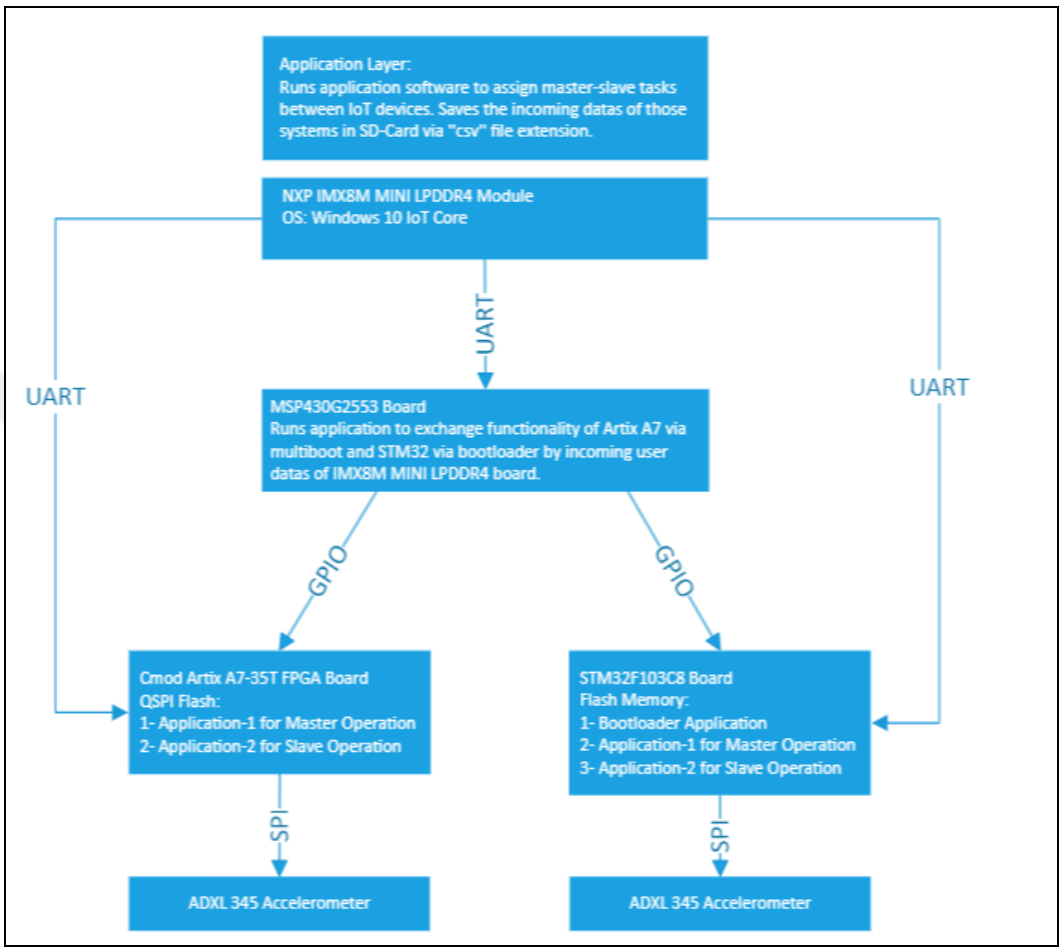

Figure 5.1: Flow diagram of IMX8 Application Processor Implementation.

As seen above, I have used NXP IMX8M Mini LPDDR4 board for implementation of generic framework proposed in the thesis work. In this application, whole software runs inside of IMX8M Mini board with its own operating system which is Windows 10 IoT Core. Therefore, I have created Visual Studio XAML project which can be deployed to 64 bit ARM processor. In Visual Studio software, it is possible to deploy XAML projects to 32 bit/64 bit ARM, 32 bit/64 bit Intel processors after defining the IP address of the device[26]. This is a developer approach to debug applications to arm processors which are connected to Ethernet network.

After deployment of Visual Studio Project, it is possible to connect to MSP430G2553, Artix-7 and STM32F103C8 boards simultaneously. Then same operation of the boards for master-slave interchangeability is accomplished and it is possible to save the incoming sensor data to the SD card as storage file with csv extension. I have used 16 GB SanDisk SD Card to boot windows 10 IoT Core for IMX8M board and save the data in the remaining space of the SD Card. For large volumes of data this implementation can be extended by NVMe SSD memory connected to the PCI express bus underneath this IMX8M board since those memories could have 1Terabyte or more space. Then system could run for longer periods of time without interruption to have data acquisition function in an IoT Network.

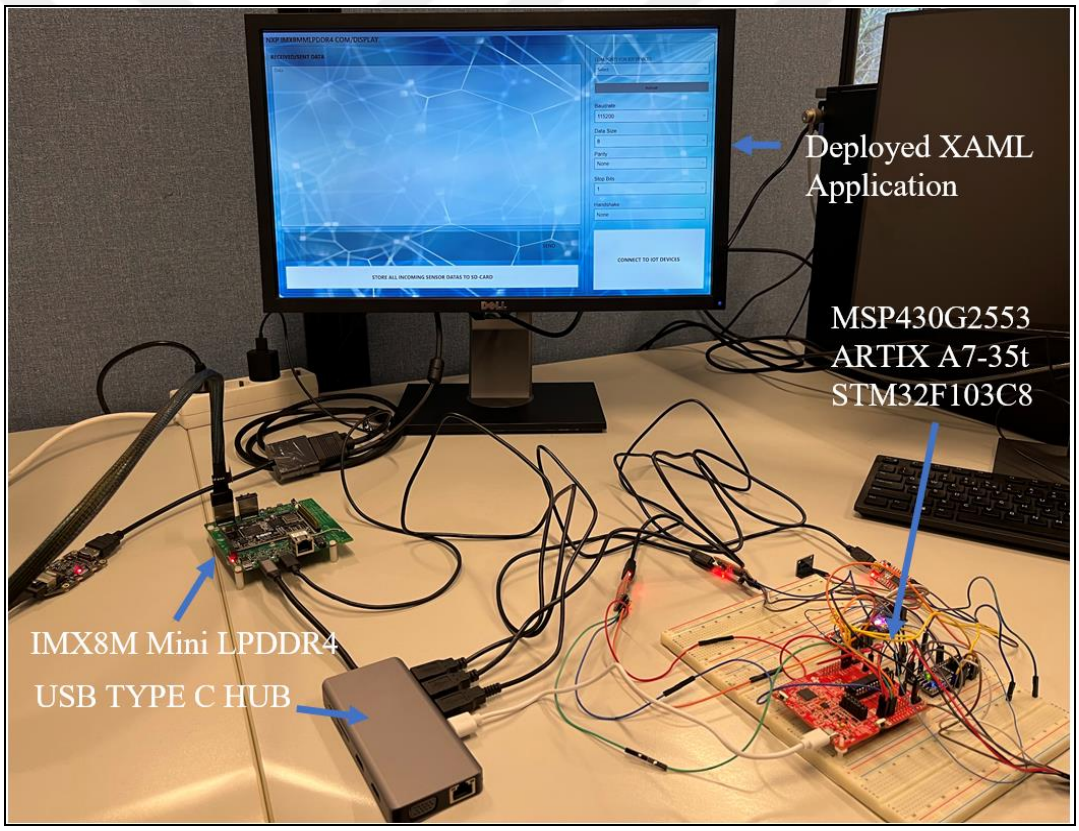

Figure 5.2: Generic framework implementation with imx8m processor.

5.3.2. Cluster Network Implementation of Generic Framework

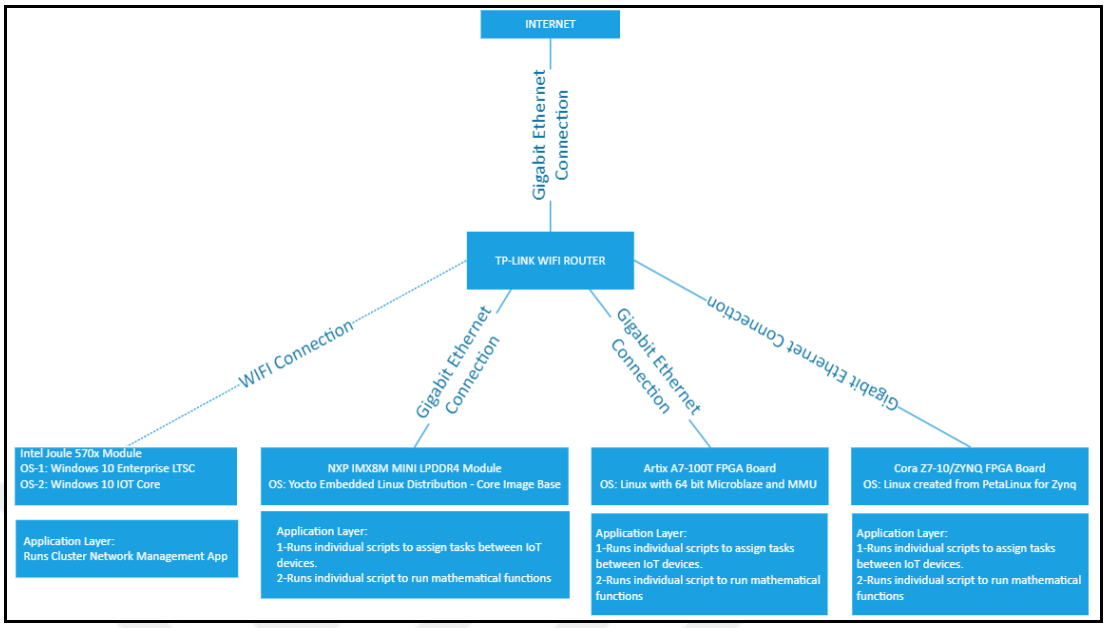

Figure 5.3: Generic framework implementation with multiple processor boards.

As seen above, I have enlarged the implementation of generic framework with different processor boards which includes Intel Joule series, NXP IMX series and 7 series Xilinx FPGA boards.

As seen below, there are 4 different operating systems run on each board:

- Intel Joule 570x module is capable of running windows 10 IoT core and windows desktop versions as well. In this specific example, intel joule module runs 64 bit windows 8.1.
- NXP IMX-8 board runs core-image-base image file which is created by yocto project. Yocto project is used to create custom embedded linux to access all peripherals of the board. I have created ‘’core-image-base’’ file for imx8 and partitioned it in SD card which boots linux on power-up[27].
- Artix A7-100t FPGA board is used with 64 bit Linux which runs on microblaze processor. This implementation’s programmable/hardware logic have been done by VIVADO software. Then I have exported this hardware for configuring petalinux to have embedded linux for Artix A7-100t FPGA board. There are several other demonstrations possible with RISC-V processor

implementations inside this FPGA but I have used microblaze processor to run linux for this application[28].

- Cora Z7-10 Zynq FPGA board is used with petalinux tools to create embedded linux distribution to access all peripherals of the board after exporting hardware from VIVADO software.

The purpose is to use the application software of Intel Joule Module to access all application processors via SSH connection and assign functionalities of IoT devices that are connected to Zynq, Artix, IMX8M boards via message passing interface[29].

For this setup, Artix-7, Zynq and IMX8M boards are connected to TP-LINK router via gigabit-ethernet cables. Intel Joule Module is connected to TP-LINK router via WIFI connection. I have used USB 3.0 compatible WIFI adapter which have 1200 Mbps speed on 2.4GHz and 5GHz networks. Then it is possible to access Artix-7, Zynq, IMX8M boards via router by ethernet connection through Intel Joule's user application which can be seen on Figure 5.5. I have used SSH connection to remotely login each of the board and I have created individual shell scripts to run executables dedicated for master-slave functionality assignment between IoT devices connected[30]-[31]. Then generic framework proposed in this thesis is applied to IMX8M, Artix-7 and Zynq boards separately. Thus it is possible to attach Artix A7-35t and STM32F103C8 to these boards to switch between master – slave modes and data acquisition purposes.

In short, this generic framework can be widened with multi operating system with multi user applications instead of one operating system to control IoT devices with one user application. In thesis studies, I have created visual studio application in PC to save the incoming datas from Artix A7-35t or STM32F103C8 boards accelerometers.

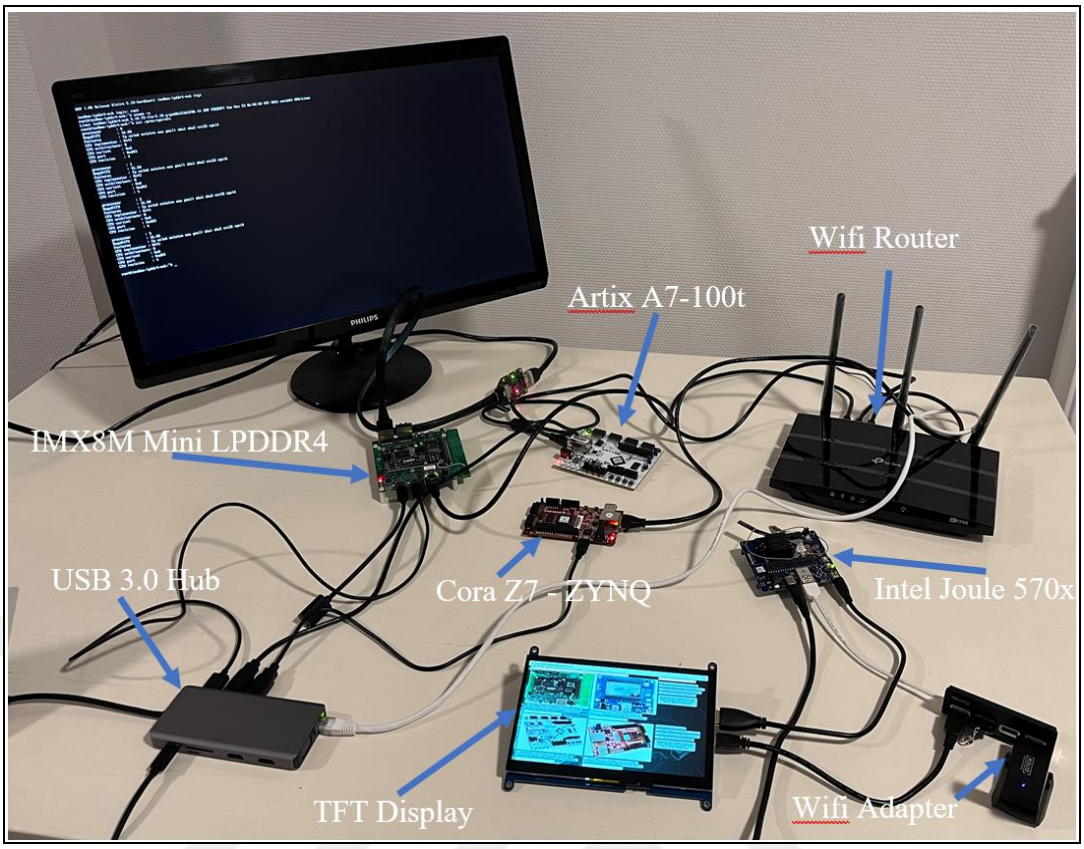

Figure 5.4: Generic framework implementation for cluster network.

Figure 5.5: Cluster network master node application software.

REFERENCES

[1] Piłat A., Grega W., (2007), "Hardware and software architectures for reconfigurable time-critical control tasks", Computer Science, 8, 69-69.

[2] Reinders J., (2012), "An overview of programming for Intel Xeon processors and Intel Xeon Phi coprocessors", Intel Corporation, Santa Clara, 1, 1550002.

[3] Anitha A., Latha M. M., (2017), "Partial dynamic reconfiguration framework for FPGAs through remote access", International Journal of High Performance Systems Architecture, 7 (2), 98-104.

[4] Li B., Gakhal K. K., (February 28, 2017), "MultiBoot with 7 Series FPGAs and SPI", Technical Report.

[5] Xilinx, (2018), "7 Series FPGAs Configuration, UG470", v1.13.1.

[6] Xilinx, (2016), "7 Series FPGAs Configurable Logic Block, User Guide, UG474", v1.8.

[7] ST, (2013), "AN2606 Application Note, STM32 microcontroller system memory boot mode", Rev 15.

[8] Pilotto C., Azambuja J. R., Kastensmidt F. L., (2008), "Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications", Proceedings of the 21st annual symposium on Integrated circuits and system design, Association for Computing Machinery, Gramado, Brazil, 199–204.

[9] Lino M., Leão E., Soares A., Montez C., Vasques F., Moraes R., (2020), "Dynamic Reconfiguration of Cluster-Tree Wireless Sensor Networks to Handle Communication Overloads in Disaster-Related Situations", Sensors, 20 (17), 4707.

[10] Shafique A., Cao G., Aslam M., Asad M., Ye D., (2020), "Application-Aware SDN-Based Iterative Reconfigurable Routing Protocol for Internet of Things (IoT)", Sensors, 20 (12), 3521.

[11] Liu Y., Lu Y., Li X., Yao Z., Zhao D., (2020), "On Dynamic Service Function Chain Reconfiguration in IoT Networks", IEEE Internet of Things Journal, 7, 10969-10984.

[12] Vlacheas P. T., Giaffreda R., Stavroulaki V., Kelaidonis D., Foteinos V., Poulios G., Demestichas P., Somov A., Biswas A. R., Moessner K., (2013), "Enabling smart cities through a cognitive management framework for the internet of things", IEEE Communications Magazine, 51, 102-111.

[13] Fernandez Molanes R., Amarasinghe K., Rodríguez-Andina J. J., Manic M., (2018), "Deep Learning and Reconfigurable Platforms in the Internet of Things: Challenges and Opportunities in Algorithms and Hardware", IEEE Industrial Electronics Magazine, 12, 36-49.

[14] Foteinos V., Kelaidonis D., Poulios G., Vlacheas P. T., Stavroulaki V., Demestichas P., (2013), "Cognitive Management for the Internet of Things: A Framework for Enabling Autonomous Applications", IEEE Vehicular Technology Magazine, 8, 90-99.

[15] E V. B., Ennela, M. G., Ovind, S. K., Oteshwara, (2015). "A Reconfigurable Smart Sensor Interface for Industrial WSN in IoT Environment".

[16] K.S. Rekha T. H. S., A.D. Kulkarni, (2018), "Remote Monitoring and Reconfiguration of Environment and Structural Health Using Wireless Sensor Networks", Materials Today: Proceedings,, 5 (1), 1169-1175.

[17] Javed A., Malhi A. K., Kinnunen T., Främling K., (2020), "Scalable IoT Platform for Heterogeneous Devices in Smart Environments", IEEE Access, 8, 211973-211985.

[18] Wanta D., Smolik W. T., Kryszyn J., Wróblewski P., Midura M., (2022), "A Run-Time Reconfiguration Method for an FPGA-Based Electrical Capacitance Tomography System", Electronics, 11 (4), 545.

[19] Gharehbaghi A. M., Maruoka T., Fujita M., (2019). "A New Reconfigurable Architecture with Applications to IoT and Mobile Computing", 133-146, Cham.

[20] Kuon I., Tessier R., Rose J., (2008), "FPGA architecture: Survey and challenges", Edition, Now Publishers Inc.

[21] Crockett L. H., Elliot R., Enderwitz M., Stewart R., (2014), "The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC", Edition.

[22] Xilinx, (2020), "7 Series FPGAs Data Sheet: Overview, DS180", 2.6.1.

[23] Sasamal T. N., Prasad R., (2011), "Module based and difference based implementation of partial reconfiguration on FPGA: A review", International Journal of Engineering Research and Applications (IJERA), 1 (4), 1898-1903.

[24] Xilinx, (October 22, 2021), "Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide , UG953", v2021.2.

[25] ST, (2015), "Medium-density performance line ARM®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces", Rev 17.

[26] Chatterjee A., (2017), "Building apps for the universal Windows platform: explore Windows 10 Native, IoT, HoloLens, and Xamarin", Edition, Apress.

[27] Salvador O., Angolini D., (2017), "Embedded Linux Development using Yocto Projects: Learn to leverage the power of Yocto Project to build efficient Linux-based products", Edition, Packt Publishing Ltd.

[28] Harris S. L., Chaver D., Piñuel L., Gomez-Perez J., Liaqat M. H., Kakakhel Z. L., Kindgren O., Owen R., (2021). "RVfpga: Using a RISC-V Core Targeted to an FPGA in Computer Architecture Education", 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 145-150.

[29] Gropp W., Gropp W. D., Lusk E., Skjellum A., Lusk A. D. F. E. E., (1999), "Using MPI: portable parallel programming with the message-passing interface", Edition, MIT press.

[30] Barrett D. J., Barrett D. J., Silverman R. E., Silverman R., (2001), "SSH, the Secure Shell: the definitive guide", Edition, " O'Reilly Media, Inc.".

[31] Tansley D. S., Tansley D. V., (2000), "Linux and UNIX shell programming", Edition, Addison-Wesley Professional.

BIOGRAPHY

Erdem ÇETİN successfully completed the Department of Electrical and Electronics Engineering which belongs to Faculty of Engineering at Bilkent University. In 2020, he started his graduate study in the Department of Electronic Engineering at Institute of Natural and Applied Sciences of Gebze Technical University. He worked as Electronics Design Engineer from December 2014 to December 2018 in ROKETSAN at defense industry which is the manufacturer of Missile Systems for Turkish Armed Forces. For that time, he mainly focused on designing the Fuse and Arm Mechanisms' multilayer PCBs, Signal integrity,Power integrity and reliable power distribution network analysis and design, embedded software with various types of MCUs, SOCs and VHDL programming of the FPGAs. Also he involved with test equipment designs and test application softwares' GUI development. Then he started working as Avionics Design Engineer from December 2018 to July 2020 for the development of 5th gen Turkish Fighter Aircraft project in Turkish Aerospace Industries. He mainly focused on the design and development of Avionics Core Computing platform which has IMA Architecture and suited to Open VPX standards. After that he started working in ASELSAN as Expert System Design Engineer at July 2020 for the development of hardware infrastructure of Long Range Air Defense System project known as SIPER for Turkish Armed Forces in Defense Industry. At October 2021, He has come to Netherlands as expert electronics design engineer.