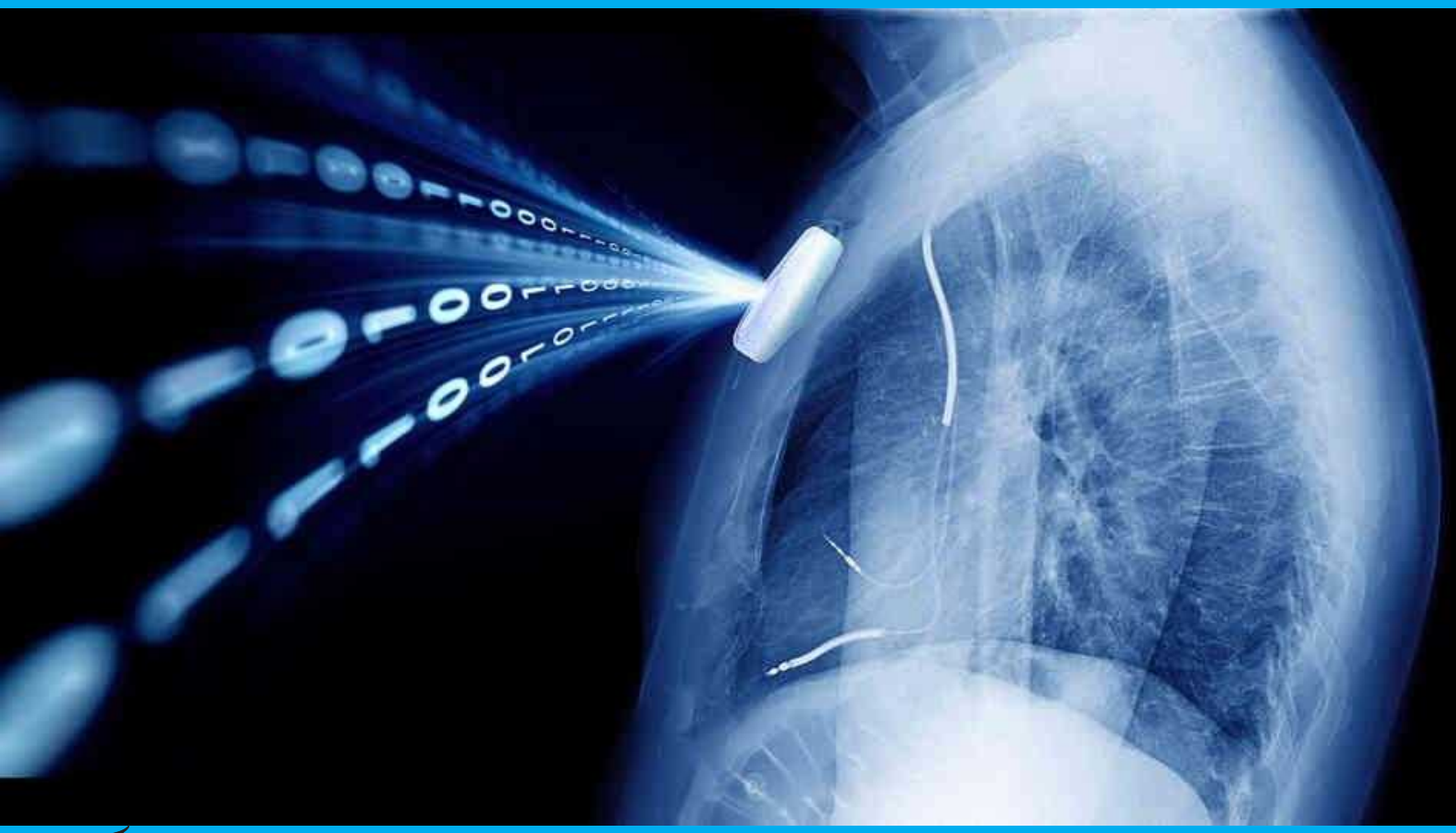
Secure bus architecture for IMDs

Erik Speksnijder
Ismail Bourhaeil



TU Delft

Secure bus architecture for IMDs

by

Erik Speksnijder
Ismail Bourhaeil

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Wednesday June 30, 2021 at 13:30 AM.

Student number:	4905210, 4875400	
Project duration:	April 19, 2021 – July 2, 2021	
Thesis committee:	Dr. I. E. Lager,	TU Delft
	Prof. dr. S. Hamdioui,	TU Delft, supervisor
	Dr. ing. R. Bishnoi,	TU Delft, supervisor
	Dr. ir. C. Strydis,	Erasmus University, supervisor
	M. A. Siddiqi,	Erasmus University, supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Epilepsy is a system-wide phenomenon which manifests physically across the body in various forms such as rapid muscle tone, sweating, elevated heart rate and synchronized neuron firing. Many of these aspects appear ahead of an epileptic incident. If combined together, these aspects are tell-tale signs of an imminent ictal event. Because of these observations, a wireless medical body area network (MBAN) has been proposed, which implements multimodal-data integration and closed-loop seizure suppression.

The design and implementation of the MBAN introduces several challenges, such as data collection, seizure prediction and suppression, secure pairing and communication and the design of a user friendly interface. This thesis will focus on the design and implementation of a secure bus architecture that connects multiple processing cores, a sensor and an actuator within an MBAN node.

The interconnect will provide communication between the masters and slaves via an AMBA AHB-Lite protocol. Furthermore, a memory protection unit (MPU) will deny accesses from unauthorized peripherals. Additionally, the prototype will provide a secure communication protocol for updating the MPU. Finally, the prototype will be able to communicate wirelessly with a smartphone.

The deliverable will be a proof-of-concept implementation on an FGPA, demonstrating the previously described functionalities. Nevertheless, the design choices will be made with the application in mind.

Preface

This thesis is written in the context of the Bachelor Graduation Project of the bachelors program electrical engineering. The deliverable of this thesis is part of a project, executed by six people, and includes two other theses. The project is embedded in a larger project by TU Delft, Erasmus Medical Center and Leiden University Medical Center. We would like to thank our teammates and supervisors for their support.

Erik Speksnijder
Ismail Bourhaeil
Delft, June 2021

Contents

1 Introduction

Epilepsy is a central nervous system disorder, characterized by seizures or periods of unusual behavior, sensations, and sometimes loss of awareness [1]. Seizure episodes are a result of excessive electrical discharges in a group of brain cells [2]. Symptoms range from strange sensations such as unusual smells or tastes to uncontrollable jerking or shaking and loss of consciousness [3].

In 2019, more than 60 thousand people in The Netherlands received medical treatment for epilepsy [4] and around 50 million people worldwide suffer from the disorder [2], making it one of the most common neurological diseases.

Depending on the duration and severity of the condition, epilepsy can impact the patient's quality of life, preventing things such as swimming or driving. Furthermore, people, living with epilepsy, may face social issues, such as stigma and discrimination.

Fortunately, treatment with appropriate anti-seizure medicines can often prevent or control seizures. However, 30% of the patients do not respond positively to anti-epileptic drugs prescribed to them [5].

In order for timely seizure prevention to become possible, single-site sensory recordings are not sufficient in order to permit in-time and robust identification of seizures: Epilepsy is a system-wide phenomenon which manifests physically across the body in various forms such as rapid muscle tone, sweating, elevated heart rate and synchronized neuron firing. Many of these aspects appear ahead of an epileptic incident. If combined together, these aspects are tell-tale signs of an imminent ictal event. Because of these observations, a wireless medical body area network (MBAN) has been proposed, which implements multimodal-data integration and closed-loop seizure suppression.

The design and implementation of the MBAN introduces several challenges, such as data collection, seizure prediction and suppression, secure pairing and communication and the design of a user friendly interface. This thesis will focus on the design and implementation of a secure bus architecture that connects multiple processing cores, a sensor and an actuator within an MBAN node, providing isolation between the modules as well as memory protection.

This chapter is structured as follows: First, Implantable medical devices (IMDs) will be discussed. Next the problem of the thesis will be defined, including the scope and bounds of the project. Finally, the structure of the thesis will be outlined.

1.1. Implantable medical devices (IMDs)

An implantable medical device (IMD) is a device which is introduced into the human body by clinical intervention. The operation of an IMD depends on an energy source other than one generated by the human body or gravity. Examples are cardiac pacemakers, implantable defibrillators, implantable neurostimulators, cochlear implants and implantable glucose monitors and infusion pumps.

In 1958, Åke Senning, a thoracic surgeon at the Karolinska Hospital in Stockholm, implanted the first IMD in the form of myocardial electrodes and a pulse generator with a rechargeable nickel-cadmium battery [6]. Since then, IMDs have evolved rapidly into reliable cures for chronic diseases, maturing in lifetime, functionality and effectiveness. This development has allowed for new ways of treatment, greatly improving patient's quality of life. The most notable improvement has emerged with the introduction of wireless communication, which allows for continuous monitoring of heart rate, insulin levels

etc. Additionally, health-care practitioners may request data or reprogram the IMD, allowing a change in the patient's treatment without the need for surgery.

While the wireless technology can greatly improve the delivery of a treatment, it also comes with potential hazards from a security perspective. The privacy and security concerns raise the need for a methodology which prevents unauthorized entities to access the IMD and control and access its components, such as sensors, actuators and memory.

1.2. Problem definition

The goal of this thesis is the design and implementation of a secure bus-architecture that connects multiple processing cores and other entities within an MBAN node, providing isolation between the modules as well as memory protection. The organization of the node is given in figure 1.1 and includes a sensor, actuator and memory as well as two processing cores: a main core (SiMS) and a security core (SISC). Additionally, the SISC may communicate with an external reader via a transceiver.

The deliverable will be a proof-of-concept implementation on an FPGA, demonstrating the functionalities and features of the design. Additionally the FPGA will have to be able to communicate wirelessly with a smartphone, for example via Bluetooth. Fabrication of a customer-oriented product falls outside the scope of the project. Nonetheless, the design choices will be made with the final application in mind.

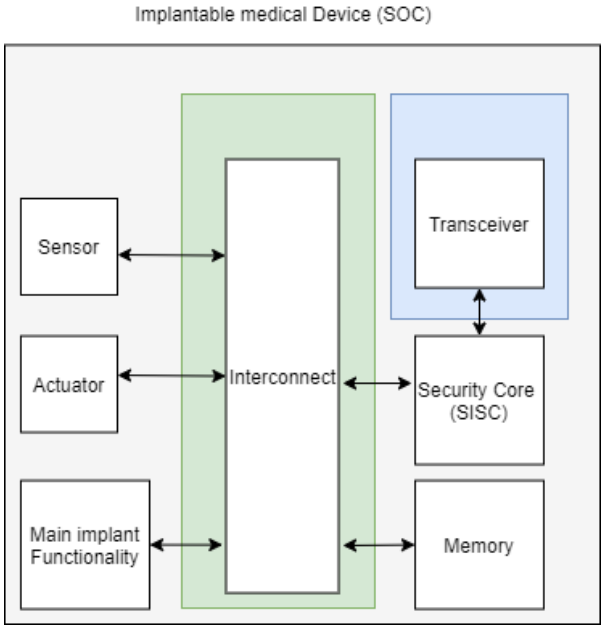


Figure 1.1: Internal architecture of an MBAN node

1.3. Thesis outline

This thesis is structured as follows: chapter 2 gives an analysis and overview of security vulnerabilities and potential cyber attacks. Chapter 3 describes the requirements of the prototype including the environmental conditions, functional requirements and performance indicators. Chapter 4 elaborates on the design of the prototype. Chapter 5 discusses the implementation and verification of the design. Finally, chapter 6 gives a conclusion and recommendations for future work.

2

Threat modeling

Threat modeling is a process occupied with identifying potential threats and vulnerabilities of the system. In this chapter we will focus on security threats associated with the communication between the IMD and an external reader. First, the common attacks, described in [7], will be summarized. Next, the adversary model will be defined. Finally, laws and regulations regarding data protection will be discussed.

2.1. Common attacks

- **Eavesdropping:** Eavesdropping, also known as sniffing or snooping, occurs when an attacker intercepts data that is transmitted between two devices. The standard defense against eavesdropping is the use of cryptography to hide information[8].

- **Modification:** A modification attack occurs when an attacker alters a portion of the message, by changing, inserting or deleting data. A modification attack can be prevented with digital signatures, hash functions or message authentication codes.

- **Replay:** A replay attack is an attack in which a message from an authorized user is intercepted and repeated. Several countermeasures exist such as tagging the message with a session ID or using a MAC with a randomly generated nonce.

- **Denial of service:** A denial of service attack is an attack in which a malicious actor aims to render a device or network resource unavailable to its intended user by disrupting services. This is typically achieved by flooding the device with requests.

- **Cryptanalysis:** Cryptanalysis is occupied with exploring weaknesses or leaks of information in a cryptographic system. Cryptanalysis typically aims to take advantage of mathematical weaknesses, but also includes side-channel attacks, which depends on extracting information from the physical system, used for encryption.

- **Birthday attacks:** The birthday attack is a brute-force method and relies on the higher probability of collision between random attack attempts and a fixed degree of permutations. Typically, a message m is accompanied with a digital signature $f(m)$. Mathematically, the objective of a birthday attack is to find a fraudulent input m' such that $f(m) = f(m')$. If such a pair is found, the attacker presents the fair contract to the victim. After the victim has signed the fair contract, the attacker attaches the signature to the fraudulent version. Birthday attacks can be prevented by making the digital signature long enough such that the attack becomes computationally infeasible.

2.2. Adversary model

An adversary model is a formalization of an attack on a device, network or protocol [9]. The adversary can be classified into two categories: a passive adversary and an active adversary.

- **Passive adversary:** A passive adversary refers to an attacker that is able to execute any malicious operation that does not involve generation or modification of a message. These operations typically involve eavesdropping.
- **Active adversary:** An active adversary refers to an attacker that has full control over the communication channel allowing them to eavesdrop, modify, drop and replay messages.

During the design of the communication protocols in section 4 we will assume an active attacker. This assumption is justified considering the security requirements of the IMD [10].

2.3. Laws and regulations

The European parliament and council published The General Data Protection Regulation (GDPR) [11] in the official journal of the European Union, providing regulations on the processing and free movement of personal data. The regulation entered into force on the 24th of May in 2016 for all Member States of the European Union and applied from the 25th of May in 2018. Of particular relevance are articles 25, 32 and 35.

Article 25, paragraph 1 and 2 state the following.

1. ”The controller shall, both at the time of the determination of the means for processing and at the time of the processing itself, implement appropriate technical and organisational measures, which are designed to implement data-protection principles in an effective manner and to integrate the necessary safeguards into the processing.”
2. ”The controller shall implement appropriate technical and organisational measures for ensuring that, only personal data which are necessary for each specific purpose of the processing are processed.”

Article 32, paragraph 1 describes in more detail the regulations regarding the security of processing and states that, among others, the controller shall implement the following measures to ensure a level of security, as appropriate.

1. ”the pseudonymisation and encryption of personal data”
2. ”the ability to ensure the ongoing confidentiality, integrity, availability and resilience of processing systems and services”
3. ”the ability to restore the availability and access to personal data in a timely manner in the event of a physical or technical incident”
4. ”a process for regularly testing, assessing and evaluating the effectiveness of technical and organisational measures for ensuring the security of the processing”

Article 35, paragraph 1 describes regulations on data protection impact assessment and states that, prior to the processing, ”the controller shall carry out an assessment of the impact of the envisaged processing operations on the protection of personal data”.

The US Food and Drug Administration (FDA) issued nonbinding recommendations on management of cybersecurity in medical devices on the 28th of December in 2016 [12]. Among others, the FDA recommends using a cybersecurity vulnerability assessment tool for rating vulnerabilities and determining the need for and urgency of the response as well as having a process for assessing the severity of patient harm, if the cybersecurity vulnerability were to be exploited.

3

Program of requirements

As stated in section 1.2, the goal of this thesis is the design and implementation of a secure bus-architecture that connects the entities within an MBAN node. This chapter describes the functional requirements and performance indicators of the prototype as well as the software and hardware tools used during the design process.

3.1. Design tools

For the design and implementation, we will rely on dedicated software and hardware tools, produced to ease the design process. These design tools are listed below.

- The prototype will be implemented on an Altera DE1 development board.
- A Pmod BLE Bluetooth low energy interface will be used to communicate wirelessly between the FPGA and the smartphone.
- The Questasim software will be used to simulate the VHDL code.
- The Quartus II 13.0sp1 software will be used to program the FPGA.

3.2. Functional requirements

The deliverable described in section 1.2 should be able to perform certain functions or exhibit specific input-output behaviour. These requirements are given below.

- The prototype should connect 2 masters and 3 slaves.
- The masters and slaves should be able to communicate via an AMBA AHB-lite protocol.
- A memory protection unit (MPU) should deny accesses from unauthorized masters.
- The prototype should provide a secure protocol for updating the firmware of the MPU. This protocol should satisfy the security goals defined in section 3.3.
- The prototype should be able to communicate wirelessly with a smartphone.

3.3. Security goals

As stated in section 3.2, the prototype will provide a secure protocol for updating the firmware of the mpu. The corresponding security goals are described below.

- **Confidentiality:** Confidentiality is the ability to prevent unauthenticated parties from extracting information from exchanged data. This prevents unauthorized parties from obtaining sensitive patient data. Attacks aimed to impact the confidentiality are eavesdropping and cryptanalysis.

- **Authentication:** Authentication is occupied with proving the identity of the involved parties. This prevents an unauthorized party from updating the IMD with malicious firmware.
- **Integrity:** Integrity of a message is the ability to prevent malicious parties from changing the message without the trusted parties noticing. Similar to authentication, this prevents an attacker from injecting malicious firmware. Attacks aimed to impact integrity are modification and birthday attacks.
- **Availability:** Availability is the probability that the system performs its intended functionality satisfactorily at a given point in time. Considering the life-critical functionality of an IMD, a high availability is required during its lifetime. Attacks targeting availability are denial of service attacks.

3.4. Performance indicators

As described in section 1.2, the design choices and performance will be evaluated with the final product in mind. In this case, the performance indicators should adhere to requirements of implantable medical devices. These performance indicators are described below.

- **Area:** Area indicates the occupied physical space. IMDs typically operate under a constrained. environment. Therefore, the available area is limited. The area of IMDs are typically in the order of a few mm^2 [13].
- **Power consumption:** Power consumption indicates the amount of energy consumed in a certain time frame. IMDs typically operate under a constrained environment. In addition, the IMD should operate throughout a relatively long lifespan. This explains the need for low-power devices. These systems typically operate off of currents between $10 - 20\mu A$ at voltages between $1 - 2V$ [14], resulting in a power consumption between $10 - 40\mu W$.

4

Design

4.1. Overview

In order to meet the functional requirements described in section 3.2, the design of the interconnect is split into three modules, as shown in figure 4.1. The function of the AHB-Lite Bus is to regulate the data flow between the masters and the slaves according to the AMBA AHB-Lite protocol. The memory protection unit (MPU) indicates whether an access is allowed or denied. Finally, the function of the authentication manager is to execute the authentication protocol, used to provide authentication and message integrity when updating the MPU.

The modularity of the design allows separate design of the three modules, easing the design process as well as future changes or improvements. Because the authentication protocol, the implementation of the MPU and the bus interface are three important design choices that can be made independently, we believe these advantages outweigh the slight area overhead that may occur.

In addition to the design of the interconnect, the deliverable provides wireless communication with a smartphone.

The subsequent sections will elaborate on the design of the AHB-Lite Bus, the memory protection unit, the authentication manager and the wireless communication with the smartphone, respectively.

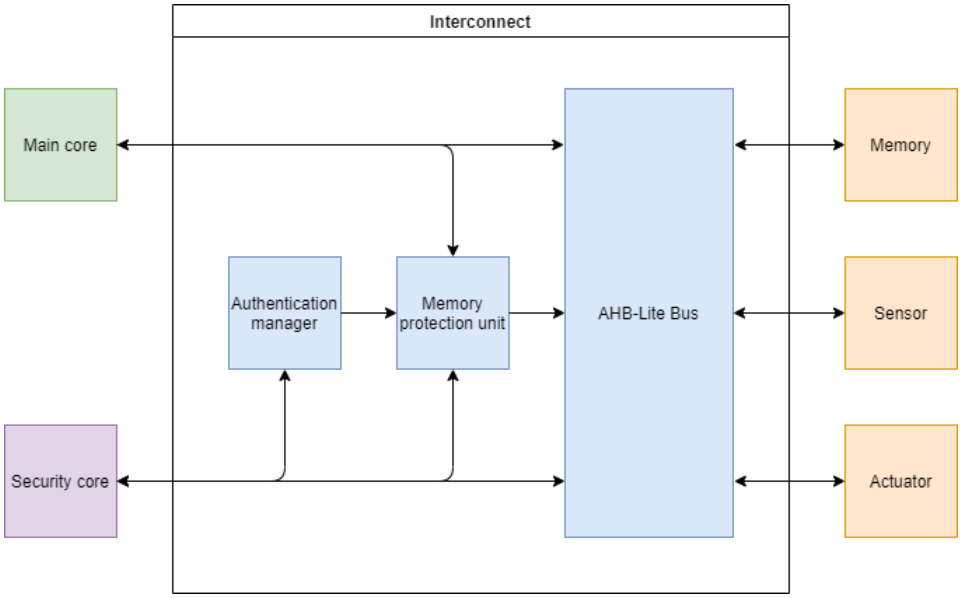


Figure 4.1: Structural overview of the interconnect, consisting of an AHB-Lite Bus, a memory protection unit and an authentication manager

4.2. AHB-Lite Bus

As described in section 3.2, the interconnect should connect 2 masters and 3 slaves. First, section 4.2.1 discusses the AHB-Lite protocol. Next, section 4.2.2 describes the design of the AHB-Lite Bus. After that, section 4.2.3 explains the decoder of the AHB-Lite Bus. Finally, section 4.2.4 describes the arbiter of the AHB-Lite Bus.

4.2.1. AHB-Lite protocol

AMBA AHB-Lite is a protocol designed by ARM for high performance synthesizable designs. The important signals included in the protocol are shown in table 4.1. A detailed description of the protocol is given below.

Signal	Bits	Description
Hwrite	1	Indicates operation (e.g. read or write)
Htrans	2	Determines the transfer type
Hready	1	Indicates a completed transfer
Hresponse	1	Indicates an error
Hsel	1	Selects slave

Table 4.1: Signals provided by the AHB-Lite protocol

The signal HWRITE, which is given by the master, controls the direction of the data flow:

- If HWRITE is HIGH, the operation is a write operation, and data flows from the master to the slave
- If HWRITE is LOW, the operation is a read operation, and data flows from the slave to the master

The signal HTRANS is used to classify the transfer. HTRANS has the following encoding:

HTRANS[1:0]	Type	Description
00	Idle	The master does not request a transfer
01	Busy	Used to insert wait states by the master during bursts
10	Nonseq	Indicates a single transfer or the first transfer of a bust
11	Seq	Indicates the remaining transfers of a burst

Since Bursts are not supported in this implementation because of area constraints, HTRANS will only take on the values 00 and 10. In this case HTRANS can be viewed equivalent to a VALID signal used in other protocols.
The two types of transfers supported are a single read and a single write.
An AHB-Lite transfer includes two phases:

- The address phase
- The data phase

This imposes a pipelined nature on the design of the bus, due to the overlapping nature of data and address phases between successive transfers. This is illustrated in figure 4.2.

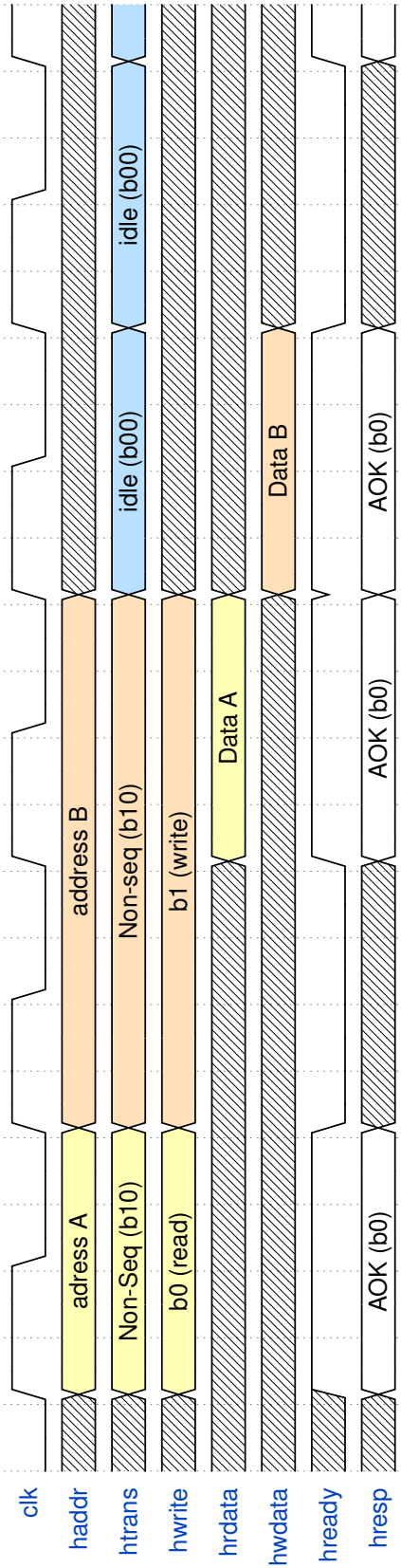


Figure 4.2: Interface between an interconnect and a master performing a read transfer followed by write transfer

Figure 4.2 shows the interface between an interconnect and a master performing a read transfer from address A, followed by a write transfer to address B.

1. During the first clock cycle, the signal HREADY is high, indicating to the master that it should execute the address phase of the first transfer by sending its desired address value via HADDR and giving HWRITE a value of 0 and HTRANS a value of 10 to indicate a valid transfer.

2. During the second clock cycle, the master immediately starts the address cycle of its desired write transfer. However, the interconnect inserts a wait state by giving HREADY a value of 0, indicating that the master should wait.

3. During the third clock cycle, the interconnect sends the master the data from address A, and sets HREADY high. HREADY having a high value means that the data in HRDATA is valid, and that the interconnect accepts the address phase of the write operation. This exemplifies the pipelined nature of the process.

4. During the fourth clock cycle, the interconnect pulls ready high, indicating that it is ready to accept the data phase of the write transfer and the address phase of any possible new transfer. The master then sends the data to be written into address B.

5. During the fifth clock-cycle, the interconnect inserts a wait state.

6. During the sixth clock-cycle, the interconnect sets HREADY high, indicating that the write transfer has been completed successfully.

The interface between the master and the interconnect is similar to the interface between the interconnect and the slave. The only difference is that the interconnect provides a signal HSEL to each slave, which is pulled high when a slave is selected.

4.2.2. Overview of the AHB-Lite Bus

Arm proposes the topology shown in figure 4.3 for designing an AHB-Lite Bus with multiple masters [15].

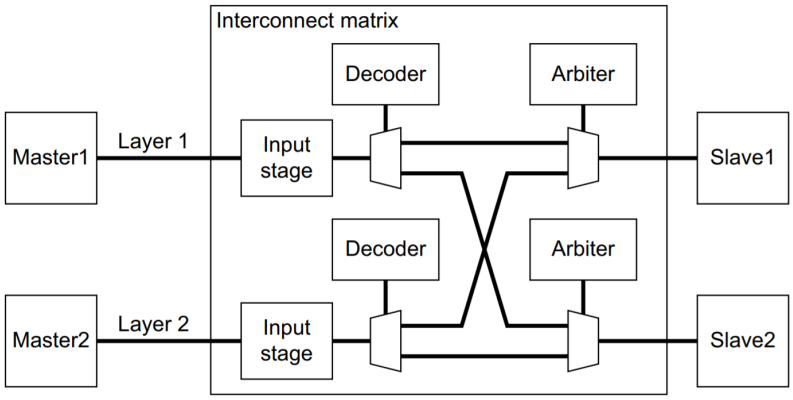


Figure 4.3: Proposed design of a multi-master AHB-Lite interconnect by Arm [15]

In figure 4.3:

- The decoder determines which slave is requested by each master
- The arbiter decides which master should get access to a slave.

After slightly modifying the topology, shown in figure 4.3, by removing the demultiplexers below the decoders, and going in detail into the data path and control signals inside of the bus, the design in figure 4.4 was reached.

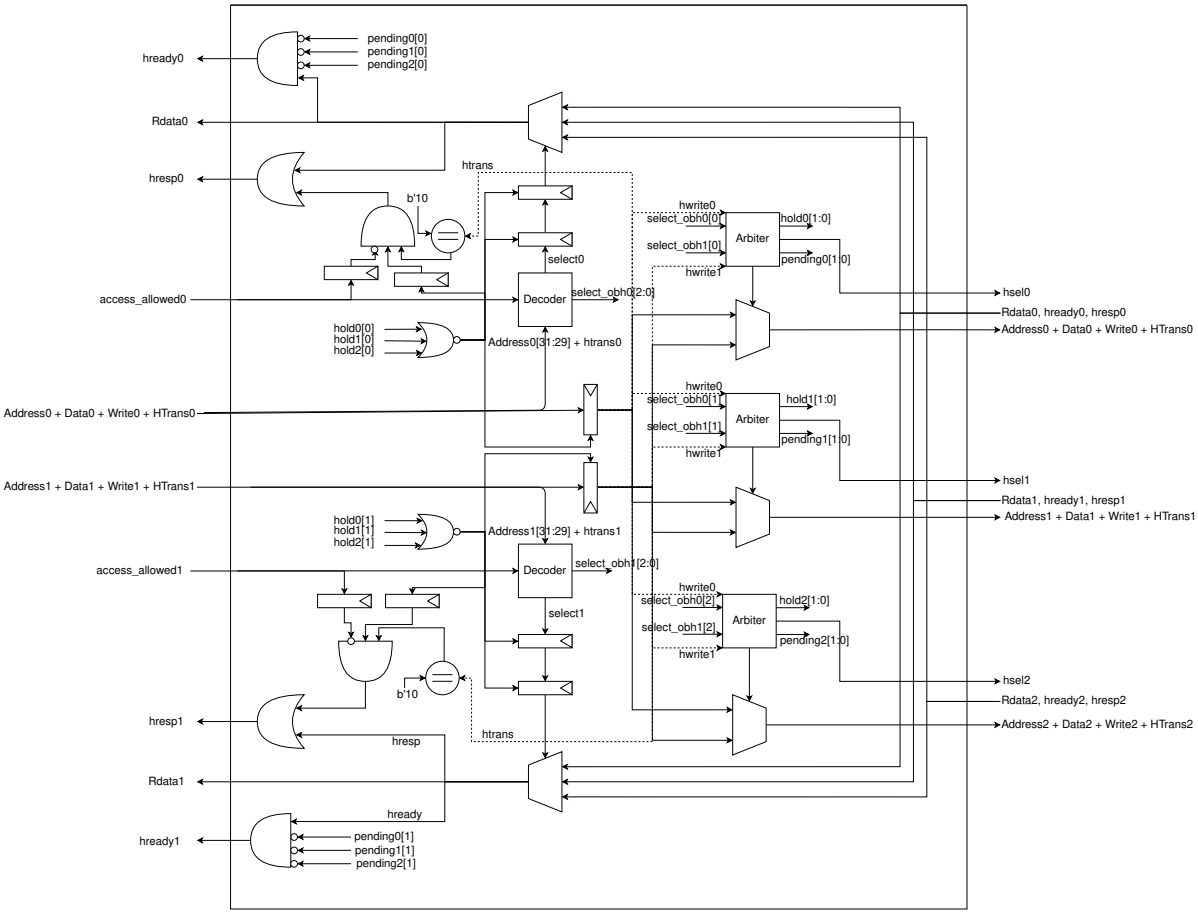


Figure 4.4: Schematic of the designed AHB-Lite Bus

4.2.3. Decoder

Each of the two decoders in figure 4.4 decode the accesses of one of the masters, determining which slave is requested. An important remark is that the decoders are designed to drive all their outputs low if the signal access allowed coming from the MPU is low.

The input and output signals of the decoder are shown in table 4.2 The signal *address_3msb* contains the three most significant bits of the address, which are used to determine which slave is accessed. The signal *valid* comes from the MPU and indicates whether the access is allowed. The signal *select* controls the multiplexer that is connected to the master. Finally, the signal *select_obh* selects the requested slave by pulling the respective bit of the arbiter high.

The decoder is implemented as a simple combinatorial circuit.

Signals	Input/Output	Bits	Description
Address_3_msb	Input	3	Three most significant bits of the address that determine which slave is accessed
Valid	Input	1	Signal from the mpu that indicates whether the access is allowed
Select	Output	2	Controls multiplexer that is connected to the master
Select_obh	Output	3	Selects the requested slave by pulling the respective bit of the arbiter high (one bit hot)

Table 4.2: Input and output signals of the decoder

4.2.4. Arbiter

Each of the three arbiters regulate accesses to a specific slave by providing the relevant control signals of the AHB-Lite Bus. The arbiter should be able to regulate collisions such as two masters targeting

the same slave and starting the address phase of a request during the same clock cycle, or one master starting its address phase during the data phase of the other. To account for different scenario's, the arbiters are implemented as an FSM.

The arbiters are designed to always prioritize requests from master 0, the main implant core. This is done to ensure availability of the life-critical functions performed by the main core, even in case of DoS attack performed through master 1, which is connected to the outside world via a transceiver.

Input-output behaviour

The inputs and outputs of the arbiter are shown in table 4.3. The signals *master0_select* and *master1_select* indicate a request from master 0 or master 1, respectively, while the signals *write_0* and *write_1* indicate the corresponding operation (e.g. read or write). The signal *select_mux* controls the multiplexer connected to the slave. The signals *pending0* and *pending1* drive the *hready* signal of master 0 and master 1, respectively, while the signals *hold0* and *hold1* stall the respective transfers. Finally, the signal *hsel* indicates the corresponding slave is being selected.

Signal	Input/Output	bits	Description
master0_select	Input	1	Indicates a request from master 0
master1_select	Input	1	Indicates a request from master 1
write_0	Input	1	Indicates operation of master 0 (e.g. read or write)
write_1	Input	1	Indicates operation of master 1 (e.g. read or write)
select_mux	Output	2	Controls multiplexer connected to the slave: 00 -> select master 0, 01 -> select master 1, 10 -> drive output low
pending0	Output	1	Drives the hready signal of master 0
pending1	Output	1	Drives the hready signal of master 1
hold0	Output	1	Stalls the transfer of master 0
hold1	Output	1	Stalls the transfer of master 1
hsel	Output	1	Select signal of the slave

Table 4.3: Input and output signals of the arbiter

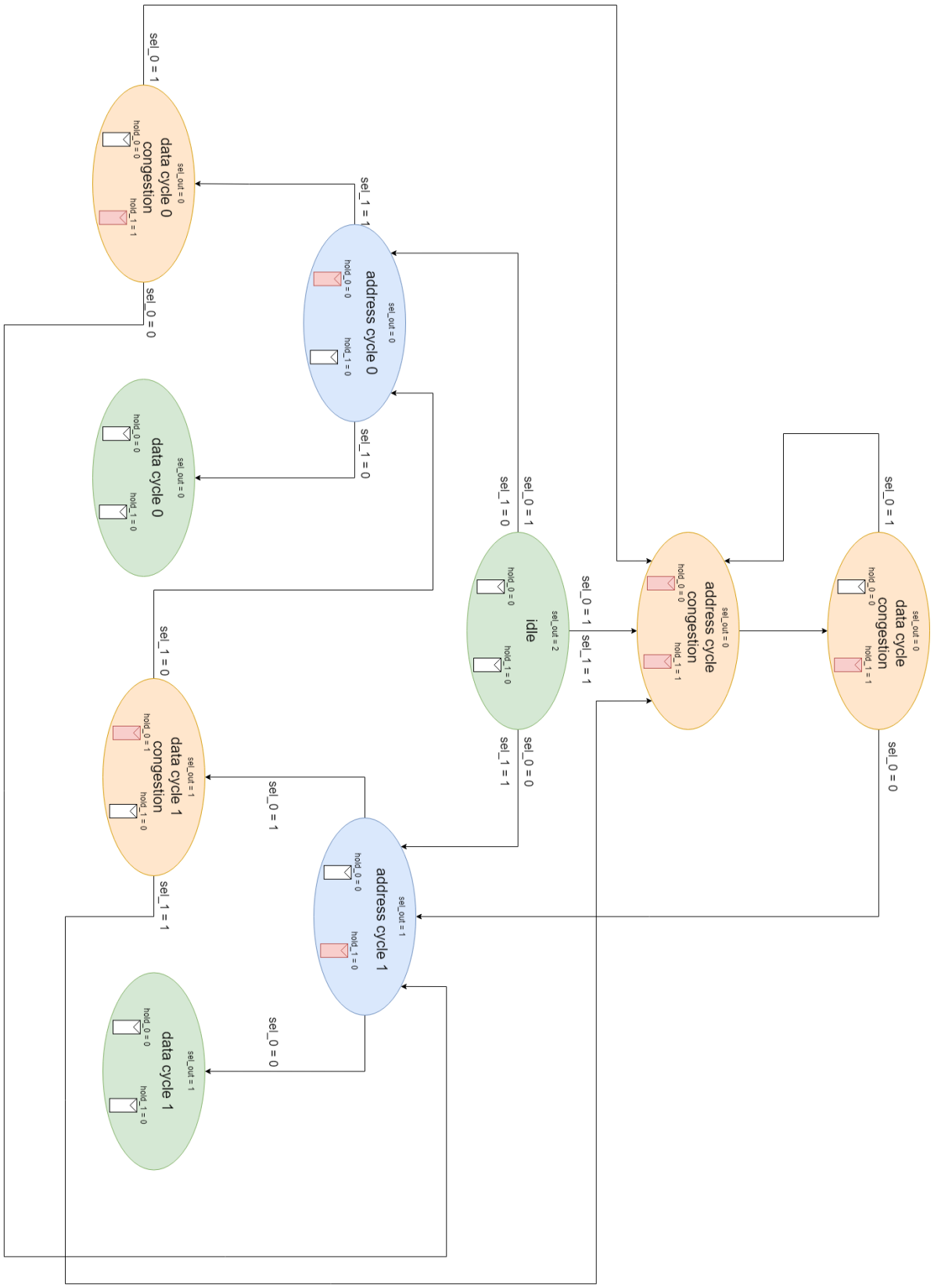


Figure 4.5: State diagram of the arbiter of the AHB-Lite Bus

Implementation

The state-diagram of the arbiter is shown in figure 4.5. In the diagram, blue states indicate a pending information transfer with no congestion (e.g. two masters accessing the same slave). Red states indicate a congestion and force one master to wait. Finally, green states are states with no pending transfer. Transitions starting from these green states are all similar to those from the idle state. To provide improved readability, the transitions starting from the states ”data cycle 0” and ”data cycle 1” were omitted from the diagram.

Furthermore, in figure 4.5, each state has two buffers drawn inside it, which symbolize the input stage buffers in figure 4.4. A buffer, coloured red, holds an address phase of its corresponding master, while a buffer, coloured white, does not.

Starting from the idle state, three scenario’s are possible: a request from either master 0 or master 1 or a request from both masters at the same time. The flow of control, following these three scenario’s, is described below.

1. An address phase is received from master 0, but no address phase is received from master 1:

   - The arbiter transitions into the ”address cycle 0” state in which it transfers the AHB-Lite address and control signal from master 0 to the selected slave
   - If no address phase is received from master 1, the arbiter goes to the green state ”data cycle 0” in which the data cycle is transferred from master 0 to the slave.
   - If an address phase is received from master 1 in the ”address cycle 0” state, the arbiter transitions into the state ”data cycle 0 congestion”, in which the data cycle of master 0 is transferred to the slave and the address phase of master 1 is saved.

     From this state, if an address phase from master 0 is received, the arbiter prioritizes it over the saved address phase of master 1, and transitions to the ”address cycle congestion” state. Otherwise, the arbiter simply transitions to the ”address cycle 1” to handle the saved address phase of master 1.

2. An address phase is received from master 1, but no address phase is received from master 0

   - The arbiter transitions into the ”address cycle 1” state in which it transfer the AHB-Lite address and control signal from master 1 to the selected slave
   - If no address phase is received from Master 0, the arbiter goes to the green state ”data cycle 1” in which the data phase is transferred from master 1 to the slave.
   - If an address phase is received from master 0 in the ”address cycle 0” state, the arbiter transitions into the state ”data cycle 1 congestion”, in which the data phase of master 1 is transferred to the slave and the address phase of master 0 is saved .

     From this state, if an address phase from master 1 is not received, the arbiter transitions to the ”address cycle 0” state to process the saved address phase of master 0. Otherwise, if an address cycle is received from master 1, the arbiter transitions to the ”address cycle congestion” to handle the saved address cycle of master 0 first, and save the address cycle received from master 1.

3. An address phase is received from both master 0 and 1 at the same time:

   - The arbiter transitions into the ”address cycle congestion” state in which it transfers the AHB-Lite address and control signal from master 0 to the selected slave and saves the address phase of master 1.
   - After that, the arbiter transitions into the state ”data cycle congestion” in which the data cycle is transferred from master 0 to the slave and the address cycle of master 1 is saved for another cycle.
   - If an address phase is received from master 0 in the ”data cycle congestion” state, the arbiter prioritizes it over the saved address phase of master 1 and transitions back into the state ”address cycle congestion” to process it. Otherwise, the arbiter transitions into the state ”address cycle 1” to process the saved address cycle of master 1.

As shown in figure 4.5, sel_out has a value of 2 in the idle state. This ensures that the signals coming from a master are only sent to a slave if the corresponding decoder gives the signal ”select[2:0]” a value other than 0 (as this is the only way to put an arbiter in a state other than idle). Therefore, requests are sent from a master to a slave if and only if the corresponding ”access_allowed” signal is high. This means that, by design, there is no way to bypass the MPU in order to access a slave.

The control signal pending is omitted from the state diagram in figure 4.5 to improve readability. As described above, this control signal is used to drive the hready signals at the interfaces with master 1 and 0, as shown in figure 4.4. The arbiter calculates the value of the signal based on its current state and the value of hwrite at the output of the input stage, which means the arbiter can be regarded as a Mealy Machine. Although this creates a direct connection between hwrite and hready at the output, it shouldn’t lead to glitches in the hready output signal as hwrite is buffered and is not the hwrite signal coming from the input of the AHB-bus.

4.3. Memory protection unit

As described in section 3.2, the purpose of the memory protection unit is to deny accesses from unauthorized masters. This can be achieved through several implementations. The associated design choices are made on the basis of the performance indicators described in section 3.4. The subsequent subsections will discuss the input-output behaviour and the design choices, respectively.

4.3.1. Input-output behaviour

The black-box model of the memory protection unit is shown in figure 4.6. Both masters send two input signals: the address and the operation (e.g. write or read). At the output, the MPU gives the access rights of both masters (e.g. access permitted or denied). Additionally, the authentication manager sends the new table and a valid signal which indicates whether the MPU should be updated with the new table.

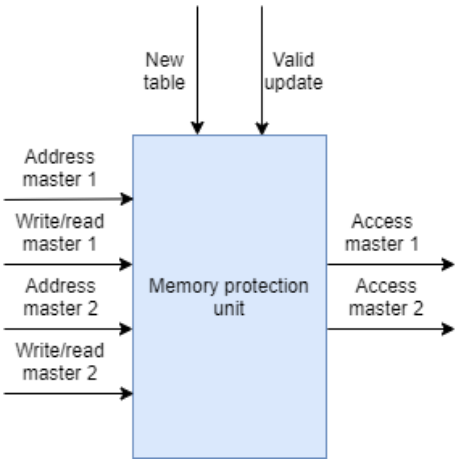


Figure 4.6: Block-box model of the memory protection unit including inputs and outputs

4.3.2. Design

The memory protection unit consists of two building blocks: a protection table, indicating the access rights of both masters and a combinatorial circuit that compares the input signals with the protection table.

During the design of the memory protection unit, two implementations have been considered. These implementations are discussed below.

- In the first implementation, every row of the protection table contains an address range, indicated by a start and end address, as well as the access rights for both writes and reads of both masters.

  In this implementation every row of the protection table contains 64 bits of memory for the addresses as well as 4 bits for the access rights. The combinational circuit includes two comparators per row for both masters, adding up to four 32-bit comparators per row.

- In the second implementation, every row of the protection table corresponds to a fixed address range and only contains the access rights of both masters. The fixed address ranges correspond to the $log_2(n)$ most significant bits of the address, where n is the number of rows.

  In this implementation, the protection table only contains four bits of memory per row. Additionally, the combinational circuit only contains some multiplexing logic to select the right protection table entry, essentially saving four 32-bit comparators per row. However, the increased efficiency in terms of hardware-cost comes at the expense of limited flexibility: the address ranges are fixed.

According to [10], the second approach saves almost 2000 LUTs for a protection table with 8 rows, as well as 528 bits of memory. Given the limited area of IMDs, this is significant overhead. Furthermore, the smaller protection table requires a smaller message during a firmware update, reducing the complexity of the encryption module described in section 4.4.2. On the other hand, the limited flexibility is not a big problem, since the physical address space of IMDs is typically in the order of hundreds of kilobytes [10]. Therefore, the second implementation was used for our prototype.

4.4. Authentication manager

The purpose of the authentication manager is to provide a secure protocol for updating the memory protection unit. The protocol should achieve the security goals defined in section 3.3. First, section 4.4.1 describes the protocol. Next, section 4.4.2 explains the design of the encryption module. After that, section 4.4.3 discusses the generation of the random nonce, used by the protocol. Next, section 4.4.4 explains the control fsm of the authentication manager. Finally, section 4.4.5 describes the structural architecture of the authentication manager.

4.4.1. Protocol

During the manufacturing of the implant chip, a key *K* will be added by the manufacturer and shared in a secure manner with trusted entities. This key can be used by the trusted entities to securely update the protection table of the MPU.

The protocol is shown in figure 4.7 and includes the following steps.

1. The reader initiates the authentication protocol by writing into a specific address.

2. The implant generates a fresh nonce N_I and sends it to the reader. To prevent replay attacks, the nonce is randomly generated every time the protocol is executed.

3. The reader generates its own random nonce N_R and encrypts this nonce together with the command Cmd. Furthermore, to provide authentication, the reader generates a MAC by encrypting the implant nonce N_I together with the encrypted command $[Cmd]_K$. Next, the encrypted nonce and command are sent to the implant along with the MAC.

4. Finally, the implant checks the received MAC by decrypting it and comparing it to a local version of the MAC. If the equality is satisfied, the implant decrypts the command Cmd and reader nonce N_R, updates the mpu and sends the reader nonce N_R back to the reader. If the validation fails, the protocol is aborted and the implant sends an error message.

An important remark is that the authentication manager is not directly connected to the transceiver. Instead the SISC is placed between the authentication manager and the transceiver and relays the exchanged data between the receiver and the authentication manager.

4.4.2. Encryption

The purpose of the encryption module is to encrypt plain texts or decrypt cipher texts. This section will subsequently discuss the design choices, the cipher itself, the input-output behaviour and the implementation.

Design choices

The encryption module permits several choices, regarding both the encryption methodology as well as the implementation. These design choices are discussed below.

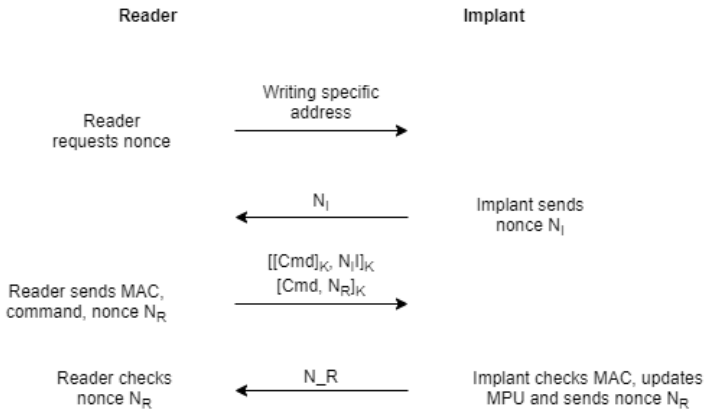


Figure 4.7: Protocol used by the authentication manager if the reader wants to update the MPU

- **Symmetric vs asymmetric key:** Cryptographic systems can be subdivided into two classes: symmetric and asymmetric key cryptography. Symmetric key cryptography uses the same pre-shared key for both encryption of the plaintext and decryption of the ciphertext. Asymmetric key cryptography uses pairs of keys: a public key, which may be known by others, and a private key, which is known by the owner only. Security of asymmetric key cryptography is based on one-way functions: functions that are impossible or computationally infeasible to invert. The advantage of asymmetric key cryptography is the lack of need for a pre-shared key. However asymmetric ciphers are computationally expensive [16], resulting in larger hardware cost, and have a larger execution time [17]. As described in section 3.4 area is an important constraint. Furthermore, the pre-sharing of the key can easily be done by encoding the key in the implant during fabrication. For these reasons, a symmetric cipher was implemented.

- **Lightweight block cipher:** To choose an appropriate lightweight block cipher, which falls under symmetric encryption, some literature research was conducted on papers comparing power consumption and hardware-cost of several lightweight block ciphers. To this end [18], [19], [20], [21] and [22] have been examined. From these papers [19], [20], [21] and [22] compare area, while [18] and [22] compare power consumption. The results vary among these papers. In terms of hardware-cost [19] and [20] indicate Simon as the best cipher, while [18] indicates that Simon has the lowest power consumption. The other papers indicate a slightly better performance for other ciphers. Nonetheless, Simon consistently scores among the top choices in terms of both hardware-cost and power consumption. For this reason, we decided to implement the Simon block cipher.

- **Parallel vs serial implementation:** As will be explained, the Simon cipher executes its encryption algorithm in several rounds. This offers a design choice regarding the parallelism of rounds, which can be accomplished by loop-unrolling. The degree of parallelism r_p can range from 1 to the number of rounds and offers a throughput/area trade-off. According to [23] full loop-unrolling increases the area by approximately 8000 LUTs when implemented on the FPGA, while the execution time is decreased by a factor r_p. Given the area constraints of the IMD, this is is significant overhead. Additionally, the increased execution time is not important, given the limited frequency of firmware updates. We therefore decided to implement a serial implementation.

- **Mode of operation:** A block cipher is only suitable for secure encryption of a message of fixed length. A mode of operation describes a procedure to securely encrypt messages, larger than a block, by repeated application of a block cipher. In this case, the message length is fixed and small enough to fit in a single block. Therefore, we can simply encrypt the message, using the block cipher, without the need for a sophisticated mode of operation.

- **Block/key size:** During the design two implementations were considered with regards to block and key size: 32-bit block size/64-bit key size and 64-bit block size/96-bit key size. This design choice relies on a trade-off between area and security. [22] shows a roughly 20% larger area for

the 64-bit block size implementation. On the other hand, the 32-bit block size implementation is more susceptible to brute-force attacks, such as birthday attacks, as well as cryptananalytic attacks [24]. Given the requirements, described in section 3.4, the area overhead of the 64-bit block size implementation is acceptable and justified by the security aspects. Furthermore, a 64-bit block cipher allows us to encrypt the 64-bit MAC without the need for a sophisticated mode of operation. We therefore decided to implement the version with a 64-bit block size and 96-bit key size.

The Simon cipher

The Simon encryption procedure consists of two phases: key scheduling and encryption in several rounds. During key scheduling, all round keys are subsequently generated from the master key. During encryption, the round function is repeatedly applied using the previously generated round keys. A RAM is used to store the round keys. The SIMON64/96 implementation uses 42 rounds.

The round function uses the following operations.

- Bitwise XOR: $\oplus$
- Bitwise AND: &
- Left circular shift by i bits: S^i

The round function performs the operation described by equation 4.1, where l_i and r_i are the 32 most significant and 32 least significant bits of the block during round i, respectively, and k_i is the key, used during round i. The round function is shown in figure 4.8.

$$(l_{i+1}, r_{i+1}) = R(l_i, r_i, k_i) = (S^1(l_i) \& S^8(l_i) \oplus S^2(l_i) \oplus r_i \oplus k_i, l) \tag{4.1}$$

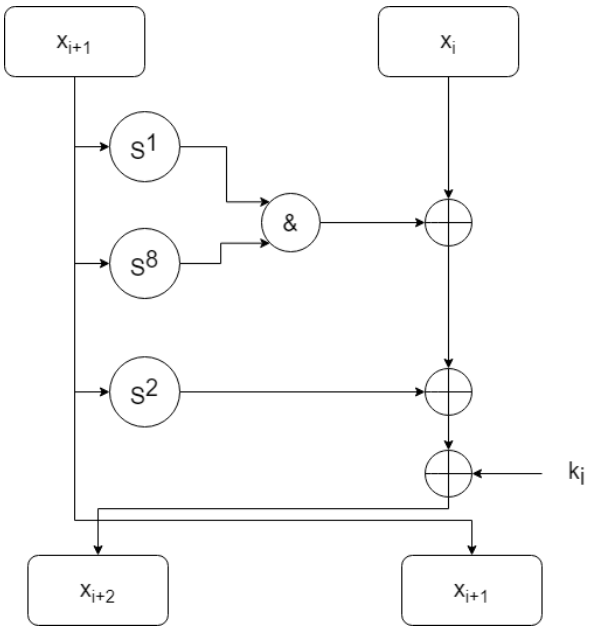


Figure 4.8: Round function of the Simon block cipher

The key scheduling uses the following operations.

- Bitwise XOR: $\oplus$
- Right circular shift by i bits: S^i

The key scheduling performs the operation described by equation 4.2, where k_i is the key, generated during round i, and c and z are predefined constants given by equations 4.3 and 4.4. The key scheduling is shown in figure 4.9.

$$k_{i+3} = K(k_{i+2}, k_{i+1}, k_i, c, z_i) = S^{-4}(k_{i+2}) \oplus S^{-3}(k_{i+2}) \oplus k_i \oplus c \oplus z_i \tag{4.2}$$

$$c = 0xFFFFFFFC \tag{4.3}$$

$$z = 10101111011100000011010010011000101000010001111110010110110011 \tag{4.4}$$

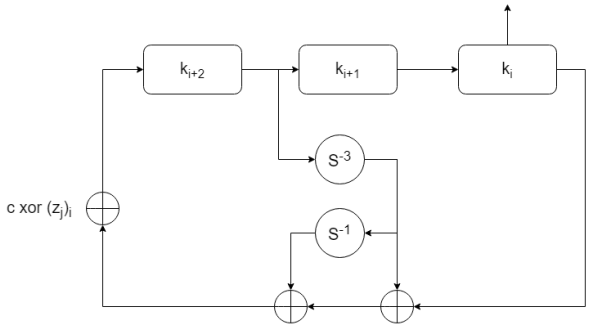


Figure 4.9: Key scheduling of the Simon64/96 block cipher

Input-output behaviour

The input and output signals of the encryption module are shown in table 4.4. Note that the signal *i_data* is 32 bits in contrast to the 64-bit block size. The reason is that the security core sends 32-bit data. The signal *i_command* indicates whether this data should be written into the upper or lower register. Furthermore, this signal is used to start encryption or decryption. In addition to the signal *o_data*, the encryption module also includes the signal *o_ready*, indicating whether the encryption or decryption has finished.

signal	input/output	bits	description
i_key	input	96	Key used by the block cipher
i_data	input	32	Data that is loaded into the register of the round function
i_command	input	3	000 -> wait, 001 -> write lower, 010 -> write upper, 011 -> start decryption, 100 -> start encryption
o_data	output	64	Encrypted or decrypted data
o_ready	output	1	Indicator of a finished encryption or decryption

Table 4.4: Input and output signals of the encryption module

Implementation

Encryption consists of repeated applications of the round function with the respective round key. Due to the nature of the key scheduling and round function, they can be run in parallel by simply bypassing the RAM and connecting the output of the key scheduling to the input of the round function.

Decryption consists of repeated application of the round function with the round keys in reverse order. The dependency on previous keys during generation of the round keys does not allow for parallel application of the key scheduling and round function. This dictates the use of a subsequent key scheduling and decryption phase, as well as the use of a RAM for key storage.

The state diagram of the corresponding control FSM is shown in figure 4.10. When encryption or decryption starts, the key is first loaded into the registers of the key scheduling. Next, the parallel or subsequent application of the key scheduling and round function is executed. When the procedure is finished, the FSM returns to the idle state. Additionally, the control FSM acts as a decoder for the signal *i_command*, during the idle state, to allow the round function to write the input data in either the upper or lower register.

The structural architecture of the encryption module is shown in figure 4.11. The internal signals of this module are shown table 4.5. The signal round key contains the key used by the round function during encryption or decryption. The signal iteration indicates the iteration number during encryption

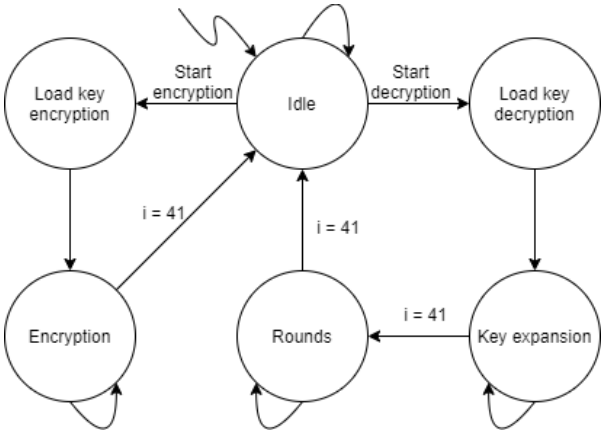


Figure 4.10: State diagram of the control FSM used by the encryption module

or decryption. The select signal of the mux determines whether the RAM is connected to the round function or bypassed by the key scheduling. The select signal of the key scheduling determines whether the input key or the operation described by equation 4.2 is connected to the registers inside the key scheduling, while the enable signal enables these registers. The select signal of the round function determines whether the input data or the operation described by equation 4.1 is connected to the registers inside the round function, while the signals enable1 and enable2 enable the lower and upper register, respectively. Finally, the signals enable, reset and ready of the counter enable the counter, reset the counter to zero and indicate that the final iteration has been reached, respectively.

signal	bits	description
round key	32	key used by the round function during encryption or decryption
iteration	6	indicates the iteration number during encryption or decryption
select mux	1	selects whether RAM is connected to the round function or bypassed
select key scheduling	1	determines whether the input key or the operation described by equation 4.2 is connected to the registers inside the key scheduling
enable key scheduling	1	enables the registers inside the key scheduling
select round function	1	determines whether the input data or the operation described by equation 4.1 is connected to the registers inside the round function
enable1 round function	1	enables the lower register inside the round function
enable2 round function	1	enables the upper register inside the round function
enable counter	1	enables the counter
reset counter	1	resets the counter to zero
ready	1	indicates the final iteration has been reached

Table 4.5: Internal signals of the encryption module

4.4.3. Nonce generation

As described in section 4.4.1, the protocol requires generation of a random fresh nonce. To this end a pseudo-random number generator has been implemented. Several cryptographically secure pseudo-random number generators have been designed, such as Microsoft's Cryptographic Application Programming Interface function CryptGenRandom, the Yarrow algorithm and Fortuna. However, we decided to use a block cipher running in counter mode as described in [25]. The advantage of this method is that it allows reuse of the encryption module, thereby saving area.

The operating principle is shown in figure 4.12. The static nonce, a random number encoded during fabrication, is concatenated with a counter that increases every time a fresh nonce is requested and given as an input to the block cipher. The output is used as the fresh nonce.

This procedure uses a simple deterministic function as input of the block cipher. However, ciphers are specifically designed to hide patterns in the input. The properties of the cipher will therefore ensure the randomness of the output and any problems with this procedure are generally considered a

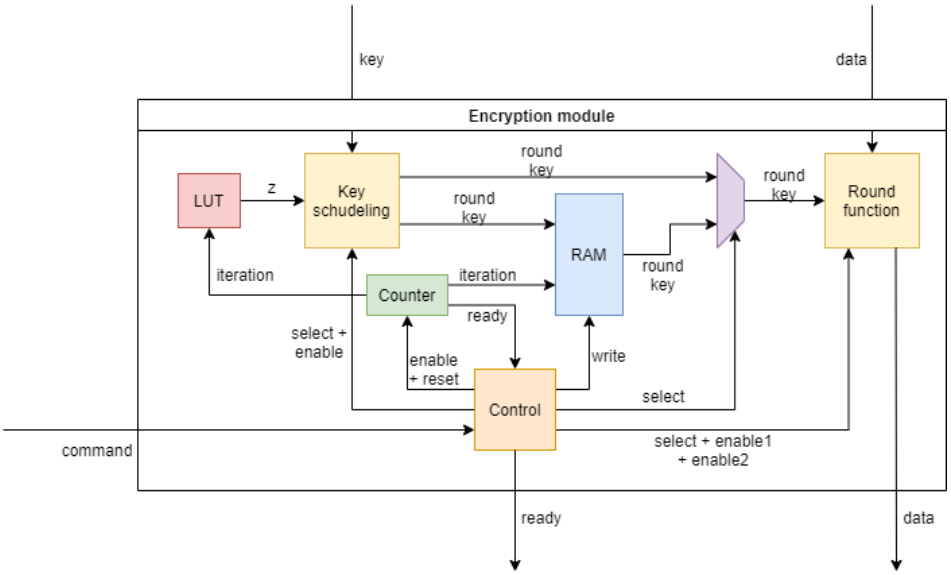


Figure 4.11: Structural architecture of the encryption module

weakness of the block cipher [26].

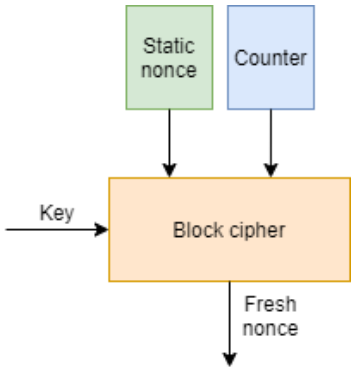


Figure 4.12: Nonce generation using a block cipher in counter mode

4.4.4. Control FSM

The state diagram of the control FSM of the authentication manager is shown in figure 4.13. As stated in section 4.4.2, the SISC sends 32-bit data. This explains why writing the lower and upper register is done in separate states. Additionally, there is a slight modification in the order of the steps described in section 4.4.1. Namely, the reader nonce is sent after checking the MAC. This allows us to write the reader nonce immediately into the lower register of the encryption module, saving one 32-bit register.

The state diagram performs the following procedure. After the protocol is initiated, the static nonce and counter value are subsequently loaded into the upper and lower register, respectively. Next, the nonce, N_I, is generated by the encryption module. Once the encryption procedure is finished, the nonce is stored and subsequently sent to the SISC. After that, the encrypted command, Cmd, and MAC are sent by the reader. Next, the MAC is decrypted and validated. If the equality is not satisfied, an error message is sent and the protocol is aborted. If the comparison results in a match, an authentication message is sent to the SISC to which the SISC replies by sending the encrypted reader nonce, N_R. After that, the command and reader nonce are decrypted. When the decryption procedure is finished, the MPU is updated and the reader nonce is sent back to the reader.

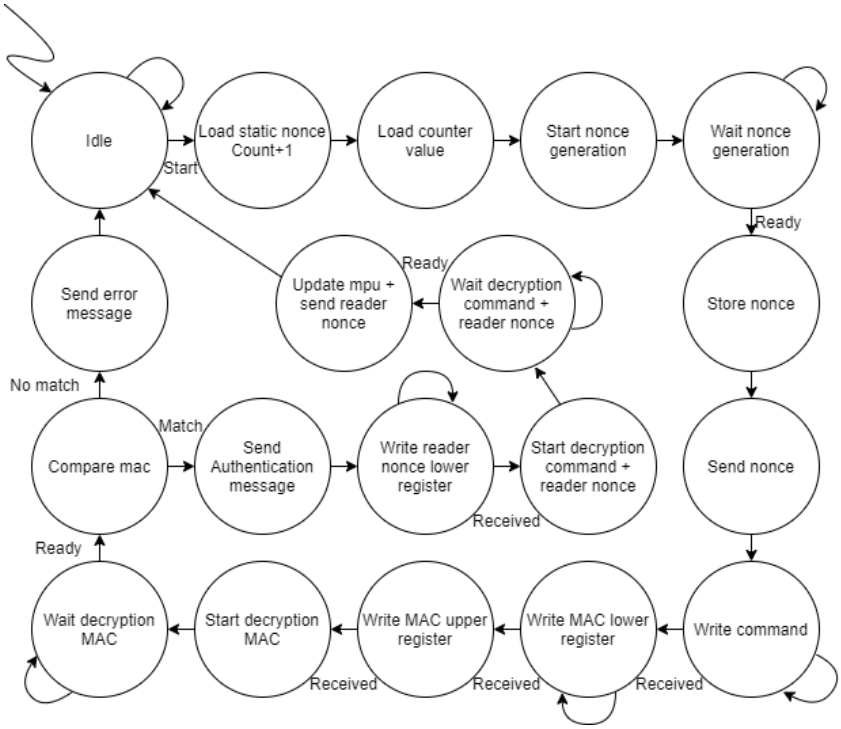


Figure 4.13: State diagram of the control fsm used by the authentication manager

4.4.5. Implementation

The input and output signals of the authentication manager are shown in table 4.6. The input signals are the signal *Sisc writedata*, which contains the data sent by the SISC, and the four most significant bits of the address, which are used to initiate the protocol. At the output, the authentication manager gives the new table of the MPU as well as a valid bit which indicates whether the MPU should be updated. Furthermore, the output signal *Sisc readdata* contains the data that is sent to the SISC, while the signals *hresponse* and *hready* are the corresponding signals of the AHB-Lite protocol. Finally, the output signal *select* indicates that the AHB-Lite Bus should be stalled and the readdata, hresponse and hready signals of the SISC should be connected to the corresponding signals of the authentication manager instead of AHB-Lite Bus.

signal	input/output	bits	description
Sisc writedata	input	32	data sent by Sisc
address[31:28]	input	4	address bits used to initiate the protocol
Sisc readdata	output	32	data sent to Sisc
new table	output	32	new table of the mpu
valid	output	1	indicates whether the mpu should be updated
select	output	1	indicates the AHB-lite Bus should be stalled and the hresponse and hready signals of the authentication manager should be connected to the Sisc
hresponse	output	1	hresponse signal of the AHB-Lite protocol during authentication
hready	output	1	hready signal of the AHB-Lite protocol during authentication

Table 4.6: Input and output signals of the authentication manager

The structural architecture of the authentication manager is shown in figure 4.14. The internal signals are shown in table 4.7. The static nonce and value of the counter are used by the encryption module to generate the random fresh nonce as described in section 4.4.3. The signals *i_data* and *o_data* are the input and output data of the encryption module as described in section 4.4.2. Furthermore, the command signal determines the function the encryption module will execute as described in section 4.4.2, while the ready signal indicates the encryption or decryption has finished. The signal

nonce contains the random fresh nonce, used during authentication. The two select signals determine the input data of the encryption module and data sent to the SISC. The enable signals enable the counter and the register files containing the fresh nonce and the encrypted command. Finally, the start and match signals indicate the initiation of the protocol and the equality of the MAC, respectively.

signal	bits	description
static nonce	32	static nonce used during nonce generation
value	32	counter value used during nonce generation
i_data	32	input data of the encryption module
o_data	64	output data of the encryption module
nonce	32	fresh nonce used during authentication
command	3	command the encryption module will execute
select mux to simon	2	determines input data of the encryption module
select mux to output	1	determines data sent to the Sisc
enable counter	1	enables the counter
enable fresh nonce	1	enables the register file storing the fresh nonce
enable command encrypted	1	enables the register file storing the encrypted command
start	1	indicates initiation of the protocol
match	1	indicates equality of the MAC
ready	1	indicates the encryption or decryption procedure has finished

Table 4.7: Internal signals of the authentication manager

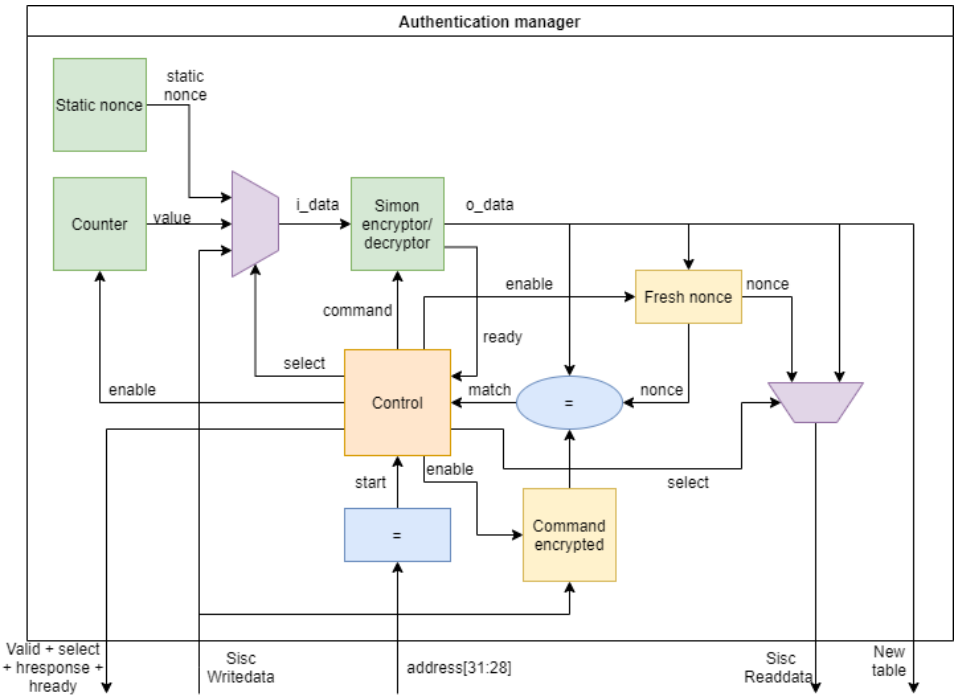


Figure 4.14: Structural architecture of the authentication manager

4.5. Bluetooth communication

As described in section 3.2, the prototype should provide wireless communication with a smartphone. Given the constraints on power consumption of the IMD, as described in section 3.4, we decided to implement Bluetooth Low Energy. To satisfy the security goals described in section 3.3, all data will be encrypted. Section 4.5.1 describes the Bluetooth module. Section 4.5.2 explains the services provided. Finally, section 4.5.3 explains describes the encryption of the data.

4.5.1. Bluetooth module

The Digilent Pmod BLE is a Bluetooth Low Energy module with a UART interface and is compatible with the FPGA. The module is shown in figure 4.15 and is used to communicate wirelessly with the phone. The simplicity of this module comes from its simple UART interface, used to send commands to the Pmod or receive updates from the Pmod. Furthermore, the Pmod BLE allows the use of Bluetooth Low-Energy without having to worry about low-level abstraction layers. This allows setting-up the demonstration without a steep learning curve.

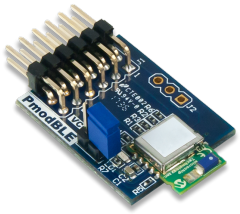


Figure 4.15: Digilent Pmod BLE [27]

The Pmod BLE is programmed to provide the Bluetooth services and characteristics shown in figure 4.16. There are two services, one for transferring heart sensor values to the phone app and one to perform the MPU firmware update. Within each service, characteristics can be programmed. Three types of characteristics are available:

- Read
- Write
- Notify

These types of characteristics are named from the phone perspective, as the phone is the central device during the demonstration, while the Pmod BLE is the peripheral device. Therefore, a characteristic of type read indicates a characteristic in which the phone app reads, while the FPGA writes. The reverse holds for a write type characteristic. A notify characteristic is similar to a read characteristic, with the exception that the phone receives a notification when it gets updated. The phone app development team requested the use of notify characteristics instead of read characteristics to facilitate programming the app for the demonstration.

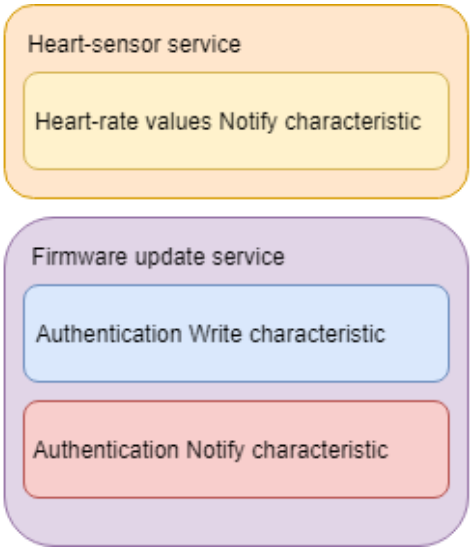


Figure 4.16: Bluetooth services and characteristics programmed in the Pmod

4.5.2. Bluetooth services

The firmware update service has two characteristics, as shown in figure 4.16. This allows performing the authentication protocol shown in figure 4.17.

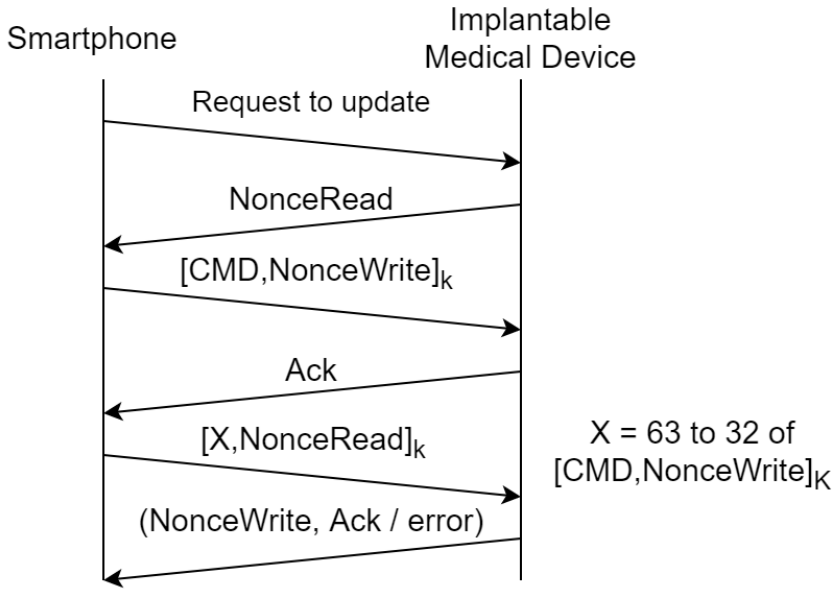


Figure 4.17: Firmware update authentication protocol between the FPGA and the phone

The authentication protocol for the firmware update, shown in figure 4.17, is derived from the authentication protocol between the SISC and the interconnect, discussed in section 4.4, and performs the following steps.

1. The smartphone sends a request for a firmware update to the FPGA.
2. The FPGA requests the implant nonce from the interconnect and relays the nonce to the smartphone.
3. The smartphone sends the encrypted command and reader nonce to the FPGA.
4. The FPGA sends an acknowledge to the smartphone.
5. The smartphone sends the MAC to the FPGA
6. The FPGA finishes the protocol described in section 4.4.1 and sends an acknowledge to the smartphone.

The heart-rate service simply allows the FPGA to send heart-rate values to the phone.

4.5.3. Encryption

All characteristics used by the BLE communication are encrypted using a Simon cipher with a 128 bits key size and a 64 bits block size. It is assumed that the key is shared via NFC with the phone app, as implemented by subgroup 1.

The heart-sensor service consists of a stream of heart rate values. A naive approach would be to encrypt each value independently, also known as electronic codebook (ECB). Usage of ECB is not recommended because it does not hide data patterns well[28]. This is particularly relevant, because sensitive medical information such as abnormalities can often be derived from patterns or irregularities. In addition, ECB is more susceptible to replay attacks, because each block is encrypted in the same way. This is relevant if a man in the middle obtains the cipher-text of the request for a firmware update. This allows them to perform a replay attack by repeatedly sending this request, keeping the SISC busy with the authentication, thus preventing heart-rate data transfer.

To avoid such problems, several more sophisticated modes of operations exist. For the wireless communication between the smartphone, we chose to implement cipher block chaining (CBC). The advantage of CBC lies in its simplicity. In general, CBC has two disadvantages. First, encryption cannot be parallelized. Second, messages must be padded to a multiple of the block size. The first feature is not necessary as words are sent in a sequential manner and will therefore be encrypted and decrypted in a sequential manner. The second disadvantage is not relevant as the encrypted values have the same size as the block cipher. The theoretical disadvantages of CBC are therefore not relevant in our implementation.

Figure 4.18 shows how successive words written into a characteristic are encrypted and decrypted using CBC. During encryption, the plaintext is xor-ed with the ciphertext of the previous block. During decryption, the output of the decryption module is xor-ed with the ciphertext of the previous block. The first block uses an initialization vector instead. It is assumed that the initialization vector is agreed on with the phone development team for each characteristic.

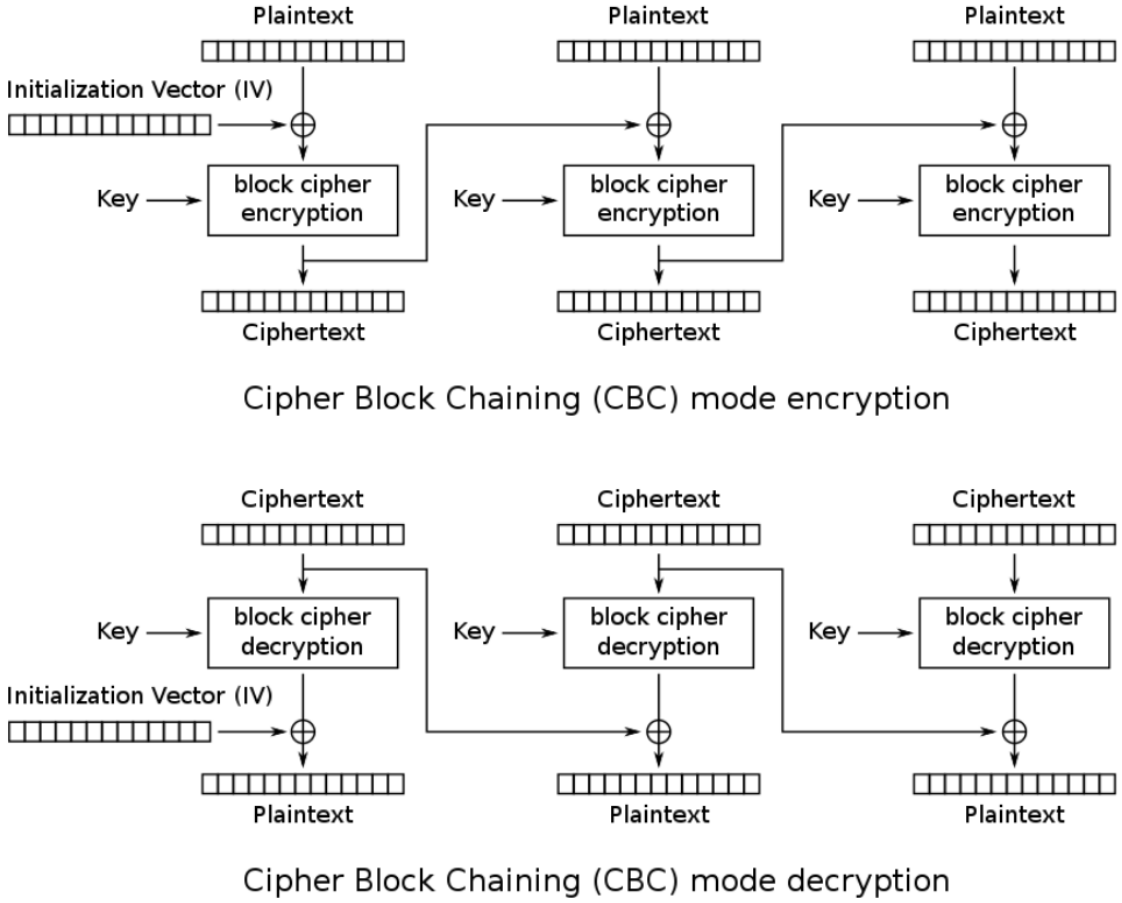


Figure 4.18: Overview of encryption and decryption using CBC [29]

5

Verification and implementation

To verify the behaviour of the prototype, the different abstraction layers have been simulated. Furthermore, the complete system has been demonstrated by an implementation on an FPGA. Section 5.1 through 5.3 describe the simulations of the AHB-Lite Bus, the memory protection unit and the authentication manager. Finally, section 5.4 describes the demonstration of the complete system and gives an overview of the hardware cost of the modules.

5.1. AHB-Lite Bus

5.1.1. Decoder

The behaviour of the decoder has been verified by simulating it with a test-bench. First, the simulation runs through every address with the valid low. Next, it runs through every address with the valid high. The resulting waveform is shown in figure 5.1. From the figure it is clear that the select signals remain zero during the first run and select the correct slave during the second run. Therefore, we conclude that the decoder exhibits the correct behaviour.

5.1.2. Arbiter

The behaviour of the arbiter has been verified by simulating it with a test-bench that runs through the state diagram described in section 4.2.4. Table 5.1 shows the states during simulation, including the corresponding transition times. In addition, the table indicates the required output signal transitions, corresponding to these state transitions, allowing us to verify the behaviour of the arbiter.

Figure 5.2 shows the waveform, resulting from the simulation. The waveform shows the expected signal transitions, as described in table 5.1, indicating that the arbiter behaves correctly.

5.1.3. Top level of the AHB-Bus

The functionality of the AHB-Bus has been tested by simulating many scenarios. As an illustration, figure 5.3 shows one such scenario.

At the beginning of the scenario, address 0XF0000000 holds the value 0x00000050.

Both masters then issue a read request to address 0XF0000000. The interconnect prioritizes master 0 and sends 0x00000050 as read data back. After that, master 0 immediately sends a write request to address 0xF0000000 with a value of 0x00000100. The interconnect still prioritizes this write request over the read request of master 1, received at the beginning. Finally, the read request of master 1 is processed, and a value of 0x00000100 is sent to master 1.

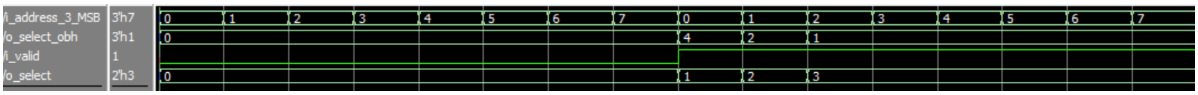


Figure 5.1: Resulting waveform of the simulation of the decoder with a test-bench

State	Transition time	Output signal transitions
Idle	0 ns	
Address_cycle_congestion	30 ns	select_mux -> 01, pending1 -> 1, pending2 -> 1, hold2 -> 1, hsel ->1
Data_cycle_congestion	50 ns	
Address_cycle1	70 ns	select_mux -> 10, hold2 -> 0
Data_cycle1	90 ns	pending2 -> 0
Address_cycle1	110 ns	pending2 -> 1
Data_cycle1_congestion	130 ns	pending1 -> 1, pending2 -> 0, hold1 -> 1
Address_cycle0	150 ns	select_mux -> 01, hold1 -> 0
Data_cycle0	170 ns	pending1 -> 0
Idle	190 ns	select_mux -> 00, hsel -> 0

Table 5.1: Subsequent states of the arbiter during simulation, including the transition times and the output signal transitions associated with the corresponding state transitions

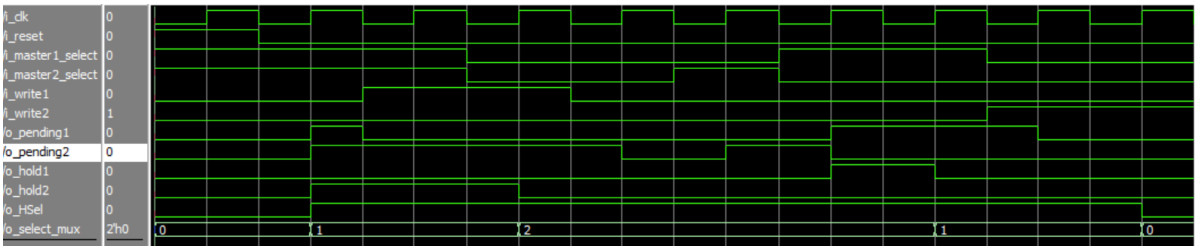


Figure 5.2: Resulting waveform of the simulation of the arbiter with a test-bench

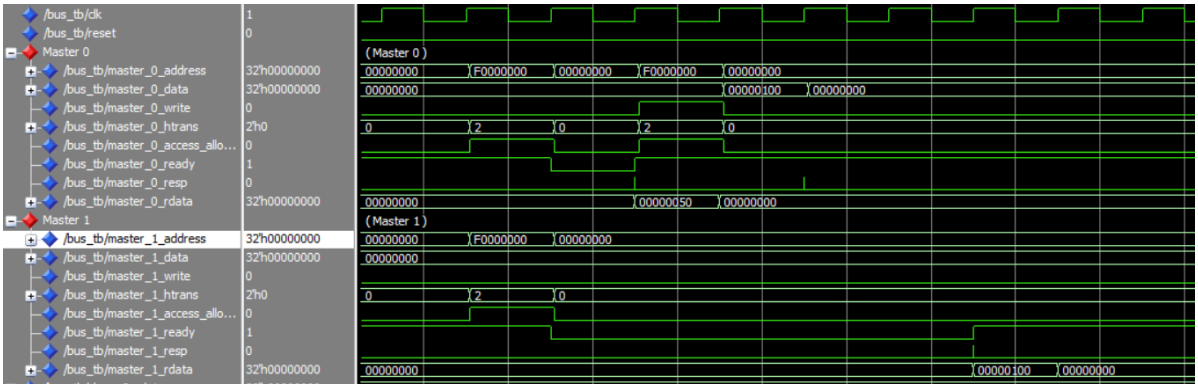


Figure 5.3: Resulting waveform of one of the scenario's, simulated to test the AHB bus

5.2. Memory protection unit

The behaviour of the memory protection unit has been verified by simulating it with a test-bench. The simulation starts by updating the memory protection unit with the protection table shown in table 5.2. Next, the read and write accesses of all addresses are checked for both masters. The resulting waveform is shown in figure 5.4. From this waveform, the access rights of both masters can be deduced. The access rights of master 1 and 2 are shown in table 5.3 and table 5.4, respectively. As explained in section 4.3 the first two bits of each entry contain the write and read access rights of the first master, respectively, while the last two bits contain the access rights of the second master. From the tables it is clear that the access rights at the output correspond with those of the protection table, indicating that the MPU shows the correct input-output behaviour.

entry	value
1	0011
2	1100
3	0010
4	1101
5	0001
6	1110
7	0000
8	1111

Table 5.2: Protection table loaded into the memory protection unit during simulation

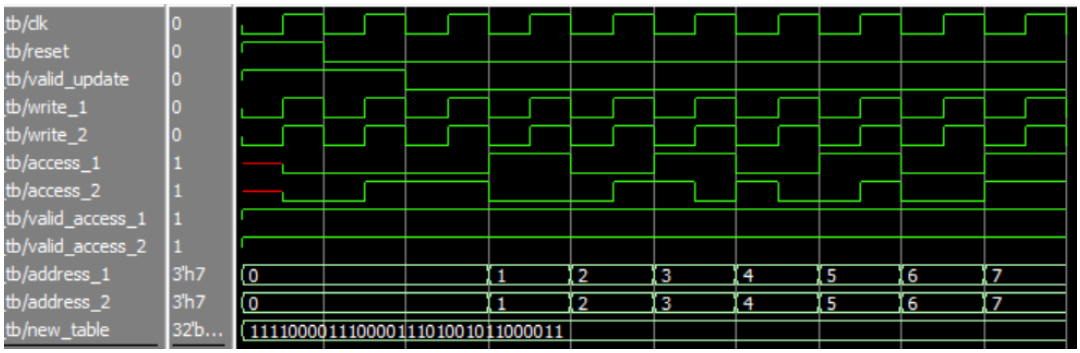


Figure 5.4: Resulting waveform of the simulation of the memory protection unit with a testbench

Address	Read	Write
000	0	0
001	1	1
010	0	0
011	1	1
100	0	0
101	1	1
110	0	0
111	1	1

Table 5.3: Access rights of master 1 deduced from the waveform during simulation

Address	Read	Write
000	1	1
001	0	1
010	0	1
011	1	0
100	1	0
101	0	1
110	0	0
111	1	1

Table 5.4: Access rights of master 2 deduced from the waveform during simulation

5.3. Authentication manager

5.3.1. Encryption module

The behaviour of the encryption module has been verified by simulating it with two test-benches: one for encryption and one for decryption. Both test-benches start by loading the data in the lower and upper register, respectively, followed by initiating either encryption or decryption. The resulting waveforms are shown in figure 5.5 and 5.6.

To validate the encrypted or decrypted data, a matlab script has been written, which is able to perform both encryption and decryption. The results are shown in table 5.5 and coincide with the results of the test-benches, indicating that the encryption module behaves correctly.

Operation	Key	Input	Output
Encryption	0xFC6CC71FFC6CC71FFC6CC71F	0xFC6CC71FFECED2FF	0x66AE922B064DA7BF
Decryption	0xFC6CC71FFC6CC71FFC6CC71F	0x66AE922B064DA7BF	0xFC6CC71FFECED2FF

Table 5.5: Results of the matlab script of the simon block cipher including the key as well as the input and output data

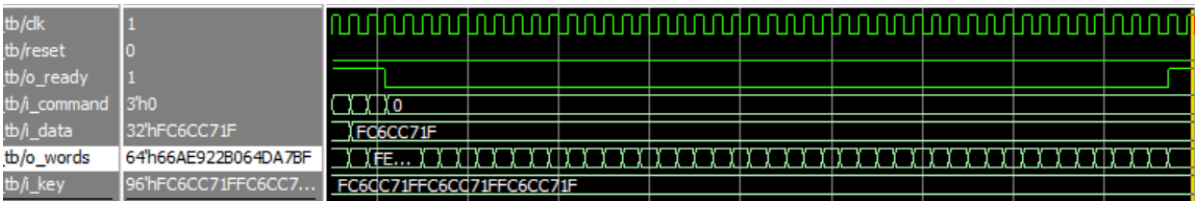


Figure 5.5: Resulting waveform of the simulation of the encryption of data by the encryption module using a test-bench

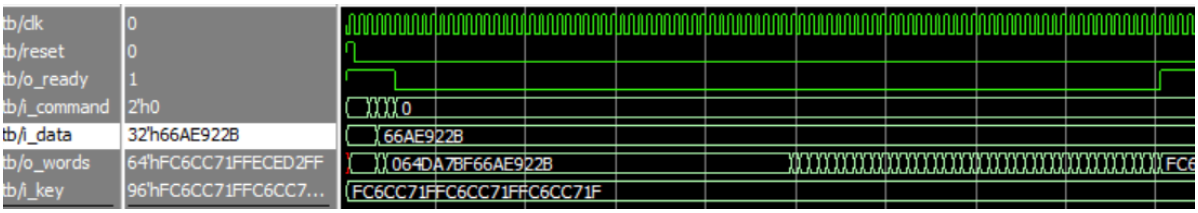


Figure 5.6: Resulting waveform of the simulation of the decryption of data by the encryption module using a test-bench

5.3.2. Control FSM

The behaviour of the control FSM of the authentication manager has been verified by simulating it with a test-bench that runs through the state diagram described in section 4.4.4. Table 5.6 shows the states during simulation, including the corresponding transition times. In addition, the table indicates the required output signal transitions, corresponding to these state transitions, allowing us to verify the behaviour of the control FSM.

Figure 5.7 shows the waveform, resulting from the simulation. The waveform shows the expected signal transitions, as described in table 5.6, indicating that the control FSM behaves correctly.

State	Transition time	Output signal transitions
Idle	0 ns	
Load static nonce, count + 1	50 ns	Count_enable -> 1, Mux_select -> 1, Command -> 1, auth_select
Load counter value	70 ns	Count_enable -> 0, Mux_select -> 2, Command -> 2
Start nonce generation	90 ns	Command -> 4
Wait nonce generation	110 ns	Command -> 0
Store nonce	150 ns	Nonce_enable -> 1
Send nonce	170 ns	Nonce_enable -> 1, hready -> 1
Write command	190 ns	Command_enable -> 1, Nonce_enable -> 0 hready -> 0
Write MAC lower register	230 ns	Command_enable -> 0, Command -> 1
Write MAC upper register	270 ns	Command -> 2
Start decryption MAC	310 ns	Command -> 3
Wait decryption MAC	330 ns	Command -> 0
Compare MAC	370 ns	No transition
Send authentication message	390 ns	Hready -> 1
Write reader nonce lower register	410 ns	Hready -> 0, Command -> 1
Start decryption command + reader nonce	450 ns	Command -> 3
Wait decryption command + reader nonce	470 ns	Command -> 0
Update mpu + send reader nonce	510 ns	Nonce_select -> 1, mpu_valid -> 1, hready -> 1
Idle	530 ns	Nonce_select -> 1, mpu_valid -> 1, hready -> 1 auth_select -> 1

Table 5.6: Subsequent states of the control FSM of the authentication manager during simulation, including the transition times and the output signal transitions associated with the corresponding state transitions

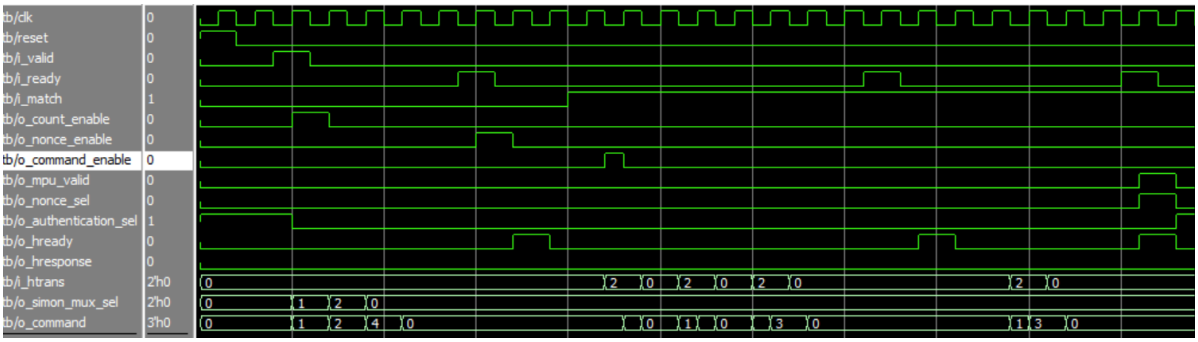


Figure 5.7: Resulting waveform of the simulation of the control FSM of the authentication manager with a test-bench

5.3.3. Top level

In order to verify the behaviour of the authentication manager, a test FSM has been written which simulates the behaviour of the SISC during authentication. The state diagram of this FSM is shown in figure 5.8. When the authentication protocol starts, the FSM first requests the nonce by writing in a specific address. After the nonce has been received, the FSM subsequently sends the encrypted command and MAC, after which it waits for the authentication message. If an error message is received the FSM returns to the idle state. If the authentication is successful, the SISC sends the encrypted reader nonce. Finally, once the reader nonce is sent back, the FSM returns to the idle state. The data used during simulation is shown in table 5.7.

Cmd	0xAAAAAAAA
N_I	0x7E66507D
N_R	0xFECED2FF
$[Cmd, N_R]_K$	0xE8BE0B0B22A39DE3
$[[Cmd]_K, N_I]_K$	0x27F51ED09ECE42C6

Table 5.7: Data used during the simulation of the authentication manager

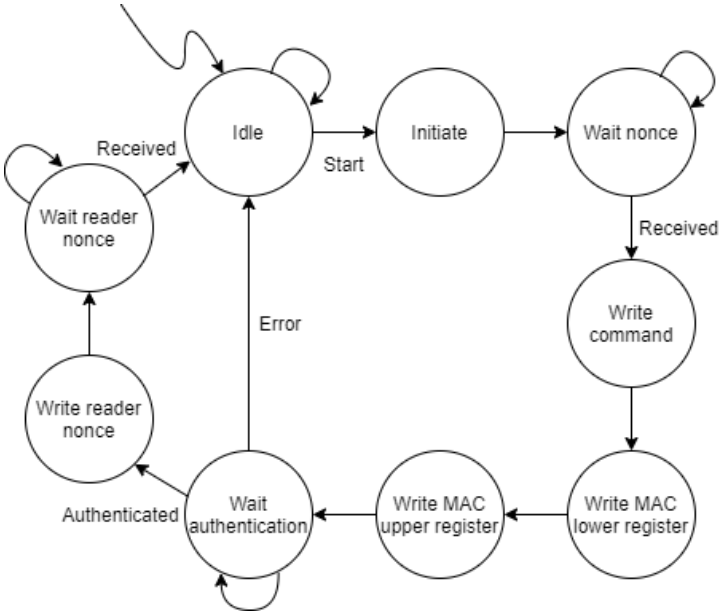


Figure 5.8: State diagram of the test fsm used to verify the behaviour of the authentication manager

The resulting waveform is shown in figure 5.9. As can be seen from the figure, the signal *nonce_out* and *new_table* correspond to the respective data from table 5.7. Additionally, the signals *hready*, which

indicates that data is sent to the SISC, behaves correctly as well as the signal *valid*, which indicates that the MPU should be updated. From these observations we conclude that the authentication manager behaves properly.

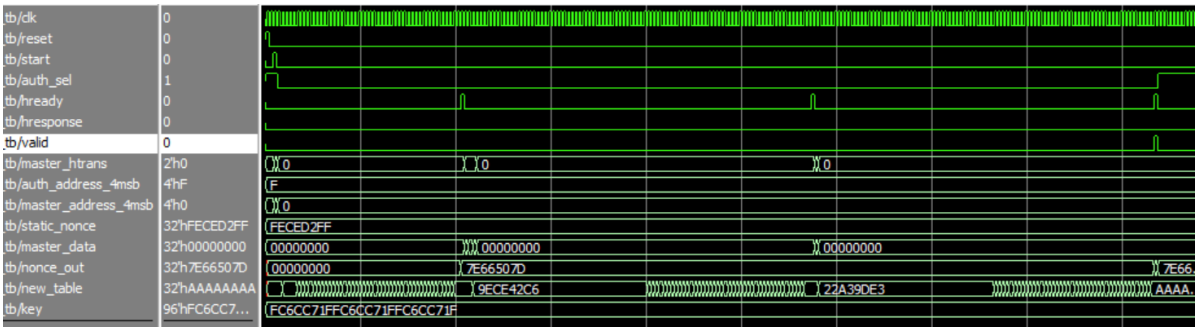


Figure 5.9: Resulting waveform of the simulation of the authentication with the aid of the test FSM shown in figure 5.8

5.4. Implementation

For the demonstration, a Cyclone II DE1 board is used. Section 5.4.1 describes the demonstration on the FPGA. Next, section 5.4.2 gives an overview of the hardware cost of the modules as well as the complete interconnect.

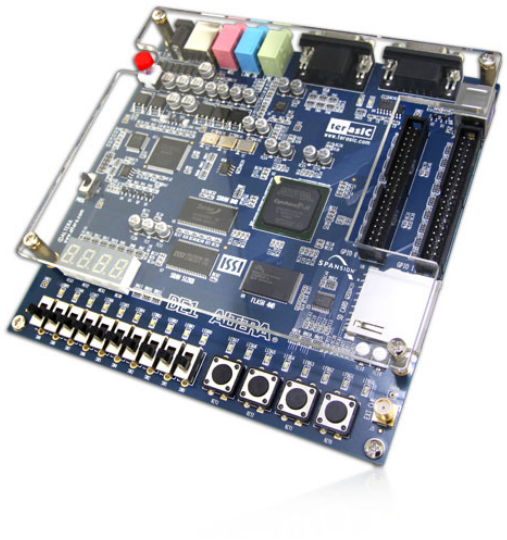


Figure 5.10: Cycle II DE1 FPGA board used for the demonstration [30]

5.4.1. Demonstration

Figure 5.11 shows the diagram of the implant SoC as well as the diagram of the system used during the demonstration.

The memory is emulated by an array of registers, the actuator by a single register, and the sensor by a register of which the input value comes from a random number added to an average value (80 in this demonstration, as this is the average heart-rate).

The main implant core and the SISC have been simulated by an FSM. Figure 5.12 shows the corresponding state diagrams as well as the physical address space.

The main implant core reads the sensor values every 2 seconds, and subsequently writes them into memory range 1, setting the MSB of the written value to 1 to indicate that it is a valid value.

The SISC reads to sensor values written in address range 1 and sends them to the Pmod BLE, which in turn sends them to the smartphone via Bluetooth.

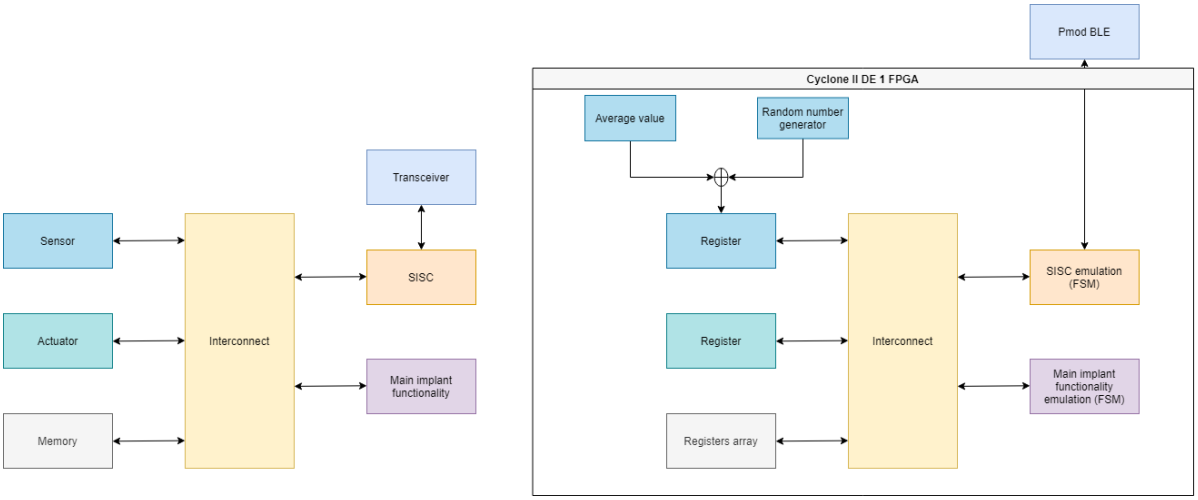


Figure 5.11: Diagram of the ”real” implant SoC (left) and diagram of the system used for the demo (right)

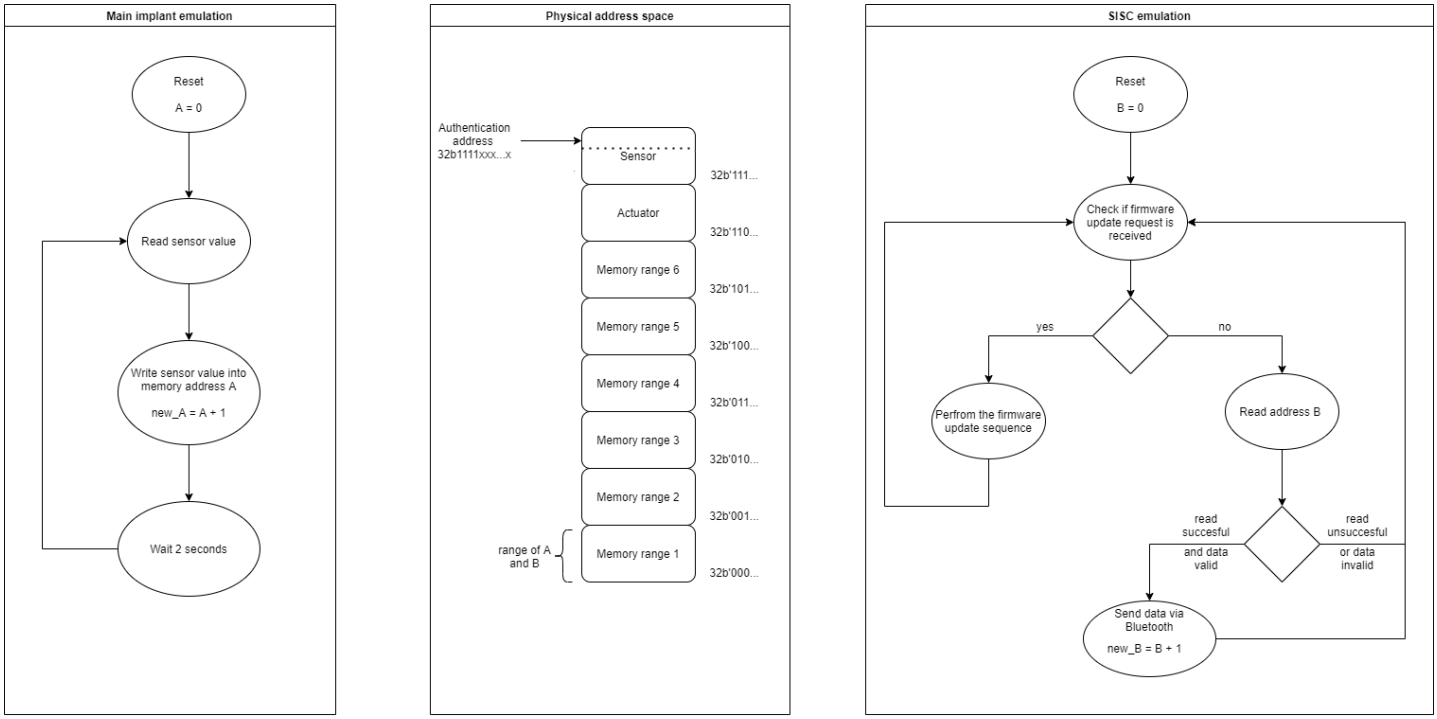


Figure 5.12: Diagram of the emulated main implant core and SISC, and the physical address space

Initially, the SISC does not have access to memory range 1. The demonstration starts with performing a firmware update to allow the SISC access to this memory range. After that, the SISC starts sending data to the phone app.

The demonstration has shown that the complete system is operating correctly.

5.4.2. Hardware cost

The designed interconnect was synthesized for the DE1 FPGA. Table 5.8 shows the resulting hardware cost. The table shows that the authentication manager occupies the largest area.

Module	Number of combinatorial functions	Number of logic registers
AHB bus	539	169
Authentication manager	1420	1595
MPU	59	32
Interconnect (total)	2018	1796

Table 5.8: Hardware cost of the different modules and the complete interconnect, implemented on the FPGA

6

Conclusion and discussion

6.1. Conclusion

The goal of this thesis was the design and implementation of a secure bus-architecture that connects multiple processing cores and other entities within an MBAN node, providing isolation between the modules as well as memory protection. These requirements have been achieved by dividing the design into three modules: an AHB-Lite Bus, a memory protection unit and an authentication manager. The purpose of the AHB-Lite Bus is to regulate the data flow between the masters and slaves according to the AMBA AHB-Lite protocol. The purpose of the memory protection unit is to deny accesses from unauthorized masters. Finally, the authentication manager provides a secure protocol for updating the firmware of the mpu. Additionally, the prototype provides secure wireless communication with a smartphone.

The AHB-Lite Bus regulates the data flow using 2 decoders and 3 arbiters. Each of the decoders decode the accesses of one of the masters, while each arbiter regulates the accesses to a specific slave. To account for different scenario's, the arbiter has been implemented as an FSM. The arbiters are designed to prioritize master 0 to ensure the availability of the main core.

The memory protection unit was implemented with a protection table with fixed address ranges. This limits the flexibility, but decreases the hardware cost, an important performance indicator, given the limited resources of the IMD.

The goal of the authentication manager was to provide confidentiality, authentication and message integrity. This has been achieved through the use of encryption as well as a message authentication code, including a randomly generated nonce. The limited power and area constraints of the IMD provided the main argument for choosing the symmetric lightweight block cipher SIMON64/96 with a serial implementation. Generation of the pseudo-random nonce has been done using a block cipher running in counter mode, which allowed for reuse of the encryption module.

The wireless communication was provided by a Pmod BLE module. A Simon block cipher was used for encryption of the data. Additionally, CBC has been used to hide patterns in the stream of heart-rate values.

All functional requirements have been verified through simulation as well as a demonstration of the proof-of-concept implementation on the FPGA.

6.2. Future work

The proof-of-concept implementation of the bus-architecture meets all functional requirements described in section 3.2. Nevertheless, future work may explore modification or improvements in the design. Such modifications have been eased by the modularity of our design as explained in section 4.1. Some recommendation on future work are listed below.

- **AHB-Lite Bus:** The current design of the AHB-Lite Bus does not support all transfer types provided by the AHB-Lite protocol. The AHB-Lite Bus may therefore be extended to support types of transfers that are currently not included, such as burst transfers.

- **Authentication protocol:** Future work may explore ways to provide authentication other than message authentication codes, such as cryptographic hash functions or digital signatures.
- **Encryption module:** The encryption module has been chosen for optimization of power consumption and hardware cost. Nonetheless, cryptography is a constantly evolving field. As a consequence, more efficient block ciphers may be designed in the future. Additionally, new security threats or attacks may be discovered, rendering current cryptographic techniques insufficiently secure. For these reasons the encryption module may need to be replaced in the future.

Furthermore, as stated in section 1.2, the deliverable of this thesis is a proof-of-concept implementation on an FPGA. Future work may move towards a more customer-oriented product.

Bibliography

[1] Mayo Clinic, "Epilepsy, symptoms causes, mayo clinic." https://www.mayoclinic.org/diseases-conditions/epilepsy/symptoms-causes/syc-20350093. Published on 31/03/2021, Accessed on 01/06/2021.

[2] World Health Organization, "Epilepsy." https://www.who.int/news-room/fact-sheets/detail/epilepsy. Published on 20/06/2019, Accessed on 01/06/2021.

[3] National Health Service, "Epilepsy, nhs." https://www.nhs.uk/conditions/epilepsy/. Published on 18/09/2020, Accessed on 01/06/2021.

[4] M. Nielen, M. Poos, A. Gommer, and M. Rodriguez, "Epilepsie, cijfers context, huidige situatie." https://www.volksgezondheidenzorg.info/onderwerp/epilepsie/cijfers-context/huidige-situatie#node-nieuwe-gevallen-epilepsie-huisartsenpraktijk.

[5] L. Xia, S. Ou, and S. Pan, "Initial response to antiepileptic drugs in patients with newly diagnosed epilepsy as a predictor of long-term outcome," *Frontiers in Neurology*, vol. 8, p. 658, 2017.

[6] K. Jeffrey and V. Parsonnet, "Cardiac pacing, 1960–1985," *Circulation*, vol. 97, no. 19, pp. 1978–1991, 1998.

[7] C. Boyd, A. Mathuria, and D. Stebila, *Protocols for authentication and key establishment*, vol. 1. Springer, 2003.

[8] J. Teng, W. Gu, and D. Xuan, "Chapter 10 - defending against physical attacks in wireless sensor networks," in *Handbook on Securing Cyber-Physical Critical Infrastructure* (S. K. Das, K. Kant, and N. Zhang, eds.), pp. 251–279, Boston: Morgan Kaufmann, 2012.

[9] Q. Do, B. Martini, and K.-K. R. Choo, "The role of the adversary model in applied security research," *Computers Security*, vol. 81, pp. 156–181, 2019.

[10] "Final implementation of the sharcs hardware techniques," *Secure Hardware-Software Architectures for Robust Computing Systems*, 2017.

[11] European Union, "Regulation 2016/676 of the european parliament and of the council." https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504. Published on 27/04/2016, Accessed on 02/06/2021,.

[12] Food and Drug Administration, "Postmarket management of cybersecurity in medical devices." https://www.fda.gov/media/95862/download. Published on 28/12/2016, Accessed on 02/06/2021.

[13] J. Charthad, M. J. Weber, T. C. Chang, and A. Arbabian, "A mm-sized implantable medical device (imd) with ultrasonic power transfer and a hybrid bi-directional data link," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 8, pp. 1741–1753, 2015.

[14] P. Gerrish, E. Herrmann, L. Tyler, and K. Walsh, "Challenges and constraints in designing implantable medical ics," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 435–444, 2005.

[15] Arm, "Multi-layer ahb technical overview v." https://developer.arm.com/documentation/dvi0045/b, 2004.

[16] L. Batina, A. Das, B. Ege, E. B. Kavun, N. Mentens, C. Paar, I. Verbauwhede, and T. Yalçın, "Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures," in *Radio Frequency Identification* (M. Hutter and J.-M. Schmidt, eds.), (Berlin, Heidelberg), pp. 103–112, Springer Berlin Heidelberg, 2013.

[17] M. Ågren, *On Some Symmetric Lightweight Cryptographic Designs*. PhD thesis, Department of Electrical and Information Technology, 2012. Defence details Date: 2012-11-28 Time: 13:15 Place: Lecture hall E:1406, E-building, Ole Römers väg 3, Lund University Faculty of Engineering External reviewer(s) Name: Rijmen, Vincent Title: Prof. Affiliation: KU Leuven, ESAT/SCD (COSIC), Heverlee, Belgium. —.

[18] S. Banik, A. Bogdanov, and F. Regazzoni, "Exploring the energy consumption of lightweight block-ciphers in fpga," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–6, 2015.

[19] A. Bossert, S. Cooper, and A. Wiesmaier, "A comparison of block ciphers simon, speck, and katan," September 2016.

[20] J. Hosseinzadeh and A. G. Bafghi, "Evaluation of lightweight block ciphers in hardware implementation: A comprehensive survey," 2017.

[21] W. Diehl, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, "Comparison of hardware and software implementations of selected lightweight block ciphers," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, 2017.

[22] Nayancy, S. Dutta, and S. Chakraborty, "A survey on implementation of lightweight block ciphers for resource constraints devices," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 0, no. 0, pp. 1–22, 2020.

[23] J. Wetzels and W. Bokslag, "Simple simon: Fpga implementations of the simon 64/128 block cipher," 2015.

[24] L. Qin, H. Chen, and X. Wang, "Linear hull attack on round-reduced simeck with dynamic key-guessing techniques," in *Information Security and Privacy* (J. K. Liu and R. Steinfeld, eds.), (Cham), pp. 409–424, Springer International Publishing, 2016.

[25] N. Ferguson, B. Schneier, and T. Kohno, *Generating Randomness*, ch. 9, pp. 135–161. John Wiley Sons, Ltd, 2015.

[26] H. Lipmaa, P. Rogaway, and D. Wagner, "Comments to nist concerning aes modes of operations: Ctr-mode encryption," in *National Institute of Standards and Technologies*, 2000.

[27] Digilent, "Pmod ble: Bluetooth low energy interface." `https://store.digilentinc.com/pmod-ble-bluetooth-low-energy-interface/`.

[28] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications, CRC Press, 2018.

[29] Wikipedia, "Block cipher mode of operation." `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`.

[30] Terasic, "Altera de1 board." `https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=83`.