Test and Repair of Reconfigurable On-chip Instrument Access Networks

Zehang Xiang
ze3514xi-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Erik Larsson

Examiner: Pietro Andreani

October 14, 2019

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

As transistors in integrated circuits (ICs) are becoming smaller, faster and more, it has become harder to avoid malfunctioning. Embedded instruments are increasingly used to test, tune, and configure the transistors in ICs. IEEE Std.1149.1-2013 and IEEE Std.1687 standardize the access to these embedded instruments. IEEE Std.1687 enables reconfigurable scan networks which allow only desirable instruments to be included in the active scan-path. Reconfigurable scan networks can become faulty, which may lead to the situation there will be no possibility left to test, trim and configure the IC. In this thesis, we focused on the test and repair of the faulty scan registers. We have introduced two solutions, based on the hardware and software, to test the reconfigurable scan network, to identify faulty scan registers, and repair the network by excluding the faulty scan register from the network. We did two experiments to measure the overhead in terms of area and data. Here, the data overhead is the data needed to be sent to IC for testing and repairing the reconfigurable network. From these experiments, we found both the solutions have unique advantages. The software solution does not use hardware area, while the hardware solution results in low data overhead.

Popular Science Summary

The integrated circuits (ICs) consist of transistors, which are the basic components of an electronic circuit. Nowadays, transistors in integrated circuits become smaller and smaller, which increases the impact of errors in the design and manufacturing process.

Facing this problem, it becomes an important task for designers to monitor the errors of various instruments of electronic devices. In order to achieve this goal, a solution has been developed that integrate IC monitoring, testing and debugging components in the manufacturing process. These basic components are called on-chip instruments.

Since the instrument is built into the chip, additional infrastructure is needed to make the environment accessible. This is called the reconfigurable scan network. This kind of network connects with instruments through registers, which are called scan registers. Scan registers can become faulty, which may lead to the situation there will be no possibility left to test, trim and configure the IC.

In this thesis, we have introduced two solutions, based on the hardware and software, to test the reconfigurable scan network, to identify faulty scan registers, and repair the network by excluding the faulty scan register from the network. Both solutions have unique advantages. The software solution does not use hardware area, while the hardware solution results in terms of low data exchanged between the chip and environment(known as data overhead).

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Motivation

As transistors in integrated circuits (ICs) are becoming smaller, faster and more, it has become harder to avoid malfunctioning. Embedded instruments are increasingly used to test, tune, and configure the transistors in ICs. IEEE Std.1149.1-2013 and IEEE Std.1687 standardize the access to these embedded instruments. IEEE Std.1687 enables reconfigurable scan networks which allow only desirable instruments to be included in the active scan-path. Reconfigurable scan networks can become faulty, which may lead to the situation there will be no possibility left to test, trim and configure the IC. The goal of this thesis is to study test methods of the reconfigurable scan networks and repair these faults. Furthermore, we will address the communication with the scan network. Instead of using a dedicated test port, for example IEEE Std. 11491.1, we will make use of a functional I/O-port, namely Universal Asynchronous Transmitter (UART).

1.2 Thesis organization

In Chapter 2, we will introduce the background of the system, UART and IEEE Std.1687. In addition, parameters that affect the performance are discussed. In Chapter 3, the details of both the solutions will be introduced. In Chapter 4, experiments will be conducted to test the performance and the results and related analyses will be stated in the following parts. In Chapter 5, we will summarize the whole work we have done in this thesis and suggestions will be given for future work.

Chapter 2
Background

In this chapter, we discuss the system, UART, IEEE Std.1687, design from [3].

2.1 System

Figure 2.1 shows an overview of the system. The embedded instruments are coupled with the scan registers and connected as a reconfigurable scan network based on IEEE Std.1687. The Test Access Port (TAP) of IEEE Std.1149.1 is used to connect the reconfigurable scan network and the outside world. Instead of using dedicated test port, IEEE Std.1687.1 introduces a way to access the scan network with the functional ports. In this thesis, the UART will be used as the functional port. Figure 2.2 is an overview of the system using UART.

2.2 UART

The UART is a computer hardware device for asynchronous serial contact that includes a transmitter and a receiver. The transmitter is essentially a special shift register that loads data in parallel and then shifts the data out at a specific rate. On the other hand, the receiver shifts the data and then reassembles the data. When the serial line is idle, it is "1". The transfer starts with a start bit of '0', then 5-8 data bits an optional parity bit, and ends with a stop bit of "1". An optional parity bit is used for error detection. For odd parity, when the number of 1s of data bits is an odd number, it is set to "0". For even parity, when the number of 1s of data bits is an even number, it is set to "0". The number of stop bits can be 1 or 2. Figure 2.3 shows a transmission with 8 data bits, 1 parity bit and 1 stop bit. No clock information is conveyed through the serial line [3]. Commonly used baud rates are 2400, 4800, 9600, 19200 and 115200 BPS.

Since the transmitter does not transmit clock information, the receiver can only retrieve the data bits using predetermined parameters. Oversampling schemes are used to overcome this problem. According to this scheme, the middle point of the transmitted bits is estimated and the receiver polls the channel at these points. The oversampling rate depends on the baud rate and clock period of the receiver.

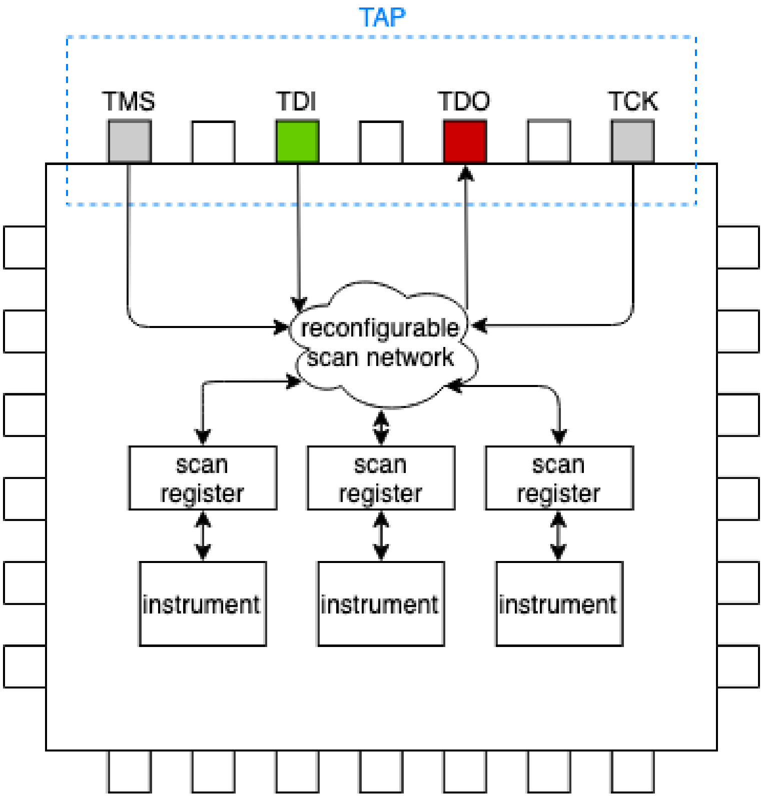


Figure 2.1: Overview of the system with TAP.

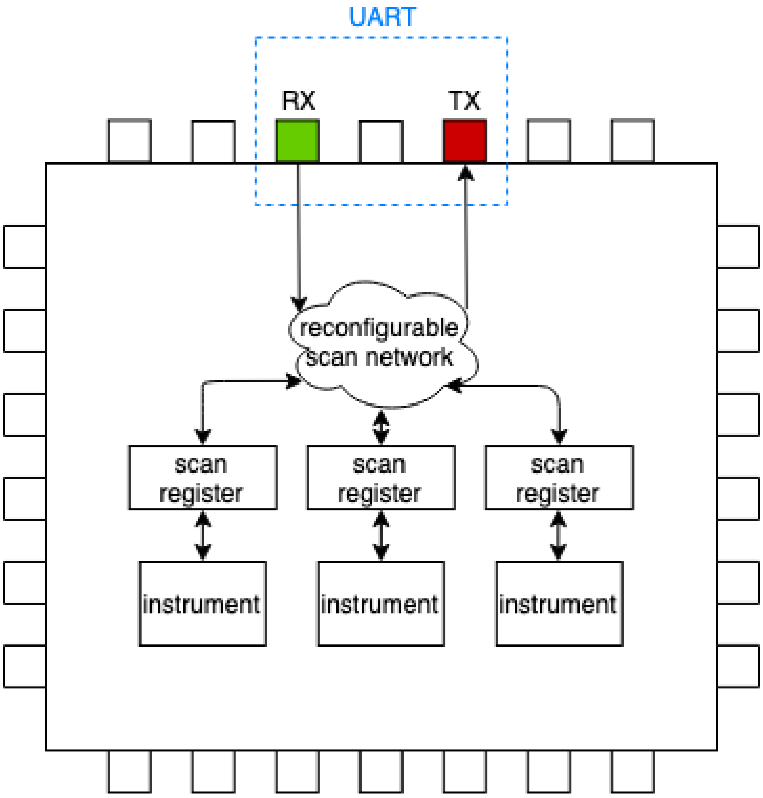


Figure 2.2: Overview of the system with UART.

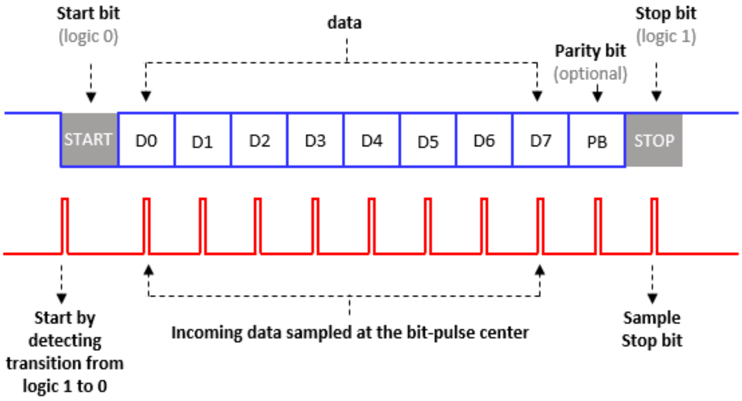


Figure 2.3: Break down of UART transmission [5].

2.3 IEEE Std.1687 (IJTAG)

In this section, the IEEE Std.1687, also known as internal JTAG, is introduced. The IJTAG standard [6] is a methodology for accessing on-chip instrumentations through the reconfigurable scan network. There are different network architectures such as flat networks, hierarchical networks, multiple networks, and daisy-chained networks. In this thesis, all the designs are based on flat architecture. Figure 2.4 shows an example of the flat reconfigurable network.

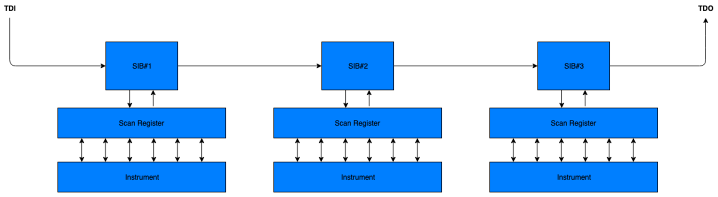


Figure 2.4: Flat IEEE Std.1687 network with three instruments.

2.3.1 Segment Insertion Bit (SIB)

The segment insertion bit (SIB) is used to include or exclude instruments in the reconfigurable network . Figure 2.5 shows the schematic of a SIB.

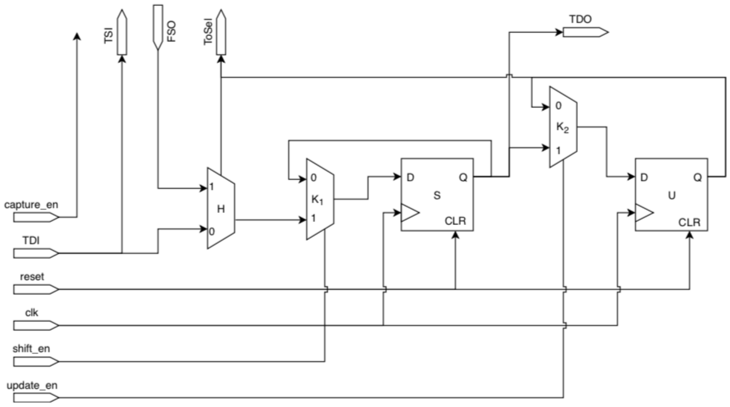


Figure 2.5: Detailed SIB schematic [3].

There are three control signals (shift_en, update_en and capture_en) to control a SIB. At the beginning, the control bit in the U flip-flop is ’0’. Therefore, the MUX H will be connected to the TDI port, and the instruments are excluded from the network. If the SIB unit is in shift mode, the shift_en becomes ’1’ and MUX K_1 will be connected to MUX H. Data will be shifted from the TDI into the S flip-flop at the rising edge of the CLK signal. When the SIB unit is not in shift mode, the shift_en becomes ‘0’ and the S flip-flop remains its value through the feedback from K_1.

If the SIB unit is in update mode, update_en becomes '1' and MUX K_2 is connected to the S flip-flop. In this way, the U flip-flop gets the values saved in the S flip-flop. When the SIB unit is not in update mode, update_en becomes '0' and the U flip-flop keeps its value through the feedback from K_2. If the value in the U flip-flop is '0', the instrument will be excluded and MUX H will receive the value directly from TDI. If the value stored in the U flip-flop is '1', the MUX H will be connected to the FSO port and the instrument select signal ToSel will enable the instrument network and the instrument will be included in the scan path.

To be noticed, only one of these control signals will be activated at one time. In addition, the function of the capture_en and the update_en is to enable parallel loading of data between instrument and shift register and shift register and instrument respectively.

2.3.2 Description languages

The IEEE Std.1687 standard introduces two description languages, Instrument Connectivity Language (ICL) and Procedural Description Language (PDL). The purpose of PDL is to describe the operations of the instruments. The ICL is used to describe the characteristics of the instruments such as data length and position within the network and the requirements for interfacing them [1].

2.4 Basics of design

Our design are based on the design from [3]. The UART are used as a interface to comunicate with the reconfigurable network in their thesis.

2.4.1 Hardware description

Figure 2.6 shows their full-featured case we used in our thesis, which consists of UART, network controller and reconfigurable scan network. When the test controller want to access the reconfigurable network, the test controller will first send the PDL commands to the UART which will transfer these commands to the network controller. The network controller will control the reconfigurable network according to these commands. The network controller consists of 4 parts: the IEEE 1687 FSM, the SIB control register (SCR), the instrument length memory and the discard unit. The IEEE 1687 FSM is for controlling the shift, update and capture signals. The SCR stores information about which instruments are part of the active scan path and what command is to be executed on them. The ILM holds information about the data lengths of the instruments in the network and the position of each instrument. The Discard Unit was introduced to allow the controller to discard garbage bits that are outputted by the network during the CSU cycles.

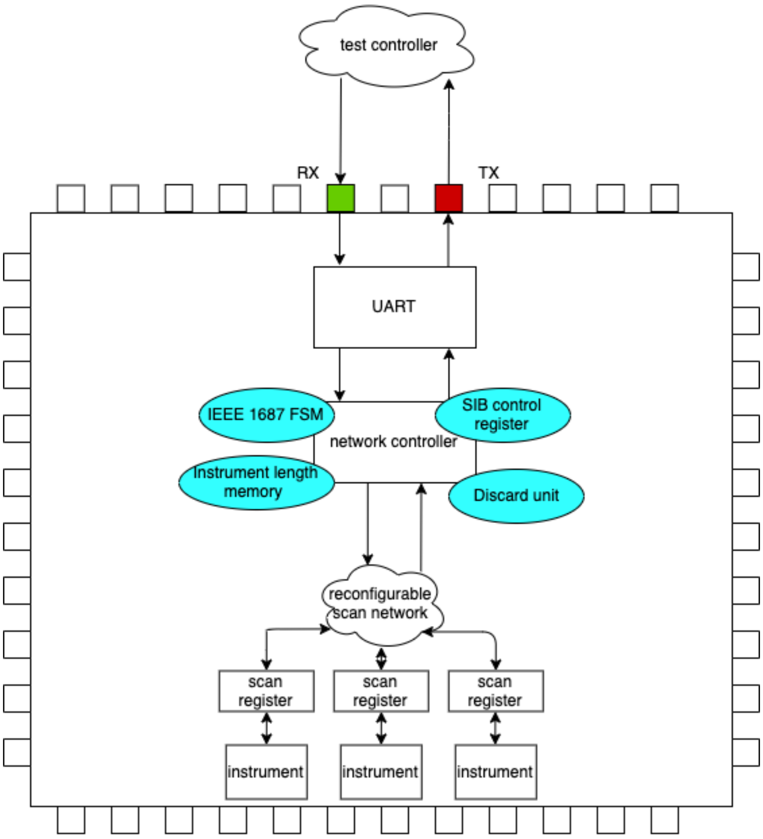


Figure 2.6: Full-featured case.

2.4.2 iApply group

iApply groups include a set of commands to specify the way to operate on an instrument such as reading from or writing to the hardware. Such commands are called Level-0 PDL commands, it consists of two types of commands, the setup and action commands[3]. Both the commands are sent by the test controller outside the device under test. The action commands is iApply command. Setup commands such as iRead or iWrite will be lined up and wait for the action commands. Algorithm 1 and 2 show the setup and action commands.

In the iRead command, the setup bits "00000000 00000000" which specifies the iRead commands will be given. In order to get the data from the network, the iRead function will insert the dummy bits into the network to push out the useful data.

In the iWrite command, the setup bits which specifies the iWrite commands are "01000000 00000000", the data bits for instruments '10101010' will be generated.

In the iApply command, firstly, it will judge the setup commands. If the commands are iRead, the iApply commands will send the sequence in the order of the setup bits, dummy bit size, and dummy bits. If the commands are iWrite, it will send the sequence in the order of the setup bits, data bit size, and data bits. Figure 2.7 shows an example of bit sequences of the writing and reading process. The first two bytes is the iWrite and iRead commands. The blue blocks sepecifying the address of the instruments will be accessed. Then the iApply command follows, which specifies the number of bytes that are required to be sent to the scan network. The last byte shows the data byte and the dummy bits.

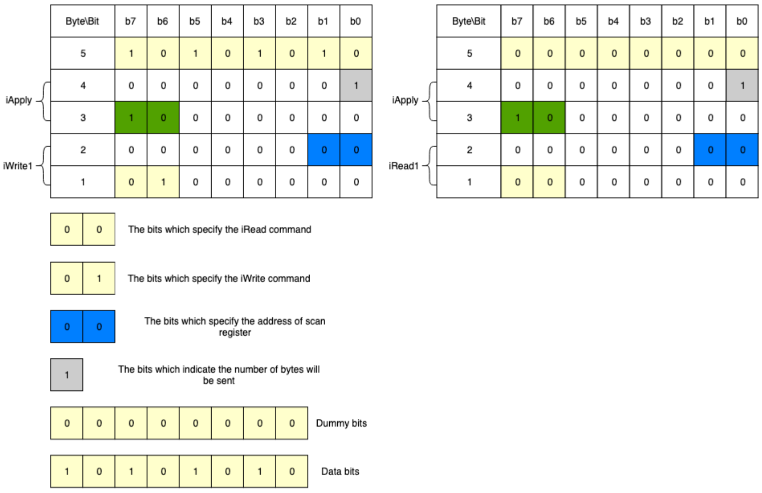
```
Algorithm1 Setup commands

def iRead(ScanRegister_address=0):
    setupBits = (0 << 15) V ScanRegisterAddress
    dummyBits = (0 << 7)
    return setupBits,dummyBits

def iWrite(ScanRegister_address=0):
    setupBits =(1 << 14) V ScanRegisterAddress
    dataBits  = '10101010'
    return setupBits,dataBits
```

```
Algorithm2 Action commands

def iApply(command):
    if command == iRead:
        dummySIZE=(1 << 15) V (length(dummyBits)/8)
        serialShift(setupBits)
        serialShift(dummySize)
        serialShift(dummyBits)
        incomingData = serialCapture(bufferSize)
    elif command == iWrite:
            dataSize =(1 << 15) V (length(dataBits)/8)
            serialShift(setupBits)
            serialShift(dataSize)
            serialShift(dataBits)
      return apply_cmd
```



Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
5	1	0	1	0	1	0	1	0
4	0	0	0	0	0	0	0	1
3	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0

Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
5	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1
3	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0

iApply
iWrite1
iApply
iRead1
0 0 The bits which specify the iRead command
0 1 The bits which specify the iWrite command
0 0 The bits which specify the address of scan register
1 The bits which indicate the number of bytes will be sent
0 0 0 0 0 0 0 0 Dummy bits
1 0 1 0 1 0 1 0 Data bits

Figure 2.7: Example of writing and reading process.

Chapter 3 Methodology

In this chapter we propose a hardware solution and a software solution to perform test and repair of reconfigurable networks. The solutions can handle any number of faults in the reconfigurable networks.

3.1 Hardware solution

Figure 3.1 shows the hardware solution which consists of the test block and the repair component in the network controller. The UART and network controller are from [3]. The testing of the reconfigurable network is initiated by an iTest command being sent from the test controller. The iTest command will enable the test block. After the testing process ends, the test block will get faulty scan registers locations. When the test controller sends the iRepair command to the UART, the faulty scan registers' locations will be sent to the repair component in the network controller. After that, when the test controller wants to access the faulty scan registers in the network, the repair component will check the address with the faulty scan registers' location. If the addresses are the same, the access process will be bypassed.

Figure 3.2 shows the bit sequence for both commands, and both commands will consist of 16 bits. The first byte '0111 1111' of iTest bit sequence specify the iTest command and the the first byte '0111 1110' of iRepair bit sequence specify the iRepair command.

3.1.1 Test Function

When the test controller send the iTest command to the device under test, the test function will be enabled. The overall idea of the test function is to detect and localize faulty scan registers. It has two processes, the FULLTEST and ONEBYONE process. First, the FULLTEST process will be executed to test if there are any faults in the the scan registers. If there are faults, the ONEBYONE process will be executed to test the scan registers one by one, which will localize the locations of the faults. Figure 3.3 shows the test block, which includes the sequence generator, the shift-update (SU) controller and the sequence detector.

Figure 3.4 shows the sequence generator, which has 4 parts controlled by an FSM. They are control bits generator, FULLTEST generator, ONEBYONE gener-

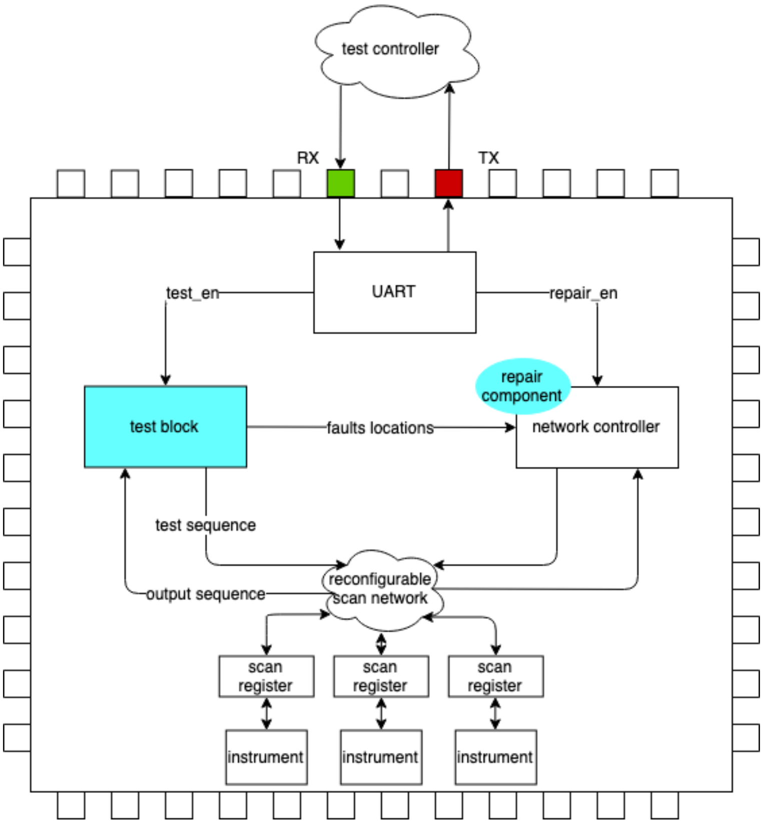
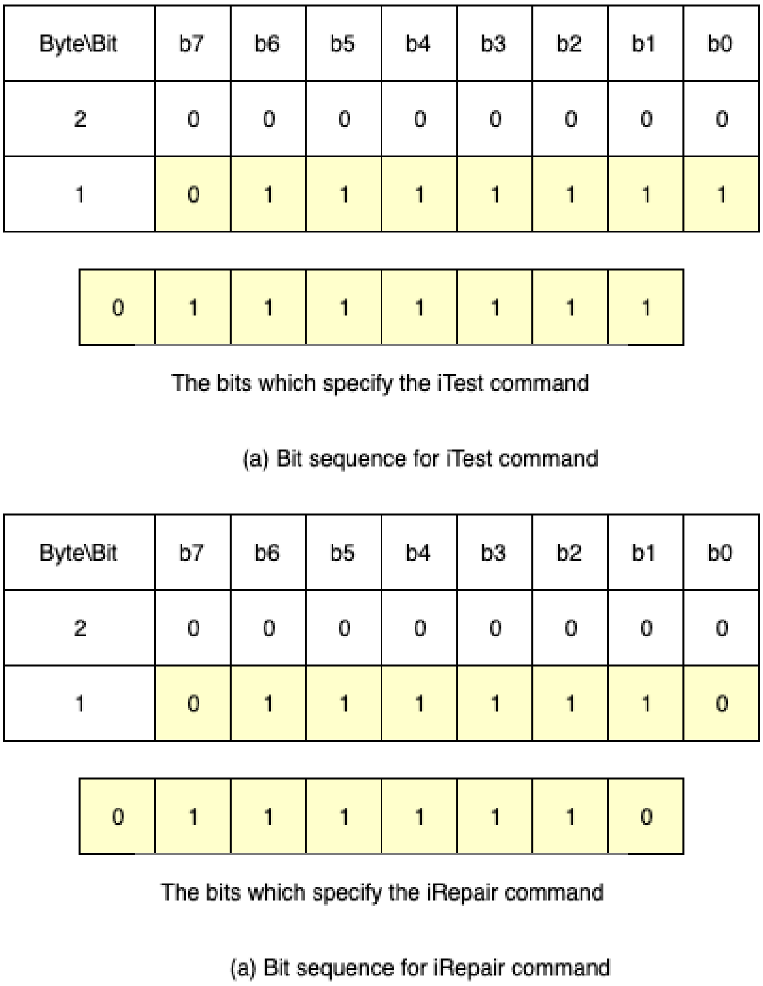


Figure 3.1: Connection diagram of the hardware solution.



Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
2	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1

0	1	1	1	1	1	1	1

The bits which specify the iTest command

(a) Bit sequence for iTest command

Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
2	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	0

0	1	1	1	1	1	1	0

The bits which specify the iRepair command

(a) Bit sequence for iRepair command

Figure 3.2: Bit sequence for iTest and iRepair commands.

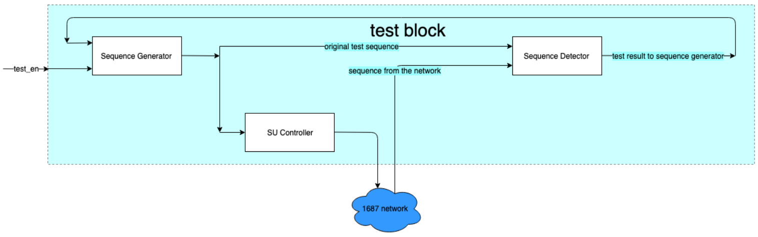


Figure 3.3: The overview of test block.

ator and repair register. The control bits generator is for generating the control bits for both FULLTEST and ONEBYONE processes. The FULLTEST and ONEBYONE generators is for generating the test sequences for both processes. The repair register stores the faulty scan register locations. The details of this block can be found in Appendix 1.

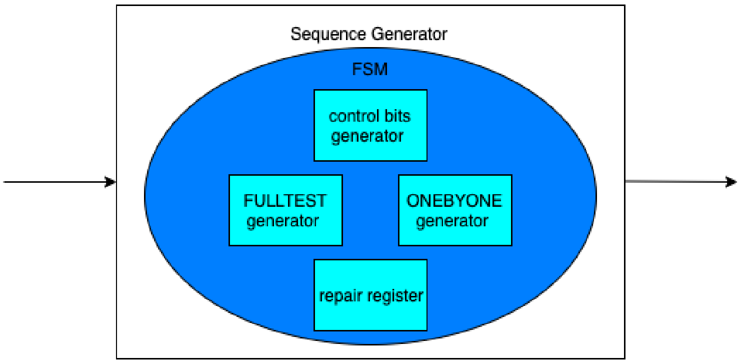


Figure 3.4: The overview of sequence generator.

Figure 3.5 shows the SU Controller. The SU Controller is for controlling the reconfigurable network. It has two components controlled by an FSM. The shift and update control the shift and update signals of the reconfigurable network. After all the control bits and the test data are shifted into the network, the test bits should be pushed out from the network to the sequence detector. Therefore the dummy bits generator will be enabled to send the dummy bits(0s) into the network. The details of this block can be found in Appendix 1.

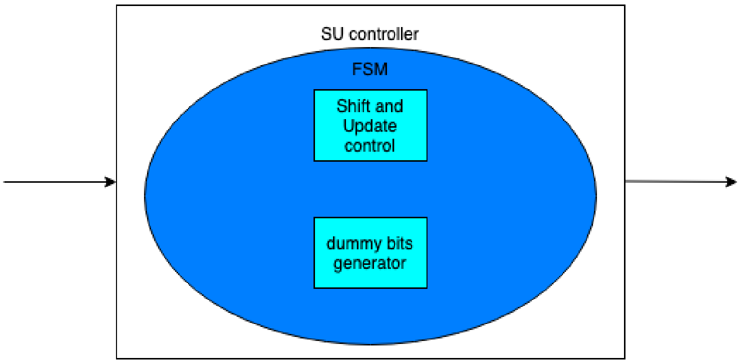


Figure 3.5: The overview of the SU controller.

Figure 3.6 shows the sequence detector, which consists of two parts, the delayer and the fault detector. The used delayer is to delay the original test sequence to match with the output of the reconfigurable network. When the sequences arrive, the fault detector compares them one bit by one bit until it finds a different bit, and the fault detector will send the test result to the sequence generator. The details of this block can be found in Appendix 1.

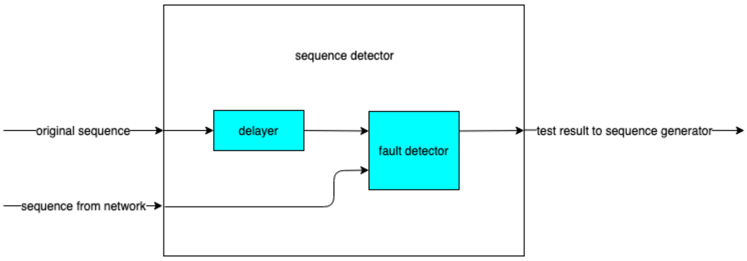


Figure 3.6: The overview of sequence detector.

Figure 3.7 (a) shows an example of a flat reconfigurable network with 3 SIBs and 3 scan registers where the length of the scan registers is 8 bit. we will illustrate the test function with this network. For illustration, we inject a fault in the second scan register, which is controlled by the second SIB. The test function operates as follows:

- The sequence generator will generate control bits and test sequence for the FULLTEST process, shown in Figure 3.7(b). In this example, the control bits should be '111', which will make the network include all the scan registers into the scan path. Then the FULLTEST sequence will be serially shifted into the network. As the scan registers are of length 8 bit, the complete test sequence is '10101010 10101010 10101010'. This sequence is shifted through the scan path. In case of no fault, the shift out sequence is the same as the shift in sequence. However, if there is a fault, for example a stuck at fault, the shift-in sequence and the shift-out sequence will not match.
- After the SU controller transfers the control bits and the FULLTEST test sequence to the SIB network, the SU controller will generate the dummy bits '00000000 00000000 00000000' into the SIB network. In this way, the test data can be pushed out from the network.
- After the two steps above, the sequence detector will receive the output data from the SIB network and the sequence detector will compare this data with the original test data. In this example, the sequence detector will find that the shift-in sequence and the shift-out sequence do not match. A fault prompt signal will be sent to the sequence generator.
- When the sequence generator receives the fault prompt signal, the ONEBY-ONE process will start, which are illustrated in Figure 3.7(c). The test sequence and the control bits for testing the first scan register will be generated. In this example, the control bit will be '100' to only include the first instrument and the test sequence will be '10101010'. The scan path is shown in red in Figure 3.7(c).
- Then the SU controller continue generates dummy bits into the network. In this step, the dummy bits should are '00000000'
- The sequence detector receives the output data from the network and the original test data. In this example, since the first scan register is not faulty, the fault prompt signal will not be enables.
- To include the second scan register in the scan path, the sequence generator generates the control bits "010" and the test sequence '10101010'. The scan path is shown in yellow in Figure 3.7(c).

• when the sequence detector receives both sequences, the mismatch will be detected, and the fault prompt signal will be enabled. The sequence generator saves the location of the faulty scan register into the repair register.

• The test block continues to work until all the scan registers have been tested. In this example the information in the repair register should be ’010’, which indicate the second scan register is faulty.

Table 3.1 shows the overall data for this example.

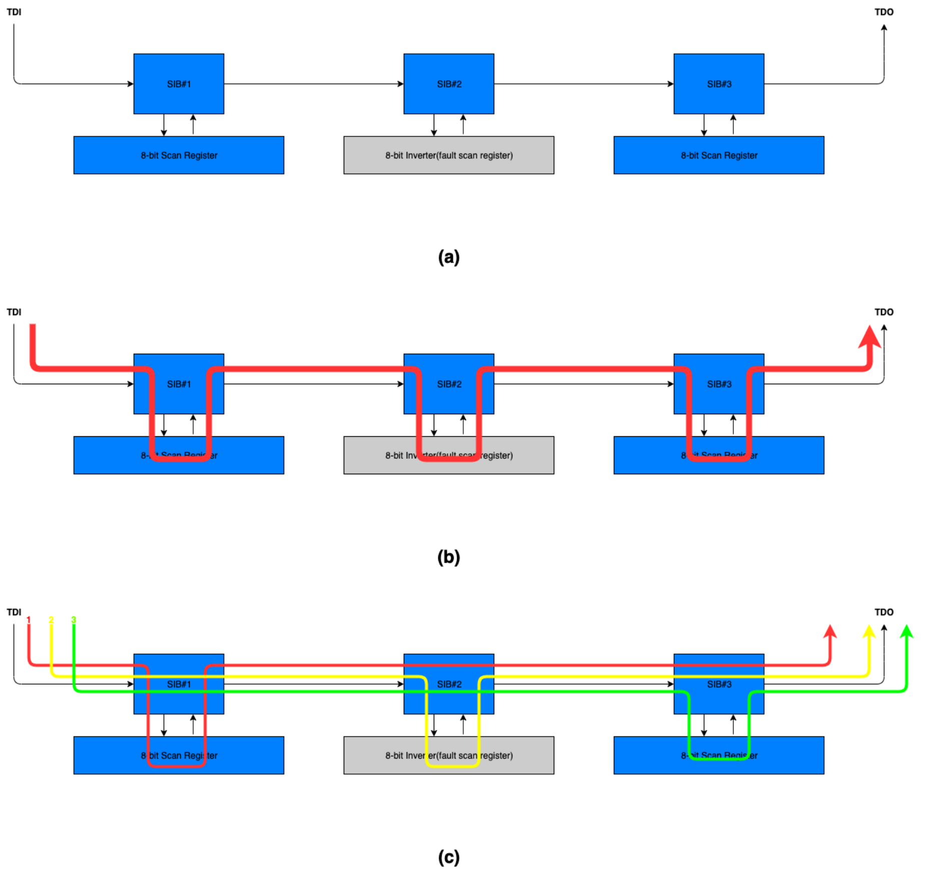


Figure 3.7: (a) 3-SIBs Network, (b) FULLTEST dataflow, (c) ONEBYONE dataflow.

3.1.2 Repair Function

The idea of the repair function is to bypass the faulty scan registers. When the iRepair commands are applied, the repair component will be enabled to save the faulty scan registers location from the repair register of sequence generator. After that, when the test controller wants to access faulty scan registers, the repair component will check the faulty scan register locations and automatically exclude

FULLTEST			ONEBYONE		
Control bits	Test bits	Dummy bits	Control bits	Test bits	Dummy bits
3	24	24	9	24	24

Table 3.1: Data for the reconfigurable network with 3 SIBs and 3 scan registers each of length 8.

them from the scan path.

Figure 3.8 shows the ASMD diagram of the repair component. We will use the reconfigurable network with 3 SIBs and 3 scan registers of length 8 bit(Figure 3.7(a)) to explain the repair component.

Before we explain this ASMD, it's necessary to mention that the FAULTScanRegister_reg is a register array of 8 bits-std_logic_vector type registers. In this example, the number of std_logic_vector registers. The FAULTScanRegister_counter is a 8 bits-std_logic_vector type counter and the SCR_counter is a integer type counter.

First, in the IDLE state, reset puts everything in default values. When the repair component starts, the state machine will go to the S0 state.

In the S0 state, the component will check the first bit of SCR_test_in signal, which is the signal that transfers the faulty scan registers location from the repair register of sequence generator. As we discussed in the last subsection, the information in the repair register is '010'. Since the first scan register is not faulty, nothing will be saved in the first register of FAULTScanRegister_reg array. After that, in the next rising edge of the clock cycle, the state machine will move to the S1 state.

In S1 state, the SCR_counter adds 1, which will make the S0 state check the second bit of the repair register. The FAULTScanRegister_counter will also add 1, which will transform the second scan register location to binary type(00000001). If there is a faulty scan register location in the next state, the fault location will be saved at the second vector of the FAULTScanRegister_reg array.

Then the state machine go to S0 state again, since the second bits of repair register is '1', the faulty scan register location will be saved as "0000 0001" in the second register of FAULTScanRegister_reg array.

The component will stop until every bits of repair register has been checked. As expected, the component will only save the second scan register location into the FAULTScanRegister_reg array.

3.2 Software solution

Figure 3.9 shows the software solution. The test and repair functions are implemented in the test controller and nothing will be modified in the device under test. By applying the iTest and iRepair commands, the test and repair functions will be enabled. First, the iTest command will be applied to enable the test function. During the test process, the faulty scan registers locations will be detected and saved into a file. After that, the iRepair command will be applied to enable the repair function. During the repair process, when the test controller wants to

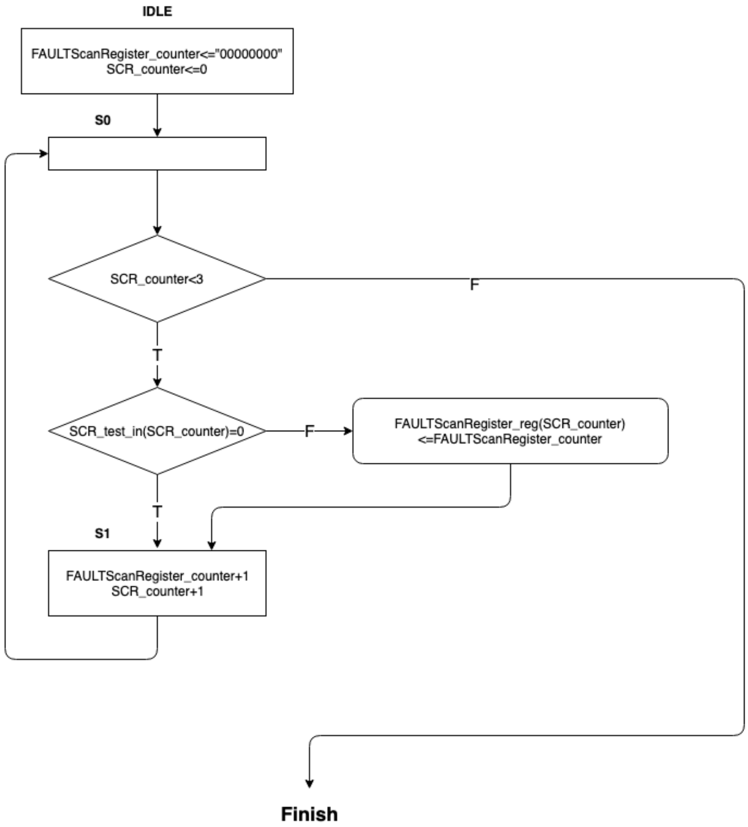

IDLE
FAULTScanRegister_counter<="00000000"
SCR_counter<=0
S0
SCR_counter<3
F
T
SCR_test_in(SCR_counter)=0
F
FAULTScanRegister_reg(SCR_counter)
<=FAULTScanRegister_counter
T
S1
FAULTScanRegister_counter+1
SCR_counter+1
Finish

Figure 3.8: ASMD diagram of the FSM for repair component.

send the iWrite or iRead commands which related to the faulty scan registers, the iRepair commands will check the faults location file and the iWrite or iRead commands will be bypassed.

3.2.1 Test function

We will use the same reconfigurable network as Figure 3.7(a) to explain the test process of software solution. Figure 3.10 shows the work flow of the test process. The test function first applies the ALLWrite command to write test data '10101010' into all scan registers one by one. Figure 3.11 shows the bit sequence of the ALLWrite process.

Then the ALLRead command will be applied to send the dummy bits '00000000' to the scan network so that all the test data can be pushed out from the scan network back to the test controller. In this example, since the second scan register is an inverter which is considered as a fault, the second incoming data should be '01010101'. Figure 3.12 shows the bit sequence of the ALLRead process.

The data flow of ALLWrite and ALLWrite process are shown in Figure 3.13.

After that, the iTest function will look for the same data as incoming data in the original data buffer. If there is same data, the corresponding scan register is not faulty. If not, the corresponding scan register has a fault and the address of the scan register will be stored in the FaultScanRegisters file. In this example the second scan register's location will be saved into this file. The algorithm of iTest commands can be found in Appendix 1.

3.2.2 Repair Function

The idea of the repair function is to check what PDL commands use. If PDL commands want to access a faulty scan register, the repair function will exclude this PDL command from the iApply group. Hence, the scan path will be modified such that the defective scan register is excluded. For example in Figure 3.13, the commands where the second scan register is bypassed, and the test controller only send the commands which is related to the normal scan registers. The algorithm of iRepair commands can be found in Appendix 1.

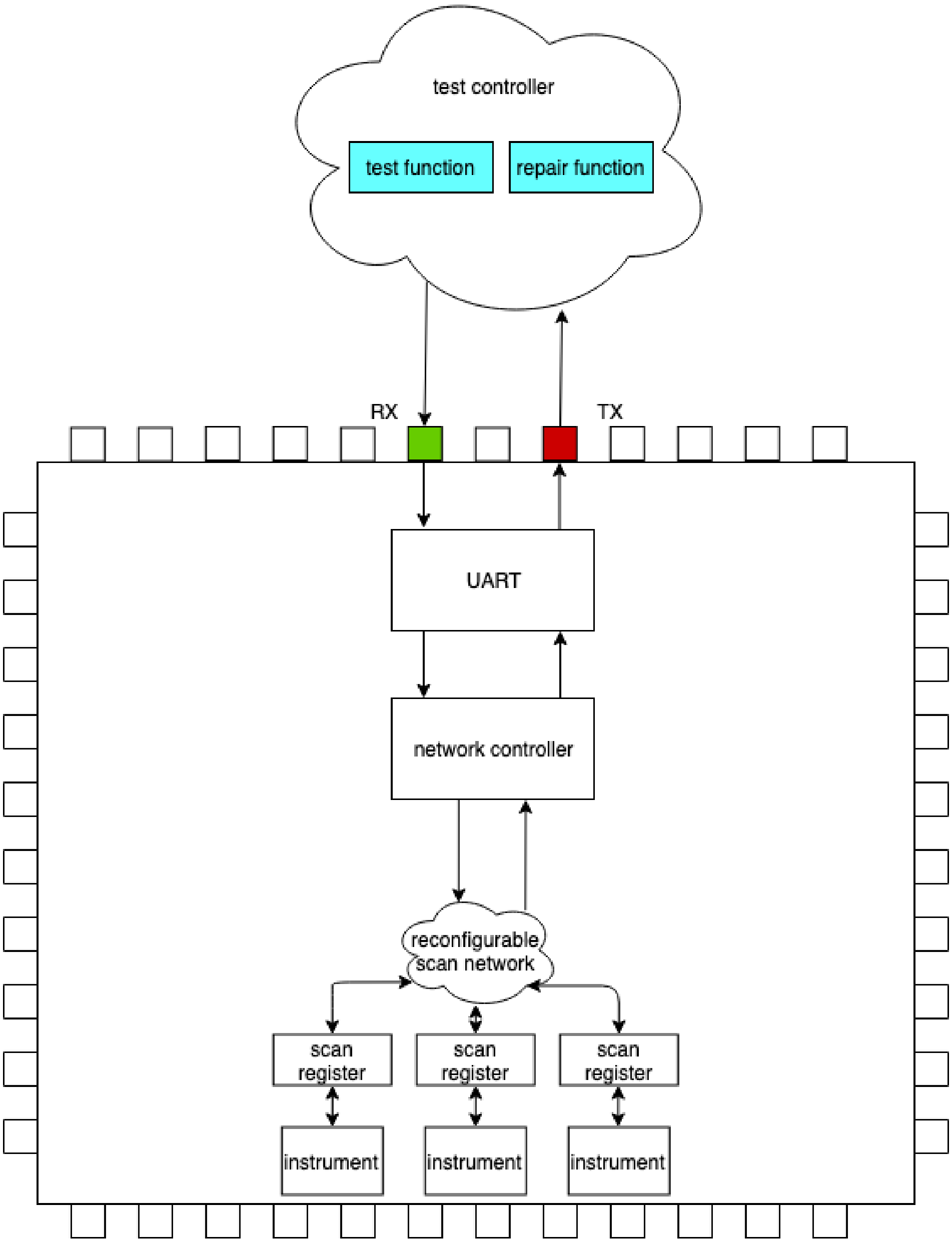


Figure 3.9: Connection diagram of the software solution.

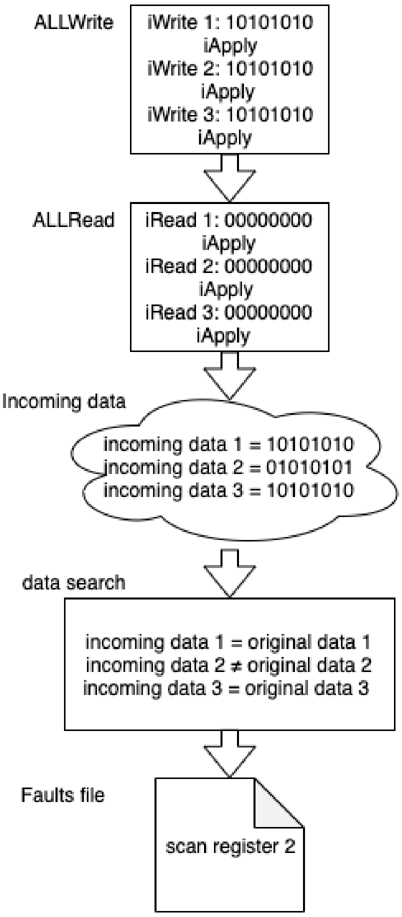
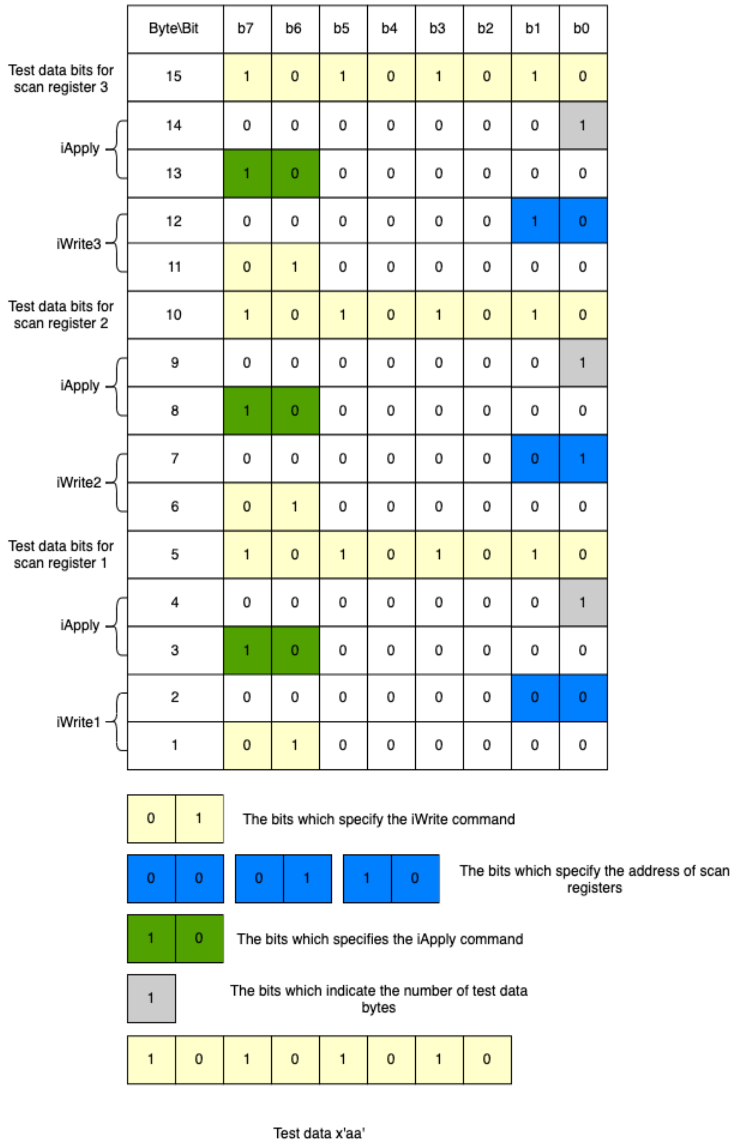


Figure 3.10: Example of test process of software solution.



	Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
Test data bits for scan register 3	15	1	0	1	0	1	0	1	0
iApply	14	0	0	0	0	0	0	0	1
	13	1	0	0	0	0	0	0	0
iWrite3	12	0	0	0	0	0	0	1	0
	11	0	1	0	0	0	0	0	0
Test data bits for scan register 2	10	1	0	1	0	1	0	1	0
iApply	9	0	0	0	0	0	0	0	1
	8	1	0	0	0	0	0	0	0
iWrite2	7	0	0	0	0	0	0	0	1
	6	0	1	0	0	0	0	0	0
Test data bits for scan register 1	5	1	0	1	0	1	0	1	0
iApply	4	0	0	0	0	0	0	0	1
	3	1	0	0	0	0	0	0	0
iWrite1	2	0	0	0	0	0	0	0	0
	1	0	1	0	0	0	0	0	0

0 1 The bits which specify the iWrite command

0 0 0 1 1 0 The bits which specify the address of scan registers

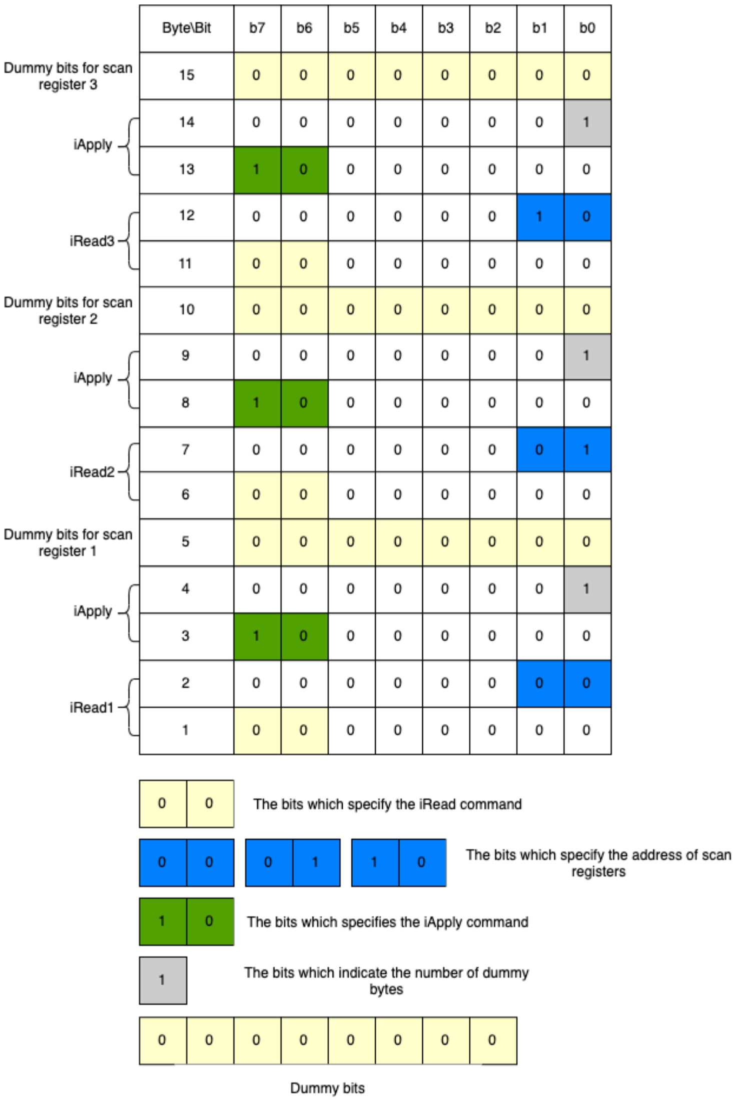
1 0 The bits which specifies the iApply command

1 The bits which indicate the number of test data bytes

1 0 1 0 1 0 1 0

Test data x'aa'

Figure 3.11: Bit sequence of AllWrite process for 3-SIBs network.



	Byte\Bit	b7	b6	b5	b4	b3	b2	b1	b0
Dummy bits for scan register 3	15	0	0	0	0	0	0	0	0
iApply	14	0	0	0	0	0	0	0	1
	13	1	0	0	0	0	0	0	0
iRead3	12	0	0	0	0	0	0	1	0
	11	0	0	0	0	0	0	0	0
Dummy bits for scan register 2	10	0	0	0	0	0	0	0	0
iApply	9	0	0	0	0	0	0	0	1
	8	1	0	0	0	0	0	0	0
iRead2	7	0	0	0	0	0	0	0	1
	6	0	0	0	0	0	0	0	0
Dummy bits for scan register 1	5	0	0	0	0	0	0	0	0
iApply	4	0	0	0	0	0	0	0	1
	3	1	0	0	0	0	0	0	0
iRead1	2	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0

0 0 The bits which specify the iRead command

0 0 0 1 1 0 The bits which specify the address of scan registers

1 0 The bits which specifies the iApply command

1 The bits which indicate the number of dummy bytes

0 0 0 0 0 0 0 0

Dummy bits

Figure 3.12: Bit sequence of ALLRead process for 3-SIBs network.

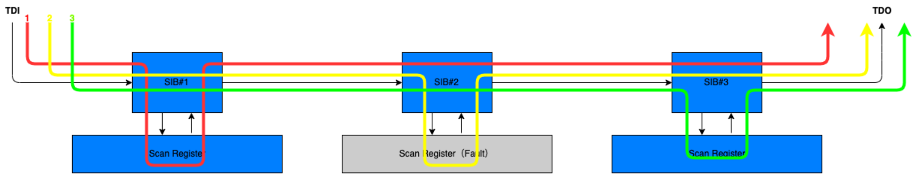


Figure 3.13: Data flow of ALLWrite and ALLRead process.

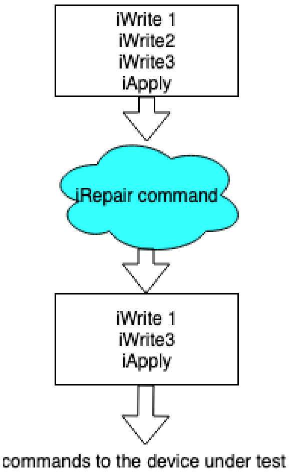


Figure 3.14: Example of the repair process.

Chapter 4 Experimental results

4.1 Experimental setup

We performed two types of experiments to test the performance of hardware and software solutions. In experiment 1, we have prepared three flat reconfigurable networks with 50, 100 and 150 SIBs where each scan register is of length 8 bits. For each network, we inject one fault in the first scan register and measured the area consumption and data overhead of both solutions. In experiment 2, we prepared a flat reconfigurable network with 150 SIBs and inject the different number of faults in the scan registers of the reconfigurable network. The data overhead is measured for both solutions.

For the FPGA board, we chose a Digilent Nexys 4 DDR board, which is based on the latest Artix FPGA from Xilinx. The baud rate we used is 115200. For coding, synthesis, and implementation we used VHDL and the Xilinx Vivado 2019.1 EDA tool.

For the test controller part, we used the Python programming language. For the Python tools, we chose Pycharm.

4.2 Experimental methodology

For area consumption, we used the resource utilization from the report in Vivado. We used Configurable Logic Block (or logic block, LB for short) which consists of two logic slices (Slice M and Slice L) and represents fundamental building block of the Xilinx FPGA fabric [8]. The equation for calculating CLB is:

$$CLB = 8 \times LUT + 16 \times FF \tag{4.1}$$

Look-up Table (LUT) is a collection of gates hardwired on the FPGA. An LUT stores a predefined list of outputs for every combination of inputs. A flip-flop (FF) is the smallest storage resource on the FPGA. Each flip-flop in a CLB is a binary register used to save logic states between clock cycles on an FPGA circuit [8].

The data overhead is the data exchanged between the test controller and the device under test.

4.3 Results of experiment 1

Table 4.1 shows the results of experiment 1. For the hardware solution, the area consumption increases linearly with the number of SIBs. For the software solution, no additional area is needed, since the solution is implemented with software. Table 4.2 shows the area consumption for the hardware solution. We see that the CLB of all blocks increases with the size of the reconfigurable network because the hardware solution need to handle more data with the increasing number of SIBs. In addition, the area consumption of the sequence generator is the largest among the three blocks, since all the test sequences are generated here. The area consumption of the reconfigurable network with different sizes are shown in Figure 4.1, we only included the scan registers into the reconfigurable network. Compares the area of the reconfigurable and the proposed hardware solution, the area of the proposed hardware solution is much lower than that of the reconfigurable network.

Number of SIBs	Hardware solution				Software solution				Area of reconfigurable network	Test bits without both solutions
	Test	Repair	Area	Data comparison(%)	Test	Repair	Area	Data comparison(%)		
	Data overhead	Data overhead	CLB		Data overhead	Data overhead	CLB			
50	16	16	83	0.19	2432	0	0	29.62	145	18500
100	16	16	130	0.06	4832	0	0	18.17	250	67000
150	16	16	161	0.03	7232	0	0	13.17	433	98900

Table 4.1: Result of experiment 1

Number of SIBs	Hardware solution											
	Sequence generator			Sequence Detector			Test SIB controller			Total		
	FF	LUT	CLB	FF	LUT	CLB	FF	LUT	CLB	FF	LUT	CLB
50	10	280	36	34	166	23	5	186	24	49	632	83
100	11	489	62	35	239	32	5	282	36	51	1010	130
150	12	548	69	36	324	43	5	387	49	53	1295	161

Table 4.2: Area consumption for the three blocks in the hardware solution

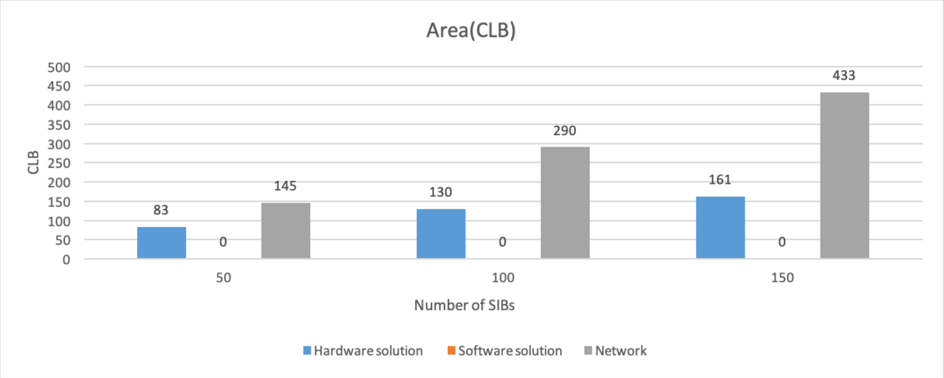


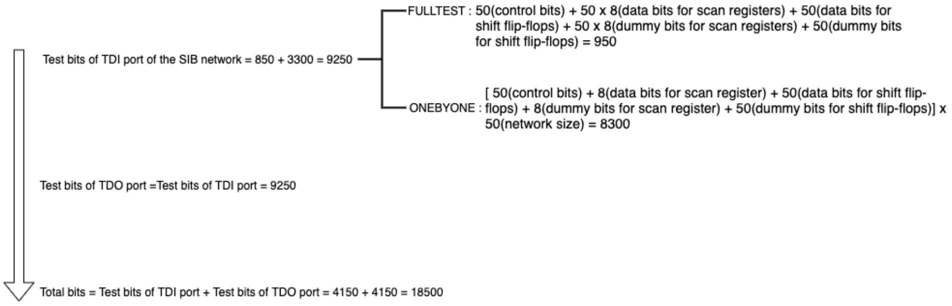
Figure 4.1: Area comparison.

For the hardware solution, the data overhead of the test and repair process is 16 bits each. The reason is that the test controller only sends the commands (iTest or iRepair) with 16 bits to the device under test, and the device under test will automatically start the test or repair process. Actually, independently of the size of the network and the number of the faulty scan registers, the data overhead of the hardware solution will always be the constant, which is shown in the experiment2.

For the software solution, the data overhead of the test process increases linearly with the number of SIBs. This is because the test controller needs to generate more test sequences when the number of SIBs increases. During the repair process, when these scan registers are faulty, the test controller will ignore these commands related to faulty scan registers. Therefore, nothing will be send to the device under test when the test controller want to access the faulty scan registers.

We also measured the test bits which test the reconfigurable network without proposed solutions. Figure 4.2 shows the calculation process of the test bits for the network with 50 SIBs. During the FULLTEST process, the 50 control bits will be firstly generated. Then the data bits for testing will be generated for each 8-bits scan registers. The 50 data bits for filling the shift flip-flops in the SIBs will be generated at the same time. After that, the dummy bits for both shift flip-flops and the scan registers will be generated. The total number of bits for the FULLTEST process is 950. During the ONEBYONE process, when testing the one scan register, the 50 control bits will be generated. Then the 8 data bits for scan register will be generated, and the 50 data bits for filling the shift flip-flops will be generated as well. The dummy bits will also be generated, which is the same as the data bits. The total number of bits for the ONEBYONE process should be 8300. Therefore, the bits exchanged for TDI or TDO should be 9250, and the total bits exchanged should be 18500.

The result of data overhead for the test process of both the solutions and the test bits without proposed the solutions also shows in Table 4.1. Figure 4.3 shows the data overhead with the method without proposed solutions. The hardware solution saves more significant data compared to the software solution.



Test bits of TDI port of the SIB network = 850 + 3300 = 9250

FULLTEST : 50(control bits) + 50 x 8(data bits for scan registers) + 50(data bits for shift flip-flops) + 50 x 8(dummy bits for scan registers) + 50(dummy bits for shift flip-flops) = 950

ONEBYONE : [50(control bits) + 8(data bits for scan register) + 50(data bits for shift flip-flops) + 8(dummy bits for scan register) + 50(dummy bits for shift flip-flops)] x 50(network size) = 8300

Test bits of TDO port =Test bits of TDI port = 9250

Total bits = Test bits of TDI port + Test bits of TDO port = 4150 + 4150 = 18500

Figure 4.2: Test bits calculation of the SIB network with 50 SIBs.

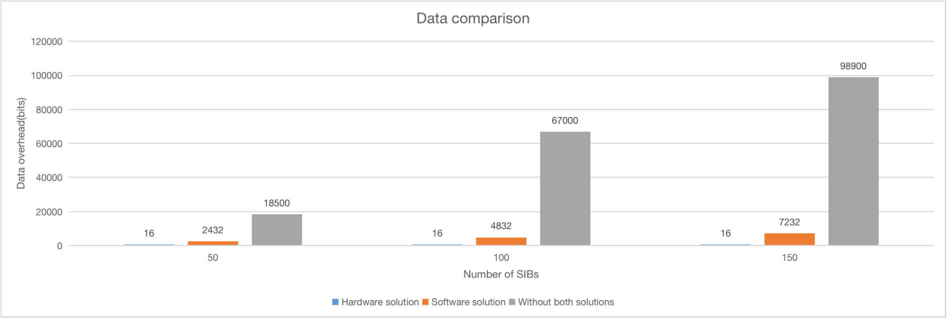


Figure 4.3: Data comparison.

4.4 Results of experiment 2

Table 4.3 shows the results of this experiment. The data overhead for proposed solutions will not change with the number of faults. For the hardware solution, as we discussed in experiment 1, the data overhead of the test process is always 16 bits. For the software solution, the data overhead of the test process is also fixed, because the number of bits of the test data is the same regardless of the number of faulty scan registers. From the result, the data overhead of test process for the software solution is higher than the overhead of the hardware solution.

For the repair process, the data overhead of the hardware solution is 16 bits. For the software solution, the data overhead is 0 bits.

Faults	Hardware solution		Software solution	
	Test	Repair	Test	Repair
	Data overhead	Data overhead	Data overhead	Data overhead
1	16	16	7232	0
2	16	16	7232	0
3	16	16	7232	0
4	16	16	7232	0
5	16	16	7232	0
25	16	16	7232	0
50	16	16	7232	0
100	16	16	7232	0
150	16	16	7232	0

Table 4.3: Result of experiment 2

Chapter 5 Conclusion

In this thesis, we proposed and compared two solutions for detecting and repairing faulty scan registers in reconfigurable network, two experiments were done to compare area and data overhead. The results show that both the solutions have their individual advantages. The hardware solution has the advantage of low data overhead and the software solution has the advantage of low area consumption.

In this thesis, we only explored the test and repair system on flat reconfigurable network. In the future, we think the test and repair functions could be made on other kinds of networks like hierarchical networks, multiple networks, daisy-chained network, etc. We could also explore the way in which the same function can be made based on other functional port interfaces like I2C and others.

Chapter 6 Appendix 1

6.1 Sequence generator

Figure 6.1 shows the sequence generator. The signals will be explained in the following parts:

INPUT PORTS:

• clk: Clock signal

• test_en: The enable signal from UART, when it becomes ’1’, the test function start to work.

• test_out: The fault prompt signal from sequence detector. If the sequence detector detects fault scan registers, the signal becomes ’1’, otherwise becomes ’0’.

• enable_from_detector: The enable signal from sequence detector. When the sequence detector finishes its work, the signal becomes ’1’, which enable the FSM in the sequence generator for continuing to send control bits and test sequence for the subsequent scan register.

OUTPUT PORTS:

• data_to_detector1: The signal which send the FULLTEST test sequence to sequence detector.

• data_to_SIB1: The signal which send the FULLTEST test sequence to SU Controller.

• data_to_detector2: The signal which send the ONEBYONE test sequence to sequence detector.

• data_to_SIB2: The signal which send the ONEBYONE test sequence to SU Controller.

• detector_en: The enable signal which enables the sequence detector.

• control_en: The enable signal which enables the SU Controller.

• SCR_control: The signal which send the SIB control bits to SU Controller.

• network_en: The enable signal which enables the SIB Network.

• fulltest: The signal which tells other blocks the current test mission for testing all scan registers in the SIB network.

• onebyone: The signal tells other blocks the current test mission for testing scan registers one by one.

• SCR_test: this signal sends the fault scan registers’ locations to the repair component in the network controller from [3].

- repair: this signal informs the network controller from [3] that there are fault scan registers in the network, and need to repair them. If there are fault locations in the SCR_test signal, it becomes ’1’, if not , it becomes ’0’.

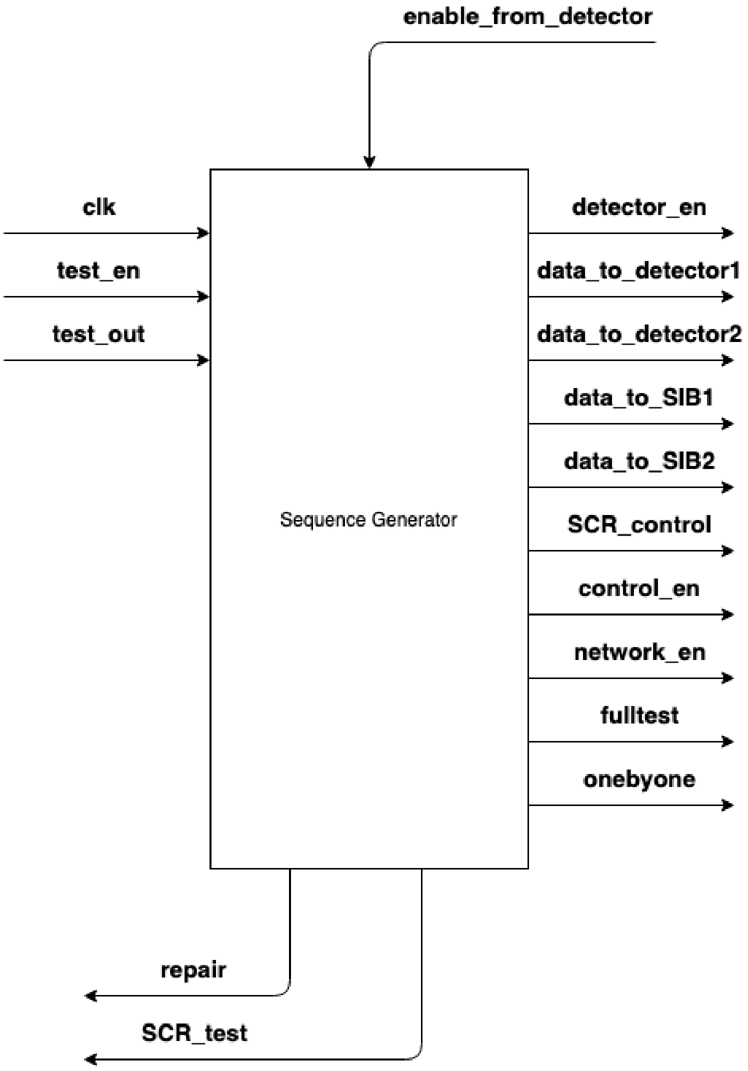


Figure 6.1: Sequence generator.

Figure 6.2 shows the simplified FSM of the sequence generator for operating with the flat reconfigurable network in Figure 3.7 (a).

In the IDLE state, every signal will be set to the default value. When the test_en signal becomes ’1’, the sequence generator starts working. At the rising edge of the clk signal, the state machine jumps to the test_fault state. In this state, the SU controller, SIB network and sequence detector will be enabled to work. The fulltest signal becomes ’1’ to inform other blocks that the current mission is testing all scan registers together. The corresponding signals of other blocks will be set up accordingly.

After sending the test sequence, the state machine will be waiting for the response of the sequence detector. When the sequence detector finish its work and then the enable_from_detector signal will become ’1’, the state machine will continue working. At the same time, the block will check the test_out signal. If it is ’0’, which means there are not faulty scan registers in the network, the state machine jumps to the finish state and the Test function finish. If the test_out signal is ’1’, the ONEBYONE test process begins in next clock cycle. In this example, this signal is ’1’.

Besides, the first bit of SCR_control signal will be set to ’1’, which will control the first SIB of the network to be open. Then the state machine will jump to the localization state. In this state, the onebyone signal, control_en, detector_en, and the network_en become ’1’ which makes another blocks switch to the ONEBYONE test mode.

After the localization, the sequence generator will be still waiting for the enable_from_detector signal. When it changes to ’1’, in this example, since the first scan register is not the faulty one, the test_out signal will get the value of ’0’. The sequence generator will test the next scan register, This time the faulty scan register will be detected and the test_out changes to ’1’. The state machine jumps to the repair state.

In the repair state, the location of the fault scan register will be stored in SCR_reg signal. The value of SCRunderline reg signal is "010". After that the system starts checking the last scan register, the test_out signal will get ’0’ as expected.

From now, all the scan registers have been tested, and the repair signal will become ’1’ and the SCR_reg signal gives its value to SCR_test port. when the repair function starts to work, the repair component in the Network Controller will get the value of the repair signal and the faulty scan register information of the SCR_test port.

Above all is how the sequence generator works with a flat reconfigurable network (Figure 3.7 (a)).

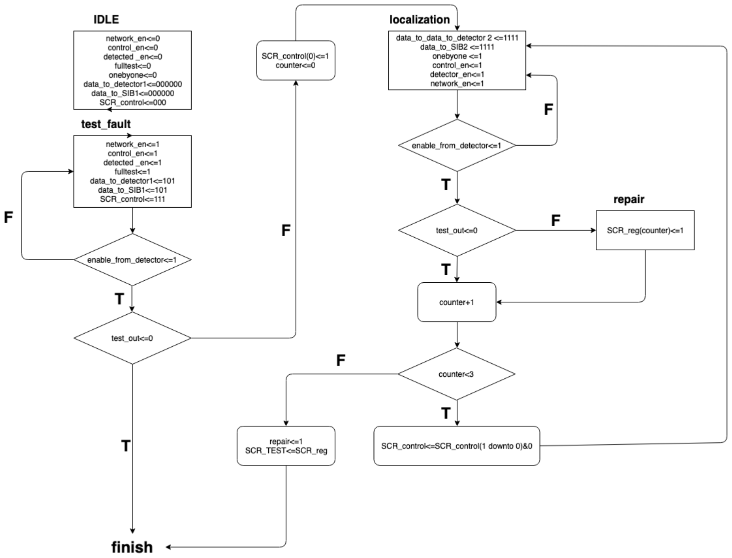


Figure 6.2: ASMD diagram of the main FSM for sequence generator.

6.2 Sequence detector

The detect function consists of two parts, the delayer and the sequence generator, which connected with each other in the top block like Figure 6.3. The corresponding ports will be explained below:

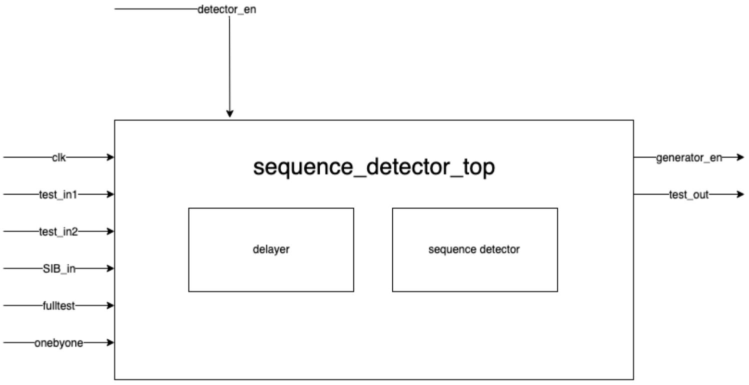


Figure 6.3: Sequence detector top block.

INPUT PORTS:

• clk: Clock signal.

• detector_en: The enable signal from sequence generator which enables the whole detector module to work.

• test_in1: The signal which receive the FULLTEST sequence from sequence generator .

• test_in2: The signal which receive the ONEBYONE sequence from sequence generator .

• SIB_in: The signal which receives the data from the SIB network.

• fulltest: The signal which receives the FULLTEST signal from sequence generator .

• onebyone: The signal which receives the ONEBYONE signal from sequence generator .

OUTPUT PORTS:

• generator_en: The enable signal that allows the sequence generator continue working.

• test_out: The signal which shows if the sequence is the same or not. If the sequence detector detects fault scan registers, the signal becomes '1', otherwise it becomes '0'.

The reason of using a delayer is that, when the generator sends the test data to the SIB network and sequence detector, the data from the test_in1and the test_in2 will arrive immediately. However, the output data from the SIB network will not arrive at the same time. Therefore the detector module will be waiting for the output data(SIB_in) for few clock cycles. In this way, all test data will arrive at the detector at the same time. Figure 6.4 is the delayer. we will explain how it works in details below:

Figure 6.5 is the simplified ASMD diagram of the delayer in the case of the 3-SIBs network. we will explain how it works with this example.

In the IDLE state, every signal will be set to the default value. The state machine will keep in this state until the detector_en becomes '1'. At the same time, the delayer and the SIB network receive the test sequence.

After the delayer enabled, the FULLTEST process will start first. Therefore, the state machine jumps to the FULLTEST branch.

The first arriving sequence is the control bits, in our case is 3 bits("111"), so the count_for_control state need to wait for 3 clock cycles.

Next is the actual data, as mentioned in the previous section, the data bits is 3("101"). Therefore, the state machine keeps in count_for_data1 state for 3 clock cycles.

From now the SU Controller will send 3 dummy bits into the SIB network to controlling the SIB network shift out the data. This time the state machine jumps to count_for_dummy1 state and wait for 3 clock cycles. In every clock cycle of this state, the data out signal will output the data one bit by one bit. At the same time, the enable signal enables the sequence detector to receive the output data one bit by one bit.

In the end, all test sequence for FULLTEST has been shifted to the sequence detector. The state machine moves to the finish state.

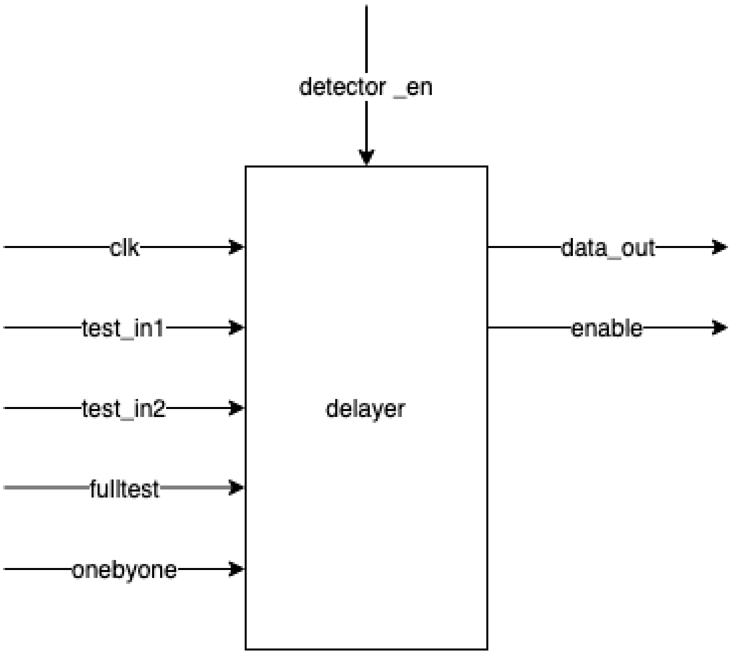


Figure 6.4: Delayer.

After the FULLTEST process finishes, in this example, since the second scan register is the faulty one, the system starts to test scan registers one by one. From the state machine we can see that the only different between ONEBYONE and FULLTEST is the state machine need to wait 1 clock cycles in count_for_data2 state and count_for_dummy2 state.

Above all are how the delayer works with the flat reconfigurable network (Figure 3.7 (a)). Next, the details of the sequence detector will be introduced.

Sequence detector is the block that detects the faulty scan register in the reconfigurable network, Figure 6.6 shows the block diagram of it.

Figure 6.7 shows the ASMD diagram of the FSM for the sequence detector. We will still use the network in Figure 3.7 (a) to explain the FSM.

In the IDLE state, all signals are set to ’0’. When the enable signal becomes ’1’, the whole block start to work. In the FULLTEST process. The detector will compare every bit come from the SIB network and the delayer one by one. The FSM keep in the datatest_full state until it found a different bit. If there is no fault in the network, the test_out signal and the generator_en signal will become ’0’ and ’1’ respectively to inform the sequence generator finishes the test function. In this example, the second scan register is a faulty one. So both the test_out signal and the generator_en signal will become ’1’. Then the sequence generator starts the ONEBYONE process. The detector continues to test the scan registers one by one until all scan registers have been tested.

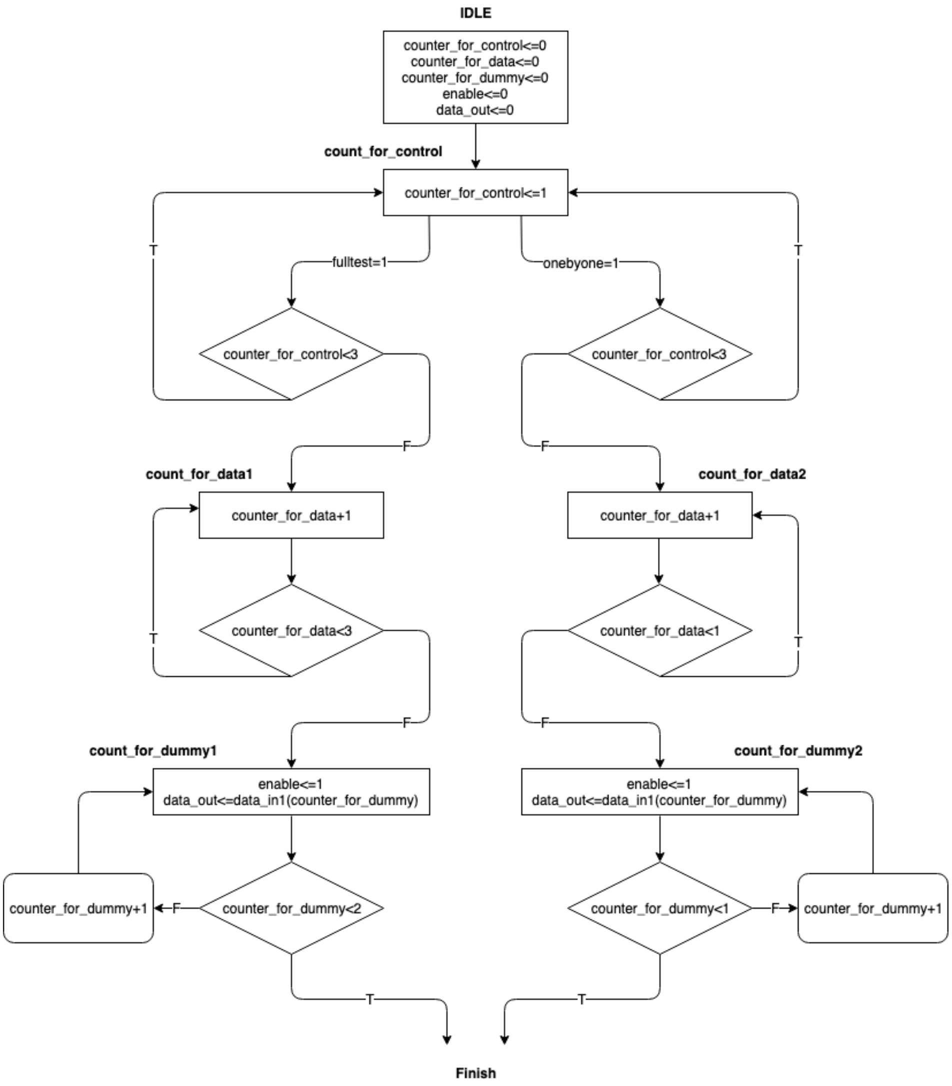


Figure 6.5: ASMD diagram of the main FSM for delayer.

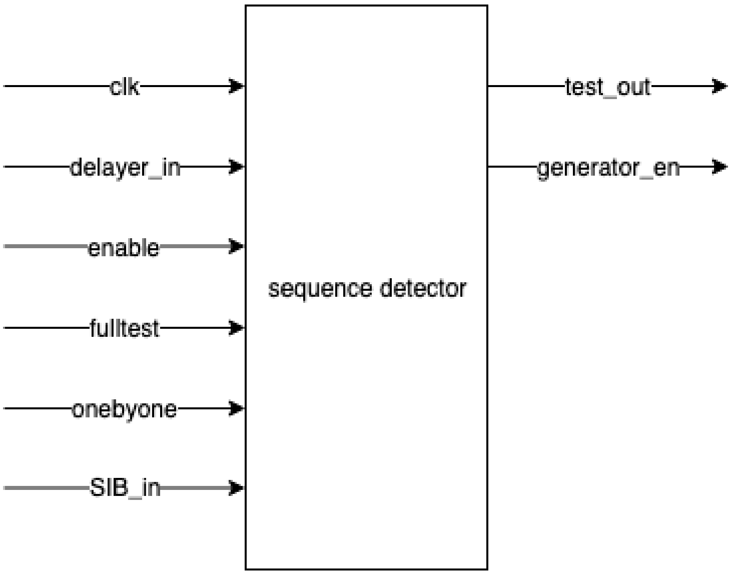


Figure 6.6: Sequence detector.

6.3 SU controller

The SU controller will shift the test sequence and generate the dummy bits into the SIB network. It controls the network by enabling the shift and update signals of the SIB network. Figure 6.8 shows the block diagram of the test network controller. The discussion about the SU controller will be made below:

Figure 6.9 shows the ASMD diagram of the FSM for The SU Controller. we will still use the network in Figure 3.7 (a) to explain how the SU controller works.

In IDLE state, everything will set to default value, when the control_en becomes '1', the SU controller start to work. The first data the SU controller received are the FULLTEST control bits. Therefore, the next state is shift_control state which makes the shift_en signal becomes '1' and control the reconfigurable network shift in the control bits. In our case, the number of bits is 3 ("111"). Therefore, the state machine will keep in this state for 3 clock cycles. In next state, update_control state, the shift_en signal becomes '0' and the update_en signal becomes '1' which stops the network shift bit into itself and update the control bits into the Update flip-flop. Then all the SIBs open.

In shift_fulltest state, the SU controller receives the FULLTEST bits ("101") from sequence generator. The shift_en signal changes to '1' again and shift the test data into the shift register and the scan register in the SIB network. After 3 clock cycles, the network gets all test bits.

After that, the test bits need to be shifted out from the network and be tested by the sequence detector. Therefore, in dummybits_full state, the SU controller will shift 3 dummy bits ("000") into the network. The FULLTEST process finish.

Then the system changed to ONEBYONE process. the SU controller will do the same work as FULLTEST process except it will shift 1 bits in shift_onebyone

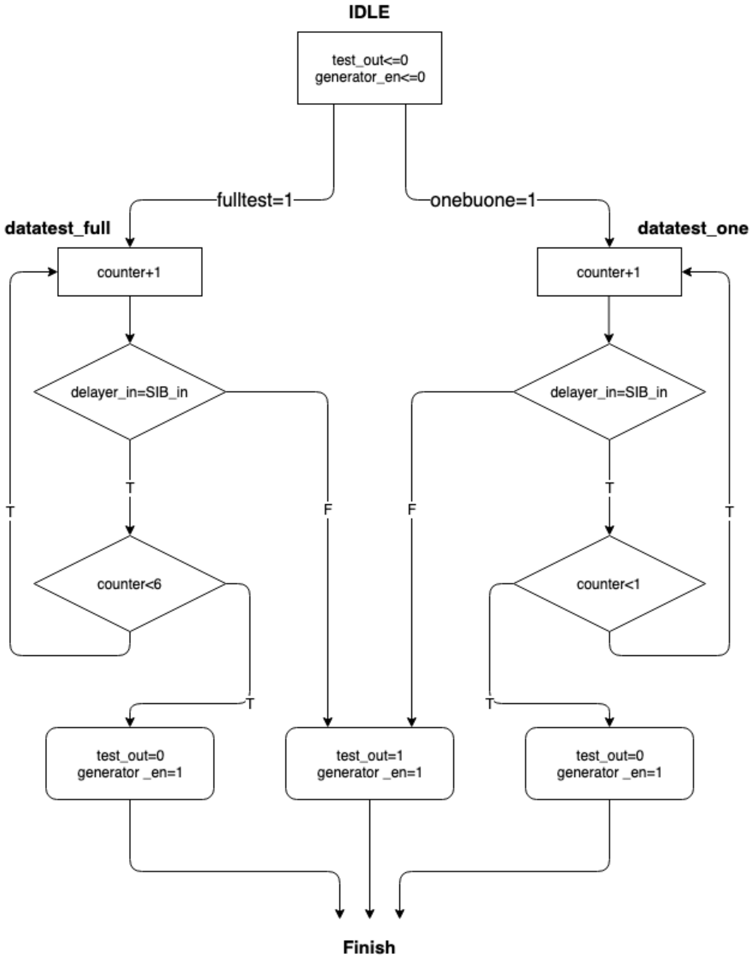


Figure 6.7: ASMD diagram of the FSM for detector.

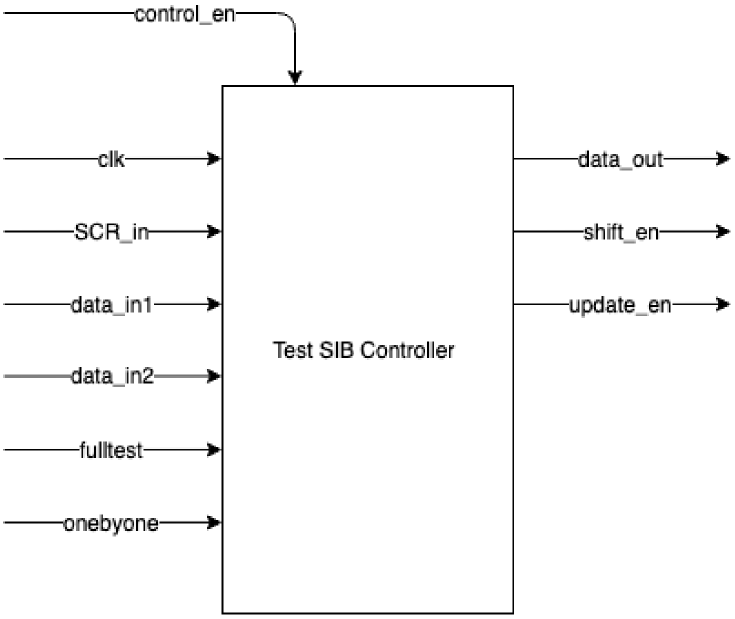


Figure 6.8: SU controller.

state and dummy bits_one state respectively.

6.4 Algorithm for software solution

Algorithm 3 shows the iTest commands. First, it applies the ALLWrite command to write the test data into all scan registers one by one. Then the ALLRead command will be applied to send the dummy bits to the scan network so that all the test data can be pushed out from the scan network back to the test controller. After that, the iTest function will look for the same data as incoming data in the original data buffer. If there is same data, the corresponding scan register is not faulty. If not, the corresponding scan register has a fault and the address of the scan register will be stored in the FaultScanRegisters file.

```
Algorithm 3 iTest

    AllWrite()
    AllRead()

    counter= dataBits / 8
    for i in range(len(original_data)):
      if re.search(incoming_data[i], original_data):
       print(f'Number{len(expected_data)-1-i}is match')
       counter -= 1
       original_data = re.sub(key[i], '', original_data, 1)
```

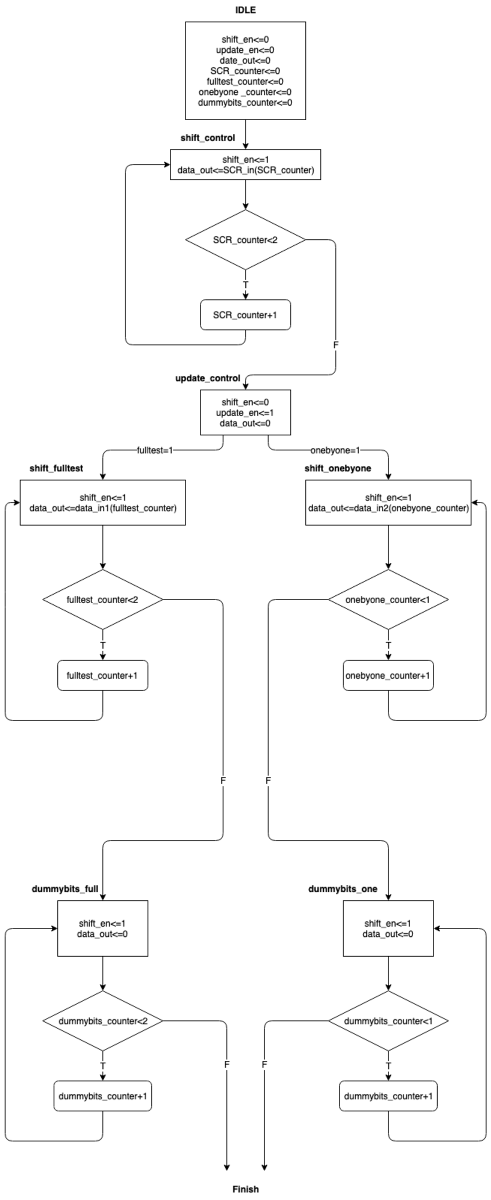


Figure 6.9: ASMD diagram of the FSM for SU controller.

```
      else:
       print('Number{len(original_data)-1-i}is not match')
       FaultSIBs.append(len(original_data)-1-i)
    if counter == 0:
    f = open('FaultScanRegisters.txt', 'w')
    print (f.name)
    f.writelines(str(FaultSIBs))
    f.close()
```

Algorithm 4 is the ALLWrite function, the idea of it is to line up the setup and action command of all scan registers, then send these data to the device under test. The ALLRead function (algorithm 5) has the same idea except it will save the incoming data to a buffer, which will be used in iTest command.

```
Algorithm 4 ALLWrite function

def AllWrite():
    for i in range(TOTAL_ScanRegisters - 1, -1, -1):
        send_cmd.append(iWrite(i))
    send_cmd.append(iApply(iWrite))
    ser.write(b''.join(send_cmd))
    for i in range(len(testvector)):
        ser.write(testvector[i])
```

```
Algorithm 5 ALLRead function

def AllRead():
    for i in range(TOTAL_ScanRegisters - 1, -1, -1):
        send_cmd.append(iRead(i))
    send_cmd.append(iApply(iRead))
    ser.write(b''.join(send_cmd))
    for i in range(len(dummybits)):
    ser.write(dummybits[i])
    incoming_data = ser.read(BUFFER_SIZE)
```

Algorithm 6 is the iRepair command. The idea of the repair function is that, when the test controller wants to access the faulty scan registers, the iRepair command will check the FaultScanRegisters file. If the address in the file is the same, then bypass all the command which related to the faulty scan registers. Figure 3.13 shows this process, in this example, the commands related to the second scan register is bypassed, and the test controller only send the commands which is related to the normal scan registers

```
Algorithm 6 iRepair

  f = open('FaultScanRegisters.txt', 'r')
  FaultScanRegisters_location = f.read()
```

```
    f.close()
    for i in range(TOTAL_SIBS - 1, -1, -1):
        if re.search(str(i), str(FaultScanRegisters_location)):
            print(f'The number{i} ScanRegister has fault')
        else:
            send_cmd.append(iRead(i))
            send_cmd.append(iApply('r'))
    ser.write(b''.join(send_cmd))
```

References

[1] Erik Larsson and Farrokh Ghani Zadegan, *Accessing embedded DfT instruments with IEEE P1687*, In: 2012 IEEE 21st Asian Test Symposium. IEEE. 2012, pp.71–76.

[2] Wikipedia, *Universal asynchronous receiver-transmitter*, `www.en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter/`

[3] Prathamesh Murali and Gani Kumisbek, *Reconfigurable Instrument Access Network with a Functional Port Interface*, In: 2019 International Test Conference.

[4] Erik Larsson, Matteo Sonza Reorda,Paolo Pasini,Marco Palena,Farrokh Ghani Zadegan and Riccardo Cantoro, *Test of Reconfigurable Modules in Scan Networks*, In: 2018 IEEE Transactions on Computers. Page 1-5

[5] Soliton Technologies, *UART PROTOCOL VALIDATION SERVICE*. 2018. `https://www.solitontech.com/uart-protocol-validation-service/`

[6] *IEEE Standard Test Access Port and Boundary Scan Architecture*. In: IEEE Std.1149.1-2001 (2001), pp. 1–212.

[7] Semantic Scholar, *Reconfigurable On-Chip Instrument Access Networks*. 2017. `https://www.solitontech.com/uart-protocol-validation-service/`

[8] Digilent Inc. Nate Eastland. FPGA – *Configurable Logic Block*. 2018. `https://blog.digilentinc.com/fpga-configurable-logic-block/` (visited on 8/17/2019).