Examensarbete

Design of an FPGA Based JTAG Recorder for use in Production of IPTV Set-Top Boxes

Examensarbete utfört i Datorteknik vid Tekniska högskolan i Linköping
av

Robert Andreasson

LiTH-ISY-EX--09/4294--SE
Linköping 2009
Design of an FPGA Based JTAG Recorder for use in Production of IPTV Set-Top Boxes

Examensarbete utfört i Datorteknik vid Tekniska högskolan i Linköping av

Robert Andreasson

LiTH-ISY-EX--09/4294--SE

Handledare: Tomas Franzon
Motorola, Linköping

Examinator: Olle Seger
ISY, Linköpings universitet

Linköping, 25 September, 2009
Design of an FPGA Based JTAG Recorder for use in Production of IPTV Set-Top Boxes

This thesis evaluates the possibility to replace the manufacturer dependent JTAG device used in the production tests of IPTV set-top boxes for storing the boot loader in the main memory in order to start the box for the first time.

An FPGA based prototype was built in order to see if it is possible to record the JTAG signals, to an external DDR SDRAM, without understanding them and be able to perform a delayed playback resulting in the same behavior as with the original JTAG device.

Overall the thesis was successful and it shows that it is in fact feasible to create a JTAG recorder based on an FPGA. A lot of data is used for storing the sequence though so the use of a fast memory is crucial. However in this thesis the speed of both the recording and the delayed playback was reduced in order to work properly.
Abstract

This thesis evaluates the possibility to replace the manufacturer dependent JTAG device used in the production tests of IPTV set-top boxes for storing the boot loader in the main memory in order to start the box for the first time.

An FPGA based prototype was built in order to see if it is possible to record the JTAG signals, to an external DDR SDRAM, without understanding them and be able to perform a delayed playback resulting in the same behavior as with the original JTAG device.

Overall the thesis was successful and it shows that it is in fact feasible to create a JTAG recorder based on an FPGA. A lot of data is used for storing the sequence though so the use of a fast memory is crucial. However in this thesis the speed of both the recording and the delayed playback was reduced in order to work properly.
Acknowledgments

This master thesis has been performed at Motorola, Linköping. I would like to show my gratitude to all working in the hardware group for welcoming me and showing their interest in my work there. Special thanks to my supervisor Tomas Franzon and his co-worker Andreas G. Larsson for all their help considering the production test system.

I would also like to thank my examiner Olle Seger as well as my opponent Erik Lindahl, for helping me improve the quality of this thesis.

Finally I would like to thank my friends and family for supporting me during all this time.

Linköping, October 2009
Robert Andreasson
## Contents

1 Introduction 1
   1.1 Background ................................................. 1
   1.2 Motorola .................................................. 1
   1.3 Purpose .................................................... 2
   1.4 Goal ......................................................... 2
   1.5 Disposition ................................................ 2
   1.6 Definitions ............................................... 3
   1.7 Abbreviations ............................................. 4

2 Prestudy 5
   2.1 JTAG Basics ................................................. 5
   2.2 Present System ............................................. 7
   2.3 Hardware Requirements ................................... 7
   2.4 FPGA Development Board ................................. 8

3 Proof of Concept 11
   3.1 Connections ............................................... 11
   3.2 JTAG Tunnel ............................................... 12
   3.3 Results and Problems .................................... 12
   3.4 Uncertainties ............................................. 13

4 Design Strategy 15
   4.1 Memory Controller ....................................... 16
   4.2 Recording JTAG Signals ................................... 17
   4.3 Delayed Playback ......................................... 17
   4.4 Playback Recorded Signals ............................... 18
   4.5 Serial Interface .......................................... 18
   4.6 System Features ......................................... 18

5 Implementation 21
   5.1 Memory Controller ....................................... 21
   5.2 Recording .................................................. 23
   5.3 Playback ................................................... 24
   5.4 Playback Recorded Signals ............................... 25
   5.5 Serial Interface .......................................... 25
5.6 System Features .............................................. 26

6 Results ......................................................... 29
   6.1 Preparations .............................................. 29
   6.2 Running the System ..................................... 30
   6.3 Relations of ‘a’, ‘c’ and ‘I’ ............................ 32
   6.4 Phase Shifted Playback ................................ 33
   6.5 Dump Recorded Data .................................... 33
   6.6 Reconstruct the Faulty Behavior ..................... 33
   6.7 Clocked Down JTAG ..................................... 34

7 Conclusion ..................................................... 37

8 Future Work .................................................. 39

Bibliography ..................................................... 41
Chapter 1

Introduction

This thesis is about implementing a JTAG recorder, to be used in production of IPTV set-top boxes, to boot the set-top box the first time when the flash is empty. The thesis is introduced in this chapter, where the background, purpose and the disposition is presented. It also includes some definitions and the abbreviations used further on.

1.1 Background

Almost every chipset suitable for todays set-top boxes (STB) have JTAG support which can be used to boot the product when the flash memory of the product is empty. The chipset manufacturer normally develops the JTAG device for the chipset too, or outsources this to a well known JTAG manufacturer. The JTAG standard only covers the electrical and low level communication part. It does not cover any specific parts of an STB chipset. This means that the JTAG device developed by the chipset manufacturer is the only one supporting the STB chipset. The target user for the JTAG device is a software developer aimed to make software development on the STB as easy as developing PC software\(^1\). The JTAG

\(^1\)Live debugging, setting break points etc.
device is normally quite expensive\textsuperscript{2}, difficult to buy in larger quantities and is not very robust to hardware errors on the board.

One solution to the problems above is to create a JTAG recorder to replace the JTAG device with, as Figure 1.1 shows. A device which can record the whole sequence that is needed to boot the chipset from the original JTAG device and then used as a playback device in the factory test stations for the STBs.

1.2 Motorola

The thesis was performed at Motorola in Linköping. The Linköping division, former known as Kreatel, develops hardware and software for IPTV set-top boxes.

Kreatel began operations in 1995 with the development of automated telephone prefix call routers. Eventually the focus was shifted in developing IPTV Set-top Boxes. In 2006 Kreatel was bought by Motorola to assume the role as the European research and development center.

1.3 Purpose

The purpose of this thesis is to evaluate whether it is possible to create a JTAG recorder that can replace the present JTAG box in the test fixtures and which costs markedly less. The JTAG recorder’s job is to boot up the STB the first time after production when the flash memory is empty.

1.4 Goal

The goal is to implement a JTAG recorder in an FPGA to investigate whether it is feasible as a substitute for the JTAG box.

If the experiment succeeds the next step is to create a working product to be integrated in the test system fixtures.

The JTAG recorder shall be constructed in such way that only one JTAG box is needed for recording a JTAG sequence once and then playback the same sequence over and over again.

1.5 Disposition

First a pre-study is done in Chapter 2, where JTAG basics, the present system, hardware requirements and the development board are discussed. In Chapter 3 the concept of running JTAG sessions through an FPGA is tested. The physical connections and some other uncertainties are also described. The design strategy of the JTAG recorder is revealed in Chapter 4 and the actual implementation in Chapter 5. In Chapter 6 the results of testing the JTAG recorder is presented.

\textsuperscript{2}Typically in the region of \$1500 - \$4000.
The conclusion is discussed in Chapter 7 and what to be done in the future for creating a working product in Chapter 8.

1.6 Definitions

**JTAG** IEEE 1149.1-1990 Standard JTAG Specification: "Standard Test Access Port and Boundary Scan Architecture" [3]. This standard is commonly known by the acronym JTAG.

**JTAG box** The device in the test system fixtures in the present system is referred to as the JTAG box.

**Bootloader** A piece of program code used for loading more complex programs such as an operating system to RAM and then start execute it. This is needed especially when the main program is stored on non volatile storage devices, such as hard disk drive, CD, DVD, USB flash drive or floppy disks, since the processor can only execute program code from the ROM or RAM.

**Set-Top Box** is the term for a device that connects and decodes an external source of a signal which is then displayed on a TV.

**STMicroelectronics** the manufacturer of the main processor in the IPTV STB’s from Motorola as well as the JTAG box used in the present test system.

**FPGA** Field-Programmable Gate Array, a customer configurable semiconductor device. It is programmed using a hardware description language such as VHDL or Verilog.

**Xilinx and Altera** The two major FPGA manufacturers.

**ModelSim** Mentor Graphics’ simulator for HDL code.

---

1.7 Abbreviations

ASCII American Standard Code for Information Interchange
CAD Computer Aided Design
CPU Central Processing Unit
DCM Digital Clock Manager
DDR SDRAM Double Data-Rate SDRAM
FIFO First In First Out
FSM Finite State Machine
FTP File Transfer Protocol
GDB GNU Debugger
GND Ground, in electrical circuits
HDL Hardware Description Language
IC Integrated Circuit
IP Core Intellectual Property Core
IPTV Internet Protocol Television
I/O Input/Output
IR Instruction Register
JTAG Joint Test Action Group
LED Light Emitting Diod
MIG Memory Interface Generator
OS Operating System
PCB Printed Circuit Board
RAM Random Access Memory
ROM Read-Only Memory
RS-232 Recommended Standard 232, used for serial data communication
SH4GDB STMicro customized GDB
ST Short for STMicroelectronics
STB Set-Top Box
SDRAM Synchronous Dynamic RAM
TAP Test Access Port
TFTP Trivial FTP
UART Universal Asynchronous Receiver/Transceiver
U-boot Universal boot loader
USB Universal Serial Bus
VCC Common Collector Voltage
VHDL VHSIC HDL
VHSIC Very-High-Speed IC
Chapter 2

Prestudy

This chapter clarifies how the old system works and what is required by the new one. Also decisions of what development equipment to use are taken in this part of the thesis.

2.1 JTAG Basics

JTAG\textsuperscript{1} is mainly used for debugging integrated circuits but could also be used for loading firmware into a system.

The JTAG standard consists of the test access port (TAP) and the boundary-scan architecture. The boundary-scan architecture deals with how to detect and communicate with multiple devices in a chain and is not so relevant for this thesis.

The TAP specifies the physical connection and the signals as well as the control unit. The TAP consists of five signals listed in Table 2.1 - one for control, two for data, one reset and the clock signal.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS</td>
<td>Test Mode Select</td>
</tr>
<tr>
<td>TCK</td>
<td>Test Clock</td>
</tr>
<tr>
<td>TDI</td>
<td>Test Data In</td>
</tr>
<tr>
<td>TDO</td>
<td>Test Data Out</td>
</tr>
<tr>
<td>TRST</td>
<td>Test Reset</td>
</tr>
</tbody>
</table>

Table 2.1. List of the TAP signals.

The TAP controller is a synchronous finite state machine that responds to changes of the TMS and TCK signals and controls the operations of the circuitry. The state machine, shown in Figure 2.1, has two branches where one is for accessing the data register (DR) and the other is for the instruction register (IR). The TMS signal is sampled upon the rising edge of TCK and its value decides the next state of the controller.

\textsuperscript{1}The whole section is based on the JTAG standard [3].
A typical JTAG operation involves first entering the branch for loading an instruction and then proceeds with the other branch for data. When a JTAG register is connected to the TAP it acts like a shift register with TDI on the input side and TDO on the output.

JTAG operations are instruction based and the procedures involve first selecting the instruction register to connect to the TAP. It acts as a shift register and is connected with TDI for the input and TDO for the output. Next step is to shift the register and it receives a new value while the former one is read out.

There is a set of instructions that must be provided to follow the standard; \textit{BYPASS, SAMPLE/PRELOAD} and \textit{EXTEST} as well as two optional ones, \textit{IDCODE} and \textit{USERCODE}. The \textit{IDCODE} instruction is of special interest in this thesis since it provides the device ID code.

\begin{center}
\textbf{Figure 2.1.} The TAP Controller state diagram.
\end{center}

In Section 5.4 an example for extracting the device ID code is explained as a part of testing the recorder and the playback unit.
2.2 Present System

When the set-top box has passed through the production line it is placed in a test station. The test station contains a little box from ST which is connected to the STB through the TAP using the JTAG standard. In the other end the box is connected to a PC using ethernet or RS-232. To control the box a modified version of Gnu debugger - SH4GDB, provided by ST - is used. This way all registers can easily be accessed in real time.

In the production all of the functions provided by this JTAG box are not really necessary. The procedure is basically exactly the same over and over again once for every STB leaving the assembly line. The system memory is loaded with the bootloader and the STB is commanded to boot.

The new system is expected to record the signals from the JTAG box, store them in the memory and finally be able to playback the signal flow and the behavior should be the same as with the old system - that is, the flash memory is loaded and the STB boots up.

Different models have different bootloaders so it is also expected that the new system should be able to record more than one session. It is also preferred that the system could record from other JTAG boxes from different manufacturers.

2.3 Hardware Requirements

The JTAG standard has four signals, except the JTAG-clock, that are of interest to record. In order to estimate how much memory required to record a whole session, the time taken to flash the system was measured. Since the speed was known it was easy to calculate the bitrate,

\[
\text{bitrate} = f_{\text{JTAG}} \cdot n_{\text{signals}} = 40 \text{ Mb/s} = 5 \text{ MB/s}.
\]

The initialization script was modified so that both the setup and loading was performed after one another. In both the beginning and the end of the script checkpoints were set to measure the system time, and a fairly good estimate of the time needed was achieved. Time measured was roughly 3200 milliseconds, and with the given bitrate it gave a rough estimate of

\[
3.2 \text{ s} \cdot 5 \text{ MB/s} = 16 \text{ MB},
\]

of memory required by the hardware.

In order to get more reliable estimates an oscilloscope was used to see how long the data stream really was. The script was executed again and the time from the first activity on TDI to the last was measured. The measurement result is presented in Table 2.2.

The measured time was approximately one third shorter - only taking about 2 seconds, ending up with a need of

\[
2 \text{ s} \cdot 5 \text{ MB/s} = 10 \text{ MB}
\]
Table 2.2. Measured time for JTAG initialization and boot loader script using an oscilloscope. Subsets are marked in italic.

memory in total.

It was also observed that during the delayed reset phase of the initiation of the chip there were no JTAG activity at all, not even the JTAG clock was ticking. It should not be to much effort to optimize down this part, reducing the total time needed for the recording to 1.5 seconds, ending up with a memory need of as little as

\[
1.5 \text{ s} \cdot 5 \text{ MB/s} = 7.5 \text{ MB}.
\]

These memory consumption calculations are only rough estimates of what one can expect. The script did not always finish at a fixed time, it was observed that it varied most of the times between 1.5 and 2.0 s. To avoid ending up in a situation where the amount of memory is not enough, the minimum requirement was set to 16 MB.

Another thing to have for consideration is how fast the system could run. The JTAG box from ST runs the signals in 10 MHz. For recording one sample per period that is the minimum frequency. It is however uncertain whether the the signals have some internal skews that are of importance. To be able to record more than one sample per period requires at least twice the speed, and it may be favorable to have more samples to choose from. The aim is to have a system speed of ten times faster than the JTAG speeed - that is 100 MHz.

An issue that is not neglectable is the set of I/O ports. The JTAG TAP consists of five signals, thus five inputs and five outputs is the minimum requirement. It is also favorable that standard components for the connectors can be used.

Other things in the wish list is RS-232, ethernet and USB interfaces.

2.4 FPGA Development Board

At this point it was clear that the main issue was to find a development board with enough memory. The FPGA itself was of less importance, since there are very few that even have 1 MB of internal dedicated RAM. Since this is an evaluation project it is a good idea to choose a development board with at least 16 MB of RAM.

There are mainly two big FPGA manufacturers, Altera and Xilinx, and the suggestion was to choose from one of these manufacturers. Motorola in Linköping does not have any real development involving FPGAs, hence the matter was not of big importance. However a video quality measurement box was recently developed
through a series of previous master theses at Motorola. That project was based on a Xilinx Spartan-3E FPGA, so there may be an advantage going for Xilinx.

With the above mentioned prerequisites and the intent to find a development board with reasonable price, it was down to mainly two candidates - DE2-70 [1] from Altera and Spartan-3E Starter Kit [8] from Xilinx. After some digging in the recesses, the development board from the previous theses was found. It was in fact one of the candidates, so in practice there was only one choice.

The final choice was a Xilinx Spartan-3E Starter Kit including a XC3S500E FPGA, 64 MB DDR SDRAM, lots of I/O ports and all the other things on the wishlist. It has a 50 MHz onboard oscillator and also an SMA port which could be used to connect an external clock. In Figure 2.2 the peripheral interfaces that are intended for use are illustrated.

![Spartan-3E Starter Kit](image)

**Figure 2.2.** A simplified block diagram of the development board.

---

2 More specific details in the datasheet.
Chapter 3

Proof of Concept

This part of the project was necessary to do in order to be ensured that the JTAG recorder in fact is possible to build. The basic idea was to build a simple tunnel to route the JTAG signals of interest through the FPGA. If this was successful it would most likely be possible to build the JTAG recorder.

3.1 Connections

The JTAG box is connected to the STB via a 2x10-pin header which the FPGA development board did not have. The most suited option was to use the 6-pin headers\(^1\). The board has three such connectors with 4 I/O pins each as well as GND and VCC [8]. There are five signals of interest, thus five inputs and five outputs need to be connected through the FPGA. To solve the problem with different connectors, an adapter card is in need for fabrication, which branches the signals for both the FPGA and the STB.

![Figure 3.1. The simplified layout of the adapter card.](image)

The adapter card is, as Figure 3.1 shows, quite simple with two 2x10-pin header

---

\(^1\) J1, J2 and J4 connectors, see Figure 2.2
sockets to connect the JTAG and the STB; and two 90° 6-pin headers to connect the J1 and J2 connectors. To be able to connect the signals from the adapter card to J4 a 6-pin female header was attached to the board via the signal cables. The adapter card routes the TMS, TCK, TDI and TRST from the first 2x10-pin header from the JTAG box to the input of the FPGA and then from the outputs of the FPGA to the STB. The TDO signal needed to be directly connected between the STB and the JTAG box, hence that signal was branched to both the FPGA and the JTAG box on the adapter card. The connectors were soldered on a prototype board, and each of the signal cables were pair twisted with ground cables in order to reduce noise.

Two double pole double throw slider switches were also mounted on the adapter card acting as FPGA bypass for the JTAG signals. This might come in handy when testing and comparing with and without the FPGA connected.

3.2 JTAG Tunnel

There were basically two tasks to perform in order to proof that the JTAG recorder may be possible to build. They were to ensure that the recording of the JTAG signals was possible, and also ensure that the playback would work.

To see if it is possible to record the signals some functionality was built to trigger on TCK. It consisted of an 8-bit shift register driving the LEDs on the FPGA board. The trigger was simply an edge detector of TCK on positive edge of the system clock. To actually be able to observe the shifts a down scaling mechanism was also implemented. The expected result should show a shift of the active LED in a rate of approximately five times per second.

The playback part should be enough proof to just tunnel the signals through the FPGA and the expected behavior would be that the JTAG communication proceed as normal.

3.3 Results and Problems

The main issue to be solved was to set up a proper tunnel and get a working JTAG communication. The VHDL code for this is really simple, and the task was mainly to get the pin-out correct. This was well specified in the Spartan 3E Starter Kit User Guide [8] and there were also notes printed on the board. This part turned out to work perfectly fine.

Next task was to get a working physical connection. The adapter card was built and to test it, jumpers were connected to test that the signals were routed the right way. The JTAG communication worked although it was observed, using an oscilloscope, that the signals were quite noisy. This was reduced by pair twisting the signal cables with the ground cables. The result was still a bit noisy but it was acceptable. By using even shorter cables or preferably CAD a real PCB could probably result in reduced additional noise.

The final test was to connect the adapter card to the FPGA, the JTAG box and the STB which turned out to work as expected. The JTAG communication
worked which resulted in that the LEDs shifted as expected. When the JTAG communication was terminated the LEDs ceased shifting. The oscilloscope showed that the signals had proper levels also when disconnected.

These results should be enough to conclude that it is possible to build the JTAG recorder.

### 3.4 Uncertainties

During the measurements in Section 2.3 it was observed that the length of the JTAG script varied between 1.5 - 2.0 seconds. This could indicate that the JTAG box does some kind of hand shaking with the STB when trying to access its registers. The JTAG box has its own operating system\(^2\) though so the differences might very well originate from the interrupts caused by the OS.

Another thing observed during the measurements was that the TDI signal occasionally got its value changed slightly after the rising edge of TCK. The JTAG standard clearly states that TDI is sampled during the rising edge of TCK so this might not be a big issue. If this really matters it should just be a timing issue which should not be that hard adjusting for. It seems though that these changes only occur in states where TDI is not sampled.

The first 500 milliseconds of the script is an initialization phase. There is a special JTAG micro controller integrated in the ST-processor of the STB. The JTAG module is set up to communicate with this micro controller by default and the initialization script must be executed in order to be able to communicate with the main processor. The initiation is performed during system reset. In order for the reset circuitry to be completed the JTAG signals freezes which includes also the TCK. This could complicate recording, for instance if the sampling is triggered for TCK transitions. It was later discovered that the reset delay was not completely needed. There is however an inevitable period of about 20 milliseconds during the JTAG reset where the TCK is silent most of the time.

The JTAG box seems to be very concerned to retain the connection with the STB. If the connection gets disrupted and then restored again it seems that the JTAG box and the STB processor gets out of sync. After such an operation trying to execute some JTAG command, SH4GDB says that it was unable to communicate with the device. This should not be a big issue for the full boot loading script since the initialization part should be included within the recording. Although it may be some complications trying to record and playback smaller JTAG scripts and a workaround to this must be implemented.

\(^2\)ST’s own operating system called OS20.
Chapter 4

Design Strategy

This part is where the actual JTAG Recorder begins to take shape. The intended design, shown in Figure 4.1, needs a memory controller for the on-board DDR SDRAM of the Spartan 3E Starter Kit in order to store the recording of the JTAG initialization and bootloading scripts. Such a controller is quite complicated and not really in the scope of this thesis, a third party controller is needed. It needs also mechanisms for the recording and delayed playback part. Finally it needs a communication module in order to be able to control the system from a PC. A third party UART is going to be used for that.

Figure 4.1. An overview of the system design.
4.1 Memory Controller

The DDR SDRAM controller to be used is an IP core from Xilinx Core Generator [7] called MIG - Memory Interface Generator [9]. The generated memory controller itself was in practice not usable. Much effort was left to do in order to get a usable interface for memory read and writes.

The main things the controller handles are timing, refreshing and address translation. The DDR SDRAM is divided into four banks with rows and columns. In order to save pins, the row and column addresses shares the same pins [5].

The communication with the memory is command based, and likewise is the memory controller. The only commands that were needed were memory initialization, write and read. The memory initialization is only needed once, and the write and read commands are very similar. Based on these facts, the interface turned out to be fairly simple to build. The MIG userguide [9] includes timing diagram with associated description for both writing to and reading from the DDR Memory. They should provide the information needed in order to proceed with the design of the interface.

The user interface shall include one address register that is used both for reading and writing, one register each for input and output data buffers and two control bits for initiating communication with the memory for the controller to know whether it shall perform a read or write.

The interface module is basically going to be an FSM. Initialization shall be done upon system reset, and when the memory controller is initialized, it goes into a ready state waiting for a command. When a command is asserted the state shall change to another wait state since the first address in a burst must be active for three clock cycles after the controller acknowledges the command. Normally the following addresses shall be asserted once every clock cycle. Since JTAG communication only operates at 10 MHz, the throughput is quite low. Hence no long bursts are needed, however the controller does not allow less than four word transactions per initiated burst. Using burst length of two, implicates that two bursts are needed. After the two bursts the burst_done\(^1\) signal is asserted and the state machine shall now wait for the command_ack\(^2\) to be deasserted to finally go back to the ready state. A write or read command is roughly going to be asserted once every 160 clock cycle in this system.

The write and read commands works similarly and they can share the basic functionality. The only difference is that when a write command is asserted the input data must be asserted at latest one clock cycle after command_ack, and the following words one clock cycle each later. Actually it must be asserted on the positive edge of a 90 degree phase shifted clock. DDR Memories work at both edges of the clock. The phase shifted clock is needed in order for the control sequence to be ready before asserting data.

The data read is also clocked with the 90 degree phase shifted clock. When the controller asserts the data valid signal the read data is stored in a register on both edges of the clock signal.

\(^1\)Shall be provided to the MIG, see Figure 5.1
\(^2\)Is provided by the MIG, see Figure 5.1
4.2 Recording JTAG Signals

The basic concept of this module is to record the whole JTAG session to the RAM. The first problem is to build a mechanism to sample the signals at the rate of TCK. The module must know whether it shall sample signals or not.

Since the controller stores four words upon a write command and a sample of JTAG signals is quite shorter than one single word, it is preferable to sample the signals into a buffer of four words first and then pass it to the memory controller. The module shall also wait for the controller to be ready before it tries to store new data. This shall not be a problem since the system clock is faster than the sample rate as well as there are many samples between each store. It is probably wise to use a store buffer just in case.

The JTAG signals must be synchronized to ensure that they are sampled with correct values. In order to ensure that only valid data is stored during recording, the system sample signals once every clock cycle into a 10-bit buffer implemented as a shift register. Upon detection of a rising edge of TCK one of the sampled bits for each JTAG signal is stored to the memory. If no edge is found, no values gets stored to the memory. This may be implemented later on to store the last sample value within each TCK period since the last known transition.

When the JTAG session is done it may very well be the case that there are unstored samples in the buffer. Therefore a mechanism to store the buffer properly in the memory is needed. It is also desired to indicate with a flag when the recording session is done.

4.3 Delayed Playback

Basically this module shall just read the recorded data from the memory and send it to the STB at a rate equal to the normal JTAG clock. Hence the first thing to do is to generate an authentic TCK signal. Maybe a mechanism for the generated TCK to be silent during the reset sequence is needed. It can also be preferable to have the ability to choose a custom frequency of the TCK.

The playback shall work much like the recorder but in reverse. A playback buffer and an output buffer from the memory controller is therefore desired. To prevent unnecessary timeouts a read from memory shall be performed as soon as the output buffer is passed to the playback buffer.

The playback module shall have a mechanism to start the actual playback. It shall also know how much data is recorded so that only valid JTAG signals are played back. Since the data is stored in the memory as a FIFO buffer the last record address shall show where the last signals are stored.

The basic concept is to just playback the JTAG signals as stored, but maybe this will not be enough. If this shall be the case the module probably must listen to the TDO to get feedback from the STB in order to playback the signals when it is ready. In this case a larger playback buffer may be needed to have enough data buffered to be able to send the right data at the right moment. There is a slight possibility that delayed playback is impossible if two-way communication
Design Strategy

is required by the JTAG box, for example if it runs an encrypted handshaking protocol or something like that.

4.4 Playback Recorded Signals

At this stage if everything works like it is presumed to do, a JTAG session shall be recordable and played back. The result shall be that after the playback, the same things shall happen as it did during the recording phase. The first test shall be a fairly simple one. According to the JTAG standard [3] some predefined registers must be included, such as a register named IDCODE which contain an identification code for the device. The first test shall be to access this register, record the process and then playback. If this works it is time to take it up a notch and try recording and eventually playing back a real JTAG session that is used in the production test system.

4.5 Serial Interface

To be able to control the system a UART is decided to be implemented. It shall run over the serial port and use a baudrate of 115,200 bits per second with 8 data bits and 1 stop bit. There are two data wires connected to a UART - one used for receiving and the other for transmitting. Both receiving and transmitting can be performed simultaneously and asynchronous, as the name UART reveal.

The main purpose with this unit is to control the system - as in start of a memory test; activate the recorder; or start a playback session. It shall also be able to provide information such as end address of a recording or even all the data from a recording and so on.

4.6 System Features

The implemented system is intended to evaluate whether the old JTAG box can be replaced in the production line test fixtures. The complete solution is not given from the beginning and the design may be changed many times during the evaluation period. To minimize the number of synthesizes a broad variety of functions are implemented from the beginning so that many different methods can be tested back and forth with one single synthesize.

Memory test To ensure that the memory still works during testing the memory test shall be easy to invoke.

Initiate recorder Set the recorder into the ready state waiting for start condition and which predefined recording method to use. These different methods can be such things as what to trigger on, when to store a sample and how many samples to be stored each TCK period.
4.6 System Features

Playback    Performing playback of recorded data shall be very easy. It shall also be possible to run the playback several times for one recorded session. It may be of interest to change the playback speed also.

Change the phase of the delayed playback    Since it has been observed that there may be timing issues concerning the TDI signal predominately there may be an advantage to be able to change the phase of the signals relative the TCK signal.

Dump recording    Being able to transfer the playback data to the PC has several advantages. For one thing the playback data can be examined and compared with other played back sessions. In the long run the played back sessions can be saved on another machine and reloaded back into the memory of the JTAG recorder for new playbacks.

Size of recording    Receiving the number of memory writes can be very useful to at least know how much of the memory is really needed.
Chapter 5

Implementation

This section describes the actual implementation and what problems occurred during development. It also describes how the design was verified.

5.1 Memory Controller

![Diagram of Memory Controller and Interface]

Figure 5.1. The Memory Controller and Interface.

The DDR SDRAM needs a clock frequency in the region of 77 MHz to 167 Mhz. The on board oscillator only operates at 50 MHz, so a DCM - Digital Clock
Manager was implemented to double the frequency of the system clock [10]. To guarantee that the new clock is stable and does not alternate the phase, it should be fed back to the DCM via a buffer.

The MIG uses a DCM of the system clock to have two additional skewed clocks by 90 and 180 degrees respectively. At the input of the controller the system clock goes through another buffer before it enters the DCM. This is not allowed when cascading DCMs so the input buffer had to be bypassed.

As stated in Section 4.1 the MIG controller did not have any user friendly interface as one could have wished. Huge effort was spent on implementing the memory controller as well as the design and implementation of the interface. The documentation was vague and somewhat inadequate for some parts but overall it was still useful, including the basic facts needed for the memory interface [9].

The interface handles the read and write operations as well as the initiation of the memory which is performed during system reset. In Figure 5.1 the integration of the memory controller and the interface is shown. The interface consists of an address register and two data buffers, accessible via the signals wr_address, wr_input for write and wr_output for read. Two control signals are used for initiating read or write and there is also a status bit for indicating whether the controller is ready or not. To perform a write to the memory the address register and data input buffer must be loaded first, then the control signal wr_doit must be asserted for one clock cycle. This can only be done as long as the status signal wr-ready is set. To perform a memory read, only the address must be loaded and both wr_doit and wr_read must be asserted for one clock cycle. This could also be done only when wr_ready is set. When performing a memory operation the wr_ready signal is cleared until it is done, then it is set back to one again and if a read was performed the result is stored in the output buffer.

At first no memory simulation model for VHDL was available. There were only verilog models available and since Xilinx evaluation version of ModelSim did not support mixed mode simulation it was quit hard to verify that the controller and interface worked as expected. A simple hardware memory test was implemented in which the test data written to the memory is fixed to ones. Subsequently the data is read back and since the data is fixed it is easy to check if the correct data is read back. A counter was then implemented to keep track of faults. Three other tests were also implemented. One where the address is written into the memory at the very same address, one where the address is inverted and one where zeros is used as fixed test data. The variations of the test patterns were used mainly in order to discover stuck-at one or zero faults.

This was not really waterproof except for checking that the hardware worked. When the memory test failed it was hard to really know if it was the memory chip or the controller that failed. It was essential to be able to simulate the whole system for that purpose. Luckily, it turned out that Motorola has a license to ModelSim PE which supports mixed mode simulation. Since there was an available Verilog simulation model [4] for exactly the same memory chip as mounted on the development board, it was now trivial to verify and troubleshoot the system.

At first the interface to the controller was built to perform single bursts of length two, since it seemed unnecessary to store more than that each time. The
memory test failed almost every time in this mode. After consulting the docu-
m entation it turned out that there was a small notice that this was an unsupported
mode of the MIG controller [9]. After the extension of double bursts the simula-
tion verified that the controller worked and the final verification was to program
the FPGA and perform the hardware test through the whole address space. The
result was as expected - a working memory interface and controller with no errors
encountered as well as a working memory chip.

5.2 Recording

The recorder is based on a control unit that works as a simple state machine. The
main features are to listen for start condition, record which also includes the stop
condition, and finalize. The start and stop condition can quite easily be modified
but requires that the system is synthesized again.

The actual recorder part is not complicated at all. It has a sampling unit and
a unit for storing data into the DDR RAM.

The sample unit simply sample the values of the TAP signals each clock cycle.
The sample frequency is 100 MHz and since the JTAG operates in 10 MHz there
are 10 samples each clock period. Each signal of the TAP has a 10 bit wide sample
buffer.

The store unit does not necessarily need to store all samples into the memory.
The control signal mux_ctrl chooses which sample to be stored and a trigger tells
the store unit when to keep values in the store buffer. The trigger could also
be customized in what ever way desired. This also requires re-synthesize of the
project though. Since the memory controller has an input buffer of 16 bits for
each signal that is also the depth of the store buffer. A counter is implemented to
keep track of the amount of buffered samples and after 16 stored samples a control
signal indicates that it is time to store the buffer to the memory. The sample
mechanism is illustrated in Figure 5.2

![Figure 5.2. The basic concept of the sample mechanism.](image)

As stated in Section 2.3 there are four JTAG signals of interest to record. The
memory bus is 16 bits wide and the memory controller stores four words per
initiated write - that is 64 bits. The JTAG operates typically with a frequency of
10 MHz and the FPGA with a frequency of 100 MHz. The number of clock cycles between every write command can now easily be calculated,

\[ f_{\text{FPGA}} = 100 \text{ MHz} \]
\[ f_{\text{JTAG}} = 10 \text{ MHz} \]
\[ n_{\text{signals}} = 4 \]
\[ l_{\text{buffer}} = 64 \text{ bits} \]
\[ N_{\text{cycle}} = \frac{f_{\text{FPGA}}}{f_{\text{JTAG}}} \cdot \frac{l_{\text{buffer}}}{n_{\text{signals}}} = 10 \cdot 16 = 160. \]

That is when recording one sample per TCK period the memory operations have 160 clock cycles to finish. Running the system in ModelSim it is derived that performing memory write operation needs 23 clock cycles most of the times and 32 clock cycles when combined with refresh. Hence there are plenty of time for the recorder to store the buffer to the memory. The memory read is a bit faster and consumes only 20 clock cycles each time and it performs no refreshes.

The four JTAG signals are represented as a nibble and the record buffer takes 16 nibbles. The recording module’s control unit is based on two counters. One for scaling down to 10 MHz sample rate, and one for storing the nibbles in the recording buffer.

When the buffer is full a flag is asserted for performing a write command to the memory controller and it is stored in the input buffer of the memory controller. This way a new nibble may be sampled even if the memory controller is busy. At this relatively slow rate such conflicts may probably not occur though.

The JTAG signals are passed through two D flip flops each in order to synchronize and avoid meta stability. To prevent skew effects the TCK signal is connected to a rising edge detector circuitry which sends a pulse in order to trig the recorder to sample the signals. According to the JTAG standard the signals are clocked in at rising edge of TCK [3].

Verification of this module was made in ModelSim. A JTAG stub was added to the testbench which generates five different clock signals - one for each signal - in order to conveniently get a nice test pattern. TCK gets a frequency of 10 MHz naturally, and since TCK is not stored, an additional signal gets the same period time. The next signal get its period time doubled and so on.

5.3 Playback

This module requires that there is a recorded session in the memory. If there is no recorded session, nothing much will happen while trying to run this module since the end and the start addresses have the same value.

The playback unit is a quite complex module. Timing is a very critical issue since it is of very high importance that the output looks exactly the same as the signals did during the recording. The output must be stable during the whole playback, as well as be free from glitches and other interference. This module is also responsible for generating an adequate TCK signal. The playback frequency
must be able to be varied and the signal flow must be properly correct relative to TCK.

It is important to ensure that there always are valid signals in the output buffer in order to be free from glitches. A solution to this problem is for the playback unit to have an additional buffer so that it can perform a memory read at the same time as the output buffer is updated. During initiation of a playback session two memory reads are performed. The second one is made in order to get the first four words\(^1\) in the output buffer and the remaining set of four words are available in the read buffer. When the output buffer is empty it gets four fresh words instantaneously from the read buffer. A new memory read command is also asserted. This will keep repeating until the address register has the same value as the register that stored the last record address.

The verification of this module was also made in ModelSim, based on the recording results. The main issues to check is weather the generated TCK is operating with the correct frequency. Next step naturally is to check if the recorded signals are output correctly. Both signal values and output frequency are of interest.

### 5.4 Playback Recorded Signals

Section 2.1 described some registers specified by the JTAG standard. The device ID Code of the STB’s main processor was known, hence accessing this register is a good way of testing the recorder. The easiest way to get the value in the IDCODE register is to generate the signals for the JTAG sequence in the FPGA. By following the TAP Controller, showed in Figure 2.1, it was trivial to generate the proper sequences for the TMS and TDI signals.

The first thing is to select the Capture-IR/Shift-IR state and provide the correct instruction code. Then it is time to simply switch state to Capture-DR/Shift-DR and after 32 cycles in that stage the IDCODE register should be visible on the TDO signal.

Since only 50 clock cycles is roughly needed the test pattern for TMS and TDI was stored in one register each. The two registers were shifted out at the speed of the generated TCK signal connected to the JTAG tunnel right after the input from the JTAG box. This way the recorder could be activated and it interpreted them the same way as a normal JTAG session. Finally the recorded data was played back and fortunately both the generated process and the playback ended with the same result - the device IDCODE was returned on the TDO line.

### 5.5 Serial Interface

To save time and to avoid reinventing the wheel a UART from OpenCores.com\(^2\) called "Minimal UART core" was chosen for this purpose \[2\]. It included a baud

---

\(^1\)In this case a word consists of 4 bits, one for each JTAG signal.

\(^2\)A collection of open IP cores, similar to open source for software.
rate generator and it was easy to achieve the desired 115,200 bps. It was built for 8 data bits and 1 stop bit so it fulfilled the requirements with ease.

The transmitter has a simple interface with one send buffer and one wire for asserting the transmitter. It also includes two status signals that indicates if the transmitter is ready and when the stop bit is transmitted.

The receiver works similarly, except for the absence of the control wire. When the stop bit is received the controller asserts an end of capture signal indicating that the receiver buffer is updated.

To check if the UART core works the first step was to implement a second UART in the test bench that could send serial data to the system. The receiver and transmitter was connected as a loop through circuit and when a character was received it immediately started to transmit the same data back. This implementation made it easy to test the UART even when running the design on the FPGA. By using a hyper terminal the letters entered would show up on the screen after pressing the keys.

5.6 System Features

The UART core works as the foundation of the control unit for the whole system. The commands are executed from a terminal on the PC and is connected via the RS-232 interface. The user gets feedback back on the screen when the command is done executing.

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>r n</td>
<td>Initiate the recorder with record method #n.</td>
</tr>
<tr>
<td>p</td>
<td>Playback a recorded session.</td>
</tr>
<tr>
<td>d</td>
<td>Dump the recorded session on the serial port.</td>
</tr>
<tr>
<td>e</td>
<td>Dump the end address of a recorded session.</td>
</tr>
<tr>
<td>m</td>
<td>Perform a standard memory test.</td>
</tr>
<tr>
<td>f</td>
<td>Perform a full memory test.</td>
</tr>
<tr>
<td>C n</td>
<td>Set divider for custom speed.</td>
</tr>
<tr>
<td>H n</td>
<td>Set delay for TDI.</td>
</tr>
<tr>
<td>L n</td>
<td>Set delay for TMS.</td>
</tr>
<tr>
<td>X n</td>
<td>Set n extra samples to record per Ttck.</td>
</tr>
<tr>
<td>R</td>
<td>Show record method.</td>
</tr>
</tbody>
</table>

Table 5.1. System command list

The commands are in this stage only one character long where some commands has an additional argument. For instance when initiating the recorder the record-command needs an argument to know which recording type is going to be used. Eg. "r1" is needed to be typed in order to initiate the recorder with recording method #1. In this case the "r" is returned at once since the record setup is executed and the "1" is transmitted upon end of the recorded session. A list of the available commands are presented in Table 5.1. As an assurance all characters
that does not represent a command are looped through anyway in order to check that the system is still running.
Chapter 6

Results

This chapter covers the testing of the implemented system. That includes all the preparations for running JTAG scripts, recording them with different methods and also the playback with variations.

6.1 Preparations

As stated in Section 2.3 the boot loading script can have a duration of about 2 seconds. A lot of register setups are made during this period of time plus the transferring of the boot loader. On top of that there is a period of about 500 milliseconds where TCK and the other JTAG signals does not change. If something does not work as planned there is a considerable risk that the cause for the failure does not get discovered.

The script has to be broken down in smaller pieces so that possible fault sources could easier be isolated. The script can only be divided into two parts. The initiation part and the boot loading part. It does not have any checkpoints during the load so there is no feedback if anything during the way passes or fails. A script where the STB sends some feedback when the playback succeeds is of high importance.

Since the JTAG box can access all the registers in the STB processor and the STB has a serial interface for maintenance purposes the smallest JTAG script possible would be to simply load the STB processor’s UART send buffer and let it send a character on the serial port. This script is called print_a since it loads the UART send buffer with the letter ‘a’. The send buffer is actually a 16 byte wide FIFO buffer and to get a little longer but not much more complex script, a print_16 script was also created. This script loads the FIFO buffer with the hex-table: ‘0123456789abcdef’.

To be able to execute the scripts, the JTAG communication must be initialized in order for the JTAG box to have direct access to the main processor of the STB. The UART of the processor must also be set up and that is done in another script called print_uart. This script enables the UART and it also loads the FIFO buffer
Another problem is when switching to playback mode, the TCK signal interrupts and gets a shifted phase when switching to the generated TCK. The problem seems to be that the JTAG state machines of the JTAG box and the STB gets out of sync. A solution to this problem is to include a little reset sequence at the beginning of the scripts. Then it works just fine to unplug and replug the cable and then run the script again.

The recorder had to be prepared for start and stop conditions as well as the trigger of the samples. Start condition was set to start at detection of a rising edge of TMS. The stop condition was implemented as a counter that counted the number of clock cycles TMS was low. If there is no activity on the TMS line within 5 milliseconds the recording should terminate. The TMS signal controls the TAP controller and is only active during JTAG sessions, hence the above start and stop conditions.

Since the JTAG standard specifies that all sampling of the signals is during the rising edge of TCK the main method for recording would be to trigger on the rising edge of TCK. It was discovered though that TDI changes its value on the rising edge sometimes, hence the ability to sample twice every TCK period was also prepared for.

The sample buffer consists of shift registers, hence the ability to select a preferred value in that chain is implemented. The sample mechanism is illustrated in Figure 5.2, and the mux_ctl is set here for this purpose. There were three cases of interest; to store the sample right on the rising edge of TCK, right before the edge and right after the edge, the trigger is modified to control this.

This makes it six recording types, listed in Table 6.1, that were going to be tested. It is three different timing modes respectively for only the rising edge trigger and the trigger for transition on both edges that are up for testing.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One sample per $T_{TCK}$, right before rising edge.</td>
</tr>
<tr>
<td>2</td>
<td>One sample per $T_{TCK}$, on rising edge.</td>
</tr>
<tr>
<td>3</td>
<td>One sample per $T_{TCK}$, right after rising edge.</td>
</tr>
<tr>
<td>4</td>
<td>Two samples per $T_{TCK}$, right before the rising and falling edges.</td>
</tr>
<tr>
<td>5</td>
<td>Two samples per $T_{TCK}$, on the rising and falling edges.</td>
</tr>
<tr>
<td>6</td>
<td>Two samples per $T_{TCK}$, right after the rising and falling edges.</td>
</tr>
</tbody>
</table>

Table 6.1. List of the six recording types.

### 6.2 Running the System

The first step was to test the `print_a` script a few times for every recording method. Each test is repeated three times with a series of 80 `print_a`. The median value

---

1 The characters for new line and carriage return respectively.
2 The terminal window is 80 characters wide, therefore each test series can fit on one line.
of the three tests is then saved and presented in the Tables 6.2, 6.3 and 6.4. The tables all display the number of successful tests. This procedure is repeated three times for each recording method and speed.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
<th>2.5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>72</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>47</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>47</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.2. First test series of print_a.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
<th>2.5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>72</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>18c</td>
<td>76a</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>76</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>44</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.3. Second test series of print_a.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
<th>2.5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>41</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>73</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.4. Third test series of print_a.

In most of the cases a character was printed on the serial port, unfortunately it was not the wanted character ‘a’ that showed always. In normal mode the result was about 50% correct. The faulty ones showed the character ‘Í’. When running at half the speed, most of the times that character showed every time. A few cases though it showed the correct character more than 90% of the times. The ‘Í’ character does not relate to ‘a’. Regardless of which character printed the same ‘Í’ showed up when it failed to print the intended one. At one time running playback from rec_cmd 4 in 10 MHz, as seen in Table 6.3, it did in fact print the character.
‘c’, instead of the intended ‘a’, but still a lot of ‘Í’. When played back in 5 MHz ‘a’ appeared again though.

For both single and double flanked samples the recording methods storing samples right before the transition of TCK seemed slightly better than the other ones, so the tests with print_16 only focused on those. Also the quarter speed playback was neglected since it did not improve the results from the previous tests anything at all. Unrecorded testing of that was performed though.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>37</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 6.5. First test series of print_16.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 6.6. Second test series of print_16.

<table>
<thead>
<tr>
<th>rec_cmd</th>
<th>10 MHz</th>
<th>5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>39</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 6.7. Third test series of print_16.

The print_16 tests, presented in Table 6.5, 6.6 and 6.7, gave about the same rate of correct characters as the print_a tests at full speed. One could expect that the result at half the speed should give either no correct characters or almost everyone correct. This was not the case, it gave much less correct ones than at full speed - although it was remarkable that the correct ones lined up equally almost every time.

6.3 Relations of ‘a’, ‘c’ and ‘Í’

There are no real explanation discovered for the ‘Í’ phenomenon. Regardless of which character is sent on the serial bus it always shows up when the expected one does not. A plausible reason is that it may be some kind of error code. It is most unlikely that there is bit error problems with the Í, it is nothing like the other two comparing the ASCII codes in Table 6.8.

The occasion when the character ‘c’ appeared is most likely due to a bit error, since the ascii codes for a and c only differ from one bit. The indication on this one
6.4 Phase Shifted Playback

A few times when recording on both flanks of TCK the playback got 180 degrees phase shifted according to the generated TCK. When this occurred the ability to phase shift TMS and TDI was a good solution to make the playback work anyway. This was not the main reason for implementation of this functionality though. It was for making it possible to skew the TMS and TDI forward and backwards in relation to the TCK signal. A lot of elaboration with this was made as an attempt to solve the ‘Í’-problem. The phase shifts did not change the behavior of the playback unless when the skew was to big to mess up the whole playback session. This was typically when the TMS signal was changed together with rising edge of TCK.

6.5 Dump Recorded Data

Using the dump-function described in Section 5.6 it showed that one recording always gave the same output. It also showed that two similar recordings did not end up with identical outputs. Each memory dump was saved to a separate file and these files was then compared using diff³.

6.6 Reconstruct the Faulty Behavior

There are reasons to believe that the ‘Í’ behavior originates from timing issues and to stress this the JTAG tunnel was modified.

Introducing a series of flip-flops into the JTAG tunnel the timing issue may be answered. The function for changing the phase of the signals is reused to skew the TMS and TDI signals in front of and after the transitions of TCK - that is selecting how many flip-flops the signals should propagate through before it is routed to the STB. Since the system clock speed is so much faster than the JTAG clock a couple of flip-flops should not have any big impacts.

³Diff is a command for comparison of text files. Each line of two files are compared for differences.
In order to be able to skew the TMS and TDI signals to change before the TCK signal, the TCK signal must be routed through at least one flip-flop, preferably two. This turned out to not work at all. GDB could not establish any contact with the STB when having it this way.

The scenario when loop through the TCK signal and delaying the other two did not have any real effect at all, unless they were delayed too much.

RC delay [6] could be used for delaying the signals by hardware without the interference of the FPGA. Taking advantage of the input capacitance a potentiometer could be used instead of a fixed size resistor as an easy way of variate the physical delay\(^4\). This was done for the TMS, TCK and TDI signals individually, see Figure 6.1.

Running the print_a script in repeat while adjusting the potentiometers separately the same behavior, as when the system was tested, could be achieved.

This two tests both support the theory that timing issues are the root for the ‘I’ problem.

![Figure 6.1. The concept of the RC-delay circuitry.](image)

### 6.7 Clocked Down JTAG

The JTAG box could be configured to run in lower speeds than 10 MHz. The speed could be halved all the way down to 610 Hz. If the problem is timing related then one should be able to record and playback JTAG scripts at lower speeds at least.

Once again print_a is used as reference and this time the testing is concentrated to rec_cmd 5. Then both the JTAG speed as well as the playback speed is varied individually different. It turned out that recording in maximum 2.5 MHz and playback in 1.25 MHz resulted in 100% correct every time. It even worked to

\(^4\)Eg. \(t = \tau = RC\)
record in 610 Hz to playback in 1.25 MHz also. The same results was also achieved with the print_16 script.

Next test to perform was the print_uart script. Success of playing back this could very well implicate that the boot loading also could work. It turned out that the enabling of the UART could not be recorded at grater speeds than 1.25 MHz. Playback could still be performed in 1.25 MHz though.

The final test was to record the boot loading script. If this works the thesis have proven that the concept with a JTAG recorder is possible. The expected behavior was achieved when trying to load the boot loader - it succeeded when recording and playing back in 1.25 MHz. The playback in 1.25 MHz also worked after recording the session in slower speeds.
Chapter 7

Conclusion

The project turned out to be a quite complex problem with many critical parts to solve. To even be able to begin the design of the JTAG recorder a development board with enough memory that is fast enough must be chosen and a proper controller for that. Another problem was how to connect the JTAG bus to the FPGA in a proper way without introducing to much noise and interference.

The system is timing critical for both recording and playback. It is very important that the playback is identical to the recorded session, not only for the data flow, but also the output must come within certain time intervals. The memory is the biggest bottleneck in this case since all data has to be written to the memory during recording as well as read from during playback. To make it a little less sensitive, read and write buffers were introduced so that data could be sampled simultaneously as the store buffer is written to the memory and when playing back another set of data is read from the memory instantly after the output buffer is copied to the playback buffer. Integrating the memory controller and creating a usable interface consumed much more time than expected.

All these critical and complex problems was overall manageable and it turned out that it was possible to load a bootimage and start a set-top box by a recorded session. Although it did not work as intended with a TCK of 10 MHz, it actually did work flawless in 1.25 MHz. This could probably be a limitation due to the I/O pins on the development board and may very well work better using better connections or another 'faster' development kit. Unfortunately time ran out and this could not be investigated any further.

Although the system did manage to record the boot loading script and the STB did start up upon playback the system still is dependent on the JTAG box. The actual initialization of the JTAG circuit inside the processor is not yet tested for record. The time was simply not enough and the part where the TCK signal is silent is currently not supported in the record mechanism. However there is no reason to think it would not work after implementing support for that.

Even though the presented system is not fully adequate right now, and there is much left to do before it could replace the JTAG box in the test fixtures, this thesis states that it is possible to build such a system that can replace it and it
can be made to a markedly less cost.
Chapter 8

Future Work

In order to get a working product that can replace the JTAG boxes there are a few things left that must be done and a few things more that can be done to increase the quality of the product. At this stage the system has a lot of features such as a simple interface of controlling it, there are still some fundamental functions missing though.

- Implement the ability to save and restore a recorded session. It should be relatively fast and could be transferred through either USB or ethernet or maybe using an on-board flash memory.
- Understand and/or solve the speed problem so that the playback could be performed in at least 10 MHz.
- Construct a smaller custom made PCB including just the components required of the system, such as the FPGA and the memory module as well as the physical connectors.

The tasks mentioned above must at least be investigated before creating a working replacement product. There are also things that could be improved to increase the quality and some of them may have a considerable advantage of implementing before some of the critical ones.

- Device ID scan - The JTAG standard specifies how to read the device identifier which every JTAG compatible device must support. This would be useful to implement as a first life sign of the target board. If the factory test station fails to read the device identifier it should quickly stop the test. The foundation for this already exists only minor tweaks are needed for implementation of this feature.
- Implement a soft CPU core that controls the system. Such a core could facilitate communication issues such as save/load of a recorded bootimage JTAG session. It could also provide a flexible interface to integrate with the rest of the test system. Other practical examples are to define start/stop conditions for the recorder, specify the recording methods and so on.
- Make it manufacturer independent so that the JTAG recorder could be used in test systems for STBs based on different architectures.

- Understanding the bits and pieces - Instead of just record the signals it would be beneficial if the purpose of the different bits could be identified. This would be useful for error handling and it would make it possible to send only a normal ROM file over TFTP instead of a pre-recorded image. A ROM file would be much smaller, about 100 kB, and much simpler to use.
Bibliography


