Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2009-02

M.Sc. Thesis

90 nm VLSI Design of an 8-bits Microcontroller

Jose A. Moar Gomez

Abstract

Integrated Circuit (IC) design complexity has increased radically since the first "hand-made" designs in the late 50s, with a few transistors. Nowadays, Very Large Scale Integration (VLSI) designs contains hundreds thousand, million or even billion transistors and not only the experience of the designer, but also Electronic Design Automation (EDA) tools and some methodology is needed.

The purpose of this thesis is the design in 90nm UMC technology of an 8-bit microcontroller for its final manufacture using Modelsim, Design Compiler and SoC Encounter and following a standard cell design methodology. During the different steps of the VLSI flow (behavioural specification and verification, synthesis and layout generation), it will be shown how to deal with the design issues that arise: (DFT insertion, clock gating, clock&reset tree generation, etc.)

Starting from a tested FPGA implementation in VHDL based on the AVR ATmega103 microcontroller from ATMEL, the final result is an 8-bit microcontroller with the following features: 16k x 16 bits of Program Memory (PM), 8K bytes Data Memory (DM), 256 bytes parameter memory, 3 8bits I/O ports, UART, SPI, JTAG interface, additional PM programming capability (apart from the JTAG) and Wishbone interface including an USB 1.1 slave and another slave to implement and test a simple scan chain prototype. In addition the clock frequency can be increased up to 200 MHz.

90 nm VLSI Design of an 8-bits Microcontroller

Thesis

submitted in partial fulfillment of the
Requirements for the degree of

Master of Science

in

Microelectronics

by

Jose A. Moar Gomez
born in A Coruña, Spain

Committee members

Advisor:	**Dr.ir. T.G.R.M. van Leuken**
Member:	**Prof.dr.ir. A.J. van der Veen**
Member:	**Dr.ir. W.A. Serdijn**

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Abstract

Integrated Circuit (IC) design complexity has increased radically since the first "hand-made" designs in the late 50s, with a few transistors. Nowadays, Very Large Scale Integration (VLSI) designs contains hundreds thousand, million or even billion transistors and not only the experience of the designer, but also Electronic Design Automation (EDA) tools and some methodology is needed.

The purpose of this thesis is the design in 90nm UMC technology of an 8-bit microcontroller for its final manufacture using Modelsim, Design Compiler and SoC Encounter and following a standard cell design methodology. During the different steps of the VLSI flow (behavioural specification and verification, synthesis and layout generation), it will be shown how to deal with the design issues that arise: (DFT insertion, clock gating, clock&reset tree generation, etc.)

Starting from a tested FPGA implementation in VHDL based on the AVR ATmega103 microcontroller from ATMEL, the final result is an 8-bit microcontroller with the following features: 16k x 16 bits of Program Memory (PM), 8K bytes Data Memory (DM), 256 bytes parameter memory, 3 8bits I/O ports, UART, SPI, JTAG interface, additional PM programming capability (apart from the JTAG) and Wishbone interface including an USB 1.1 slave and another slave to implement and test a simple scan chain prototype. In addition the clock frequency can be increased up to 200 MHz.

Acknowledgments

First of all, I would like to specially thank my parents, my godfather and all my family in general for all the support throughout my academic life; without them, this would not have been possible. I also would like to acknowledge Estela, my girlfriend, for all the help, patience and support during the consecution of this thesis.

I am extremely grateful to my advisor Rene van Leuken, for offering me the possibility to make the present thesis and all the given assistance, and to Alexander de Graaf and Laura Bruns for all the help in the technical and administrative part respectively.

And last but not least, I would like to express my gratitude to all my friends from:

Vigo: Lu, for being always wishing to help me in the difficult moments. Diego, Gonzalo and Mario, who were the best flat mates possible. Carlos, for all the good moments in Vigo and Delft. Carliños, Eviña, Raquel, Maruchi, Elena, Chus, Camilo, McEiras, Natalia, Patiño, Alberto, Fernando... for all the nice time we spent together. And of course Cris, who made really special my years in Vigo, which I will never forget.

Delft: Dani, David, Marta and Yago, who helped me a lot during my first weeks in The Netherlands. Ruben, Guille and Javi for all the unforgettable moments. And in general, to all my present friends in Delft and the ones that already left: Chema, Lola, Manu, Antieh, Anna, Victor, Xesc, Scarlett, Blanca, Ali, Iria, Yvan, Anxo, Andrs, Martn, Diego, Javi, Andres, Estefi, Esther, Raposo, Rebe, Iñigo, Hector, Luis, Alvaro,... and specially to all my house mates for making me feel as in my hometown: Calvin, Lynn, Robin, Robbert, Dennis, Bas, Arthur, Charlotte, Halie, Hans, Gaby, Annelote, Eva, Rik and Joris.

A Coruña: Bruño, Noelia, Souto, Rebro, Rubn, Uri, Chechu, Robert, Ana and Alba for having been always there.

Jose Moar
Delft
February 24th, 2009

Contents

List of Figures

List of Tables

1 Introduction

Current society would be unimaginable without electronic gadgets. Nowadays, nobody can imagine having no Internet, computers, TV, mobile phones, etc. and the trend of dependence on technology is still growing (Figure 1.1). In this scheme of development, electronic devices are becoming more and more complex and initial integrated circuits (IC) developed in the 50s with a few transistors has grown to current IC with billions of transistors. This last evolution of ICs, the VLSI (Very Large Scale Integration) generation, demands a much more sophisticated way of designing than the handmade approach of the earliest ICs. There is also a big challenge involving the more and more "narrow" technology as far as CMOS manufacturing process is concerned. Coping with these huge designs and all the problems related with shrinking technology (synthesis, clock & reset tree generation, DFT techniques, power distribution ...) are the fields of this thesis.

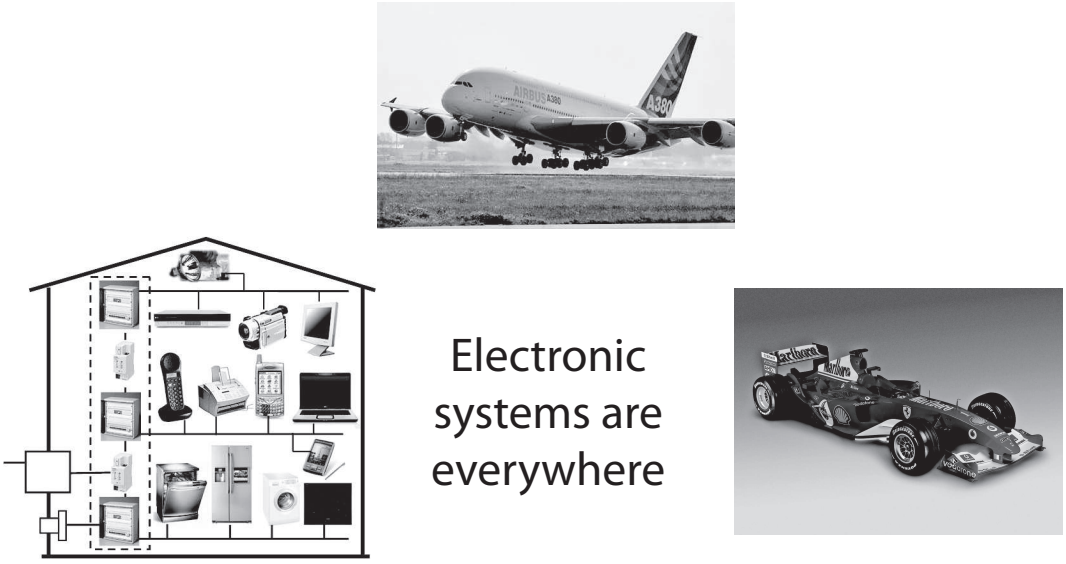


Figure 1.1: Electronic systems

1.1 Motivation

Technology advances incredibly fast allowing more complex designs which make essential a continuous activity of research and development in order to ride the technology wave in the field of VLSI design.

In the other hand, microcontrollers have a wide range of applications; it is estimated that there is around 40 microcontrollers in a typical home. Furthermore, according to Semico[1], over 4 billion 8-bit microcontrollers were sold in 2006.

[1] Semico Research Corp is an american semiconductor marketing and consulting research company

1.2 Thesis Goals

Therefore, there are two goals to be achieved in this thesis:

- The main purpose is to complete a VLSI design flow in 90 nm technology with up-to-date EDA tools (Design Compiler from Synopsys and SoC Encounter from Cadence) in order to set up a design environment for more complex and specific designs.
- Fabricate a low power microcontroller that includes the Wishbone and JTAG interfaces, USB 1.1 standard and scan chain capability.

1.3 Results

Throughout this thesis will be detailed the procedure that has been employed to achieve the following results:

- Generation of the GDSII file for an 8-bit microcontroller with the Wishbone interface, i.e., consecution of a ready-to-manufacture complex design in 90nm technology.
- Increased clock frequency from 20MHz up to 200MHz through architectural modifications and advanced design techniques such as back-annotated synthesis.
- Integration of a complex transmission interface such as the USB 1.1 into an 8-bit microcontroller using the Wishbone bus.
- Successfully application of DFT techniques using a multiplexed flip-flop scan chain.
- Clock gating implementation to reduce power consumption.
- Development of an additional programming interface for the PM.

1.4 Thesis Organisation

This thesis is organized in the following manner:

- Chapter 2 gives a basic background about the methodology and tools used in the design, and also includes a brief explanation about the microcontroller, the JTAG and Wishbone interfaces and the concept of Design for Testability using a Scan Chain approach.
- Chapter 3 presents the starting point of the design, the changes that has been made and the new functionalities.
- Chapters 4 to 7 collect all the issues that arise during the design flow with the solution(s) that has been taken.
- Chapter 8 shows the results and possible future work.

---

www.semico.com.

2 Background

It has already been said before that a line of attack should be clearly defined when dealing with actual IC designs and maybe not so indispensable with the "old" ones. Throughout this chapter will be briefly introduced the evolution of ICs, applied methodology, tools that has been used and specific technology employed for the manufacturing. Furthermore, the microcontroller, Wishbone and JTAG interfaces and Design for Testability technique will be explained.

2.1 Brief ICs history

There are some discrepancies about who conceived the first IC: Jack Kilby of Texas Instruments or Robert Noyce of Fairchild Semiconductor. Whoever was the "father", the birth of the first IC in 1958 initiated a revolution in the circuit design field.

Starting with circuits containing only a few transistors, Small-Scale Integration (SSI), the IC was already used in important applications such as the aerospace Apollo program or FM inter-carrier sound processing in television receivers. During the late 60s and mid 70s, the IC evolved to circuits with hundreds of transistors, Medium-Scale Integration (MSI), and subsequently to circuits with tens of thousands of transistors, Large-Scale Integration (LSI) .

These first designs were "handmade" and based only on the experience of the designer, but the more the circuits increase the number of transistors the more impracticable was the design without any "external help". This "external help" is known as EDA tools (Electronic Design Automation) and was an essential part of the next evolution step in IC, the VLSI age (Very Large Scale Integration), which was initiated at the early 80s with circuits including hundreds of thousands of transistors. Nowadays, this number increased beyond several billion transistors, what makes clear the need of EDA tools and some method or design flow that allows managing huge designs like that.

This growing trend was already predicted by Gordon E. Moore in his publication "Cramming more components onto integrated circuits" [6]:

> *The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.*

Later, in 1975, Moore changed his prediction to a doubling every two years and this statement remains still valid nowadays and it is known as the "Moore's law" (Figure 2.1)

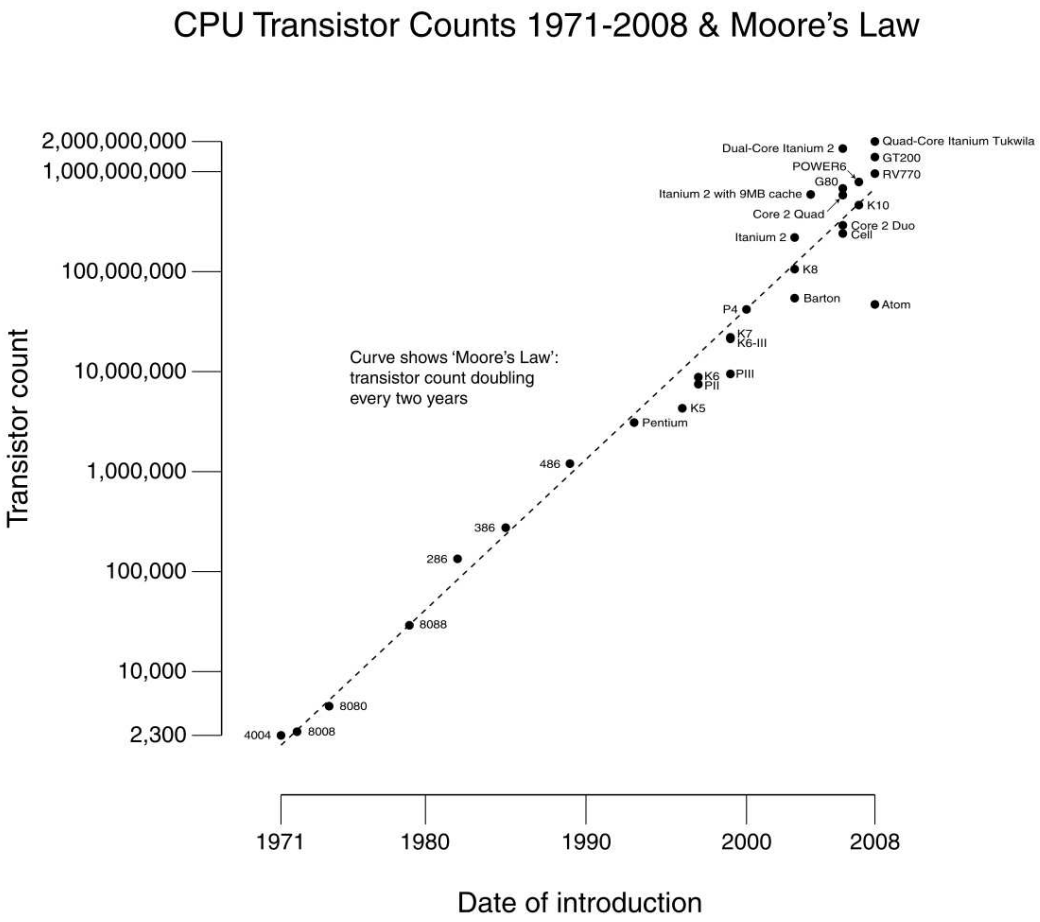


Figure 2.1: Moore's Law and CPU transistors

2.2 VLSI design flow

In the previous section, the need of some kind of methodology was stated. In this section, we will present this methodology as a design flow that allows IC designers to get an error-free circuit. The diverse steps of a typical design flow are shown in Figure 2.2 and are subsequently explained in subsections 2.2.1 to 2.2.4.

2.2.1 Specification

The design starts setting the requirements that our circuit has to fulfill. These specifications can be described with any system specification language (C, C++, Matlab, etc.)

2.2.2 RTL Behavioral Description and Verification

The specifications are now converted into an RTL behavioral description using an HDL (Hardware Description Language) such as Verilog or VHDL. The correct behaviour is then checked by a simulation program (for instance, Modelsim from Mentor Graphics or VCS from Synopsys) using test benches.

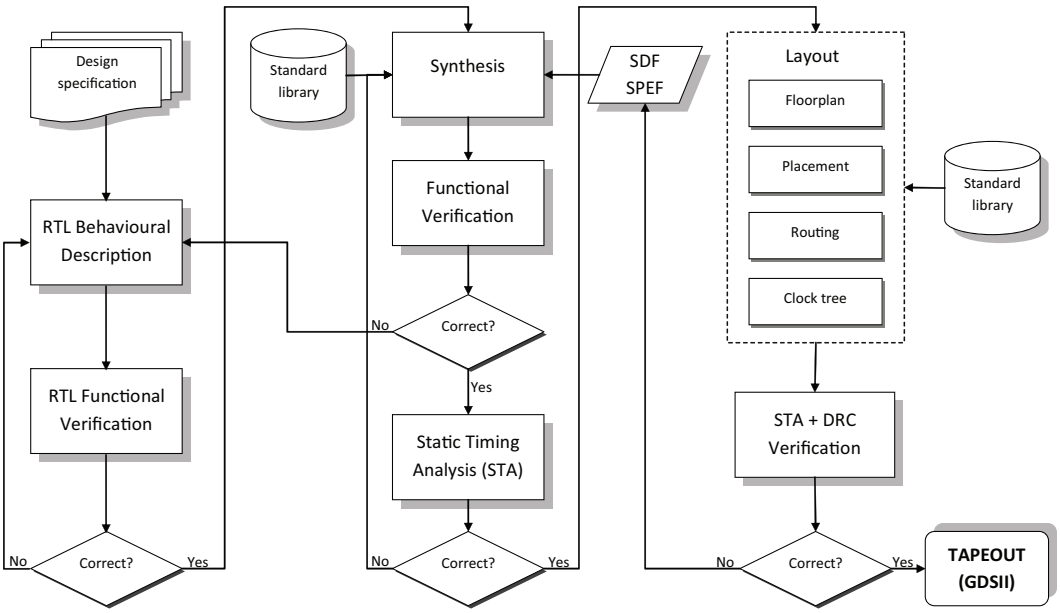


Figure 2.2: VLSI Design Flow example

2.2.3 Synthesis

Synthesis is the process by which the circuit behavior is transformed into a design implementation in terms of logic gates. It includes the following steps:

- *RTL Synthesis and Library Mapping*: The VHDL code is translated into a netlist of interconnected general gates and registers and then it is mapped to the particular gates defined in the target library. Design Compiler from Synopsys or Synplify Pro from Synplicity are typical programs used for this task
- *Functional Verification*: At this point, it is necessary to check if the structural netlist performs the same function as the original behavioral HDL. The easiest way is to run again the test benches that have been used in the behavioral step.
- *Static Timing Analysis*: The circuit is behavioral and structural equivalent but timing requirements has to be tested. A static timing analysis can be run quickly to check if the circuit is fast enough or if we should come back to a previous design step and redesign our project.

2.2.4 Layout Generation

Layout generation is the last step before sending the chip to fabrication. It takes the structural netlist from the previous step and generates the physical layout. The next steps are comprised in the layout generation:

- *Floorplanning*: It is being more and more common to perform and initial manual floorplanning before the automatic placement. In this way, some hierarchy is given to the design in order to avoid placing a "flat" design. The benefit of this approach is to get closer modules that has to communicate with each other with the purpose of minimize the wire length.

- *Placement*: Where to place the standard cells is the first task that needs to be solved in the Layout Generation. The simple idea is to minimize the length of wires, but, for example, in a *timing-driven placement* the intention is to decrease as much as possible the delay on the critical paths.

- *Routing*: At this point, the cells are placed and they need to be interconnected. The routing step can be divided in two stages: global routing and detailed routing. In the first stage (global routing) the problem is abstracted to establish through which channels the connections will flow. Then, in the second stage (detailed routing) the exact position of a connection wire within a channel is determined.

- *Timing Analysis*: This is the critical step of the design flow and it can be seen as a bottleneck in the process. The static timing analysis is rerun after the place and route in order to check if the timing requirements are accomplished. It can be necessary some iterations of synthesis and P&R (place and routing) before our goal is reached. This iteration process is called *back-annotated synthesis*.

- *Clock tree generation*: How the clock is distributed in a design is one of the most limiting factors in order to achieve high frequency circuits. The clock signal has to get all the clocked components at the same time in order to achieve a correct operation, and the more the frequency is increased, the more clock inaccuracy we have. The clock inaccuracy consists of two elements: the clock skew and the clock jitter. In addition to this, it is also important to take care about power consumption reduction using clock gating. This factor is extremely important because, for example in Intel chips, the clock network can be even 50% of the total power consumption.

2.3 Standard cell design

There is one more aspect that should be pointed out in addition to the design flow explained in Section 2.2: how is actually the physical layout generated? Typically there are two main options:

- *Full custom design*: The designer has to generate the layout of each transistor and the interconnection between all of them. This approach allows maximizing the performance of the circuit, but it is not possible to apply when dealing with big designs such a microcontroller.

- *Standard cell design*: In the other hand, standard cell design takes advantage of the repetition of smaller sub-circuits to create a level of abstraction that allows the designer focusing on the high-level (logical function) aspect of the digital design. These smaller sub-circuits can be seen as "low-level layouts", created using a full custom technique, that are encapsulated into abstract logic representations (logic gates, flip-flops, buffers, etc.).

When opting for the latter design method, as is the case in this thesis, a library containing the standard cells should be provided. The library that has been used in this design is explained next.

2.3.1 Faraday 90nm Standard Cell library

The FSD0A_A library from Faraday used in this thesis is a 90nm standard cell library for UMC's 90 nm logic SP-RVT (Low-K) process, where SP[1] stands for "Standard Performance" and RVT for "Regular Voltage Threshold". Figure 2.3 shows a 3D representation of a standard cell. The Low-K term refers to the small dielectric constant (κ) of the material that has been used to replace the silicon dioxide in the manufacturing process. This substitution is aimed to reduce parasitic capacitance, enable faster switching and get lower heat dissipation. All these manufacturing process related issues are beyond the scope of this thesis and we refer to [2], [3] and the UMC documentation for more details.

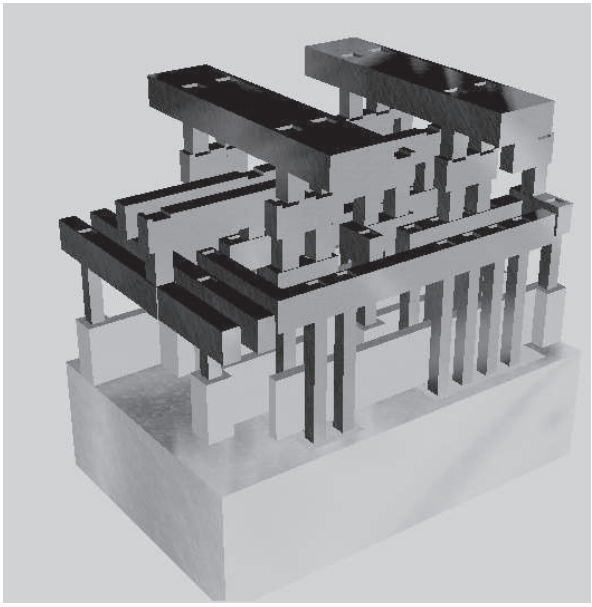


Figure 2.3: Standard cell with three metal layers

This library supplies a set of common core cells (logic cells, flip-flops, latches, RAM cells...) with up to 12 different drive strengths in order to improve performance and also includes 2.5V I/O cells with the following programmable capabilities:

- Input pull-up/pull-down/keeper control
- Schmitt trigger control
- Input gated control
- Output slew rate control
- Different output driving possibilities (from 2 to 16 mA)

All the I/O cells are available in Pad-Limited or Core-Limited[2] versions.

2.3.2 Memory Compiler from Faraday

The embedded memories used in this thesis are also intended for UMC's 90 nm logic SP-RVT (Low-K) process. Faraday Technology supplies a memory compiler where different

[1] UMC offers different options in 90nm technology depending on the application: LL (Low Leakage) devices are intended for portable and wireless applications, HS (High Speed) option is available for graphics applications while SP is a trade-off solution between performance and power consumption.

[2] Pad- and core-limited refer to the most restrictive factor in terms of size: the amount of core logic or the number of IO pads. Pad-limited pads are narrower and longer, while core-limited are wider and shorter. This makes sense as overall area is smaller.

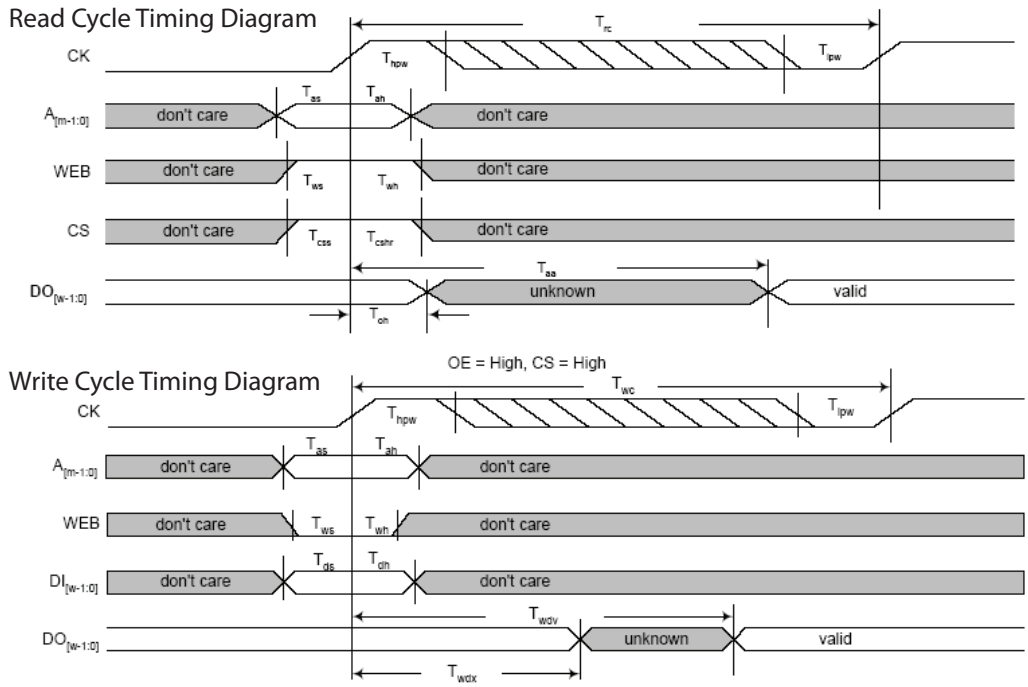


Figure 2.4: Read and write cycle timing for the SRAM used as PM

combinations of words, bits, and aspect ratios can be selected to generate the memory. In addition, the memory compiler also provides the data sheet, Verilog and VHDL behavioural simulation models, LEF files for the place & routing, etc. For instance, Figure 2.4 and Table 2.1 show the read and write cycle timing diagram for the SRAM used as PM with the corresponding values.

2.4 EDA tools

We will present concisely the three EDA tools that have been used during the VLSI design flow in this thesis:

- *Modelsim SE 6.3a* from Mentor Graphics is a hardware simulation and debug environment quite popular between IC designers. It provides the designer with the possibility of verifying the functionality of the circuit and also including timing information through an "SDF" file (Standard Delay Format). It supports multi-language simulation (VHDL, Verilog, SystemVerilog) and TCL/tk scripting, among other features.

- *Design Compiler* (version Z-2007.03-SP4 and B-2008.09-SP1-1) from Synopsys is the EDA tool use to perform the logic synthesis, i.e., transform the abstract description of the circuit into a design implementation in terms of logic gates. It offers the facilities for facing successfully with the challenges of a design: timing, area, low power and high test coverage.

- *SOC Encounter* 6.2 from Cadence completes the last step of the VLSI design flow, i.e., it performs the floorplanning, power distribution, place and routing, clock tree

Symbol	Description	BC	TC	WC
taa	Address access time from CK rising	0.93	1.46	2.70
toh	Output data hold time after CK rising	0.45	0.71	1.31
trc	Read cycle time	1.17	1.79	3.21
tcss	CS setup time before CK rising	0.10	0.17	0.31
tcshr	CS hold time after CK rising in read cycle	0.00	0.00	0.00
tcshw	CS hold time after CK rising in write cycle	0.00	0.00	0.00
twh	WEB hold time after CK rising	0.06	0.08	0.14
tah	Address hold time after CK rising	0.00	0.00	0.00
tas	Address setup time before CK rising	0.04	0.08	0.19
twc	Write cycle time	1.18	1.80	3.17
tws	WEB setup time before CK rising	0.05	0.08	0.15
tdh	Input data hold time after CK rising	0.03	0.04	0.07
tds	Input data setup time before CK rising	0.08	0.13	0.25
twdv	Output data valid after CK rising	0.93	1.46	2.70
twdx	Output data invalid after CK rising	0.45	0.71	1.31
thpw	Clock high pulse width	0.09	0.12	0.16
tlpw	Clock low pulse width	0.10	0.15	0.27
toe	Output data valid after OE rising	0.07	0.11	0.19
toz	Output data go to Hi-Z ater OE falling	0.04	0.06	0.11

Table 2.1: Timing values for the SRAM used as PM (ns)

synthesis..., in order to produce the GDSII file (Graphic Data System) for the final tape-out of the circuit.

2.4.1 Getting help

All the tools described before are delivered with extensive documentation and huge manuals, but it does not take too long in turning out insufficient. This problem mainly applies for Design Compiler and SOC Encounter which are quite expensive programs, mostly used in companies, and therefore, it is difficult to get information about how to solve specific problems. On the contrary, Modelsim is a much more common software being used in professional but also educational environments. According to this, the following "sources of help" are suggested:

- When dealing with Modelsim, the fastest way to solve any kind of problem is Google (even faster than consulting the own Modelsim manual)
- For problems with Design Compiler, after checking the user guide, SolvNet can be used. *Solvnet.synopsys.com* is the on line resource for Synopsys tool support and downloads, that offers access to the Synopsys knowledge data base containing up-to-date product manual, technical articles and day-a-day issues posted by Synopsys users.
- *Sourcelink.cadence.com* is the equivalent of Solvnet for Cadence.

Both Synopsys and Cadence allows designers to post question on theirs sites, but unfortunately, this facility is not supported for university accounts.

In addition, there are some web sites that can be also really helpful with these tools and also for clarifying some concepts. A small selection is presented:

- www.edaboard.com

- www.deepchip.com

- www.edacafe.com

2.5 Design for Testability

Testing and validation is an important issue in IC design that is often overlooked. A correct design is not synonymous with an error-free manufactured component because manufacturing defects has to be taken into account: impurities in the silicon crystal, short circuits between wires or layers, broken interconnections, etc. Clearly, it is necessary to validate the circuit after the manufacturing process, nevertheless, testing a component that has been designed without having that purpose in mind can be extremely expensive in terms of money and time. Therefore, when considering test capabilities in a design, there are two important properties:

- *Controllability* determines the ease of setting an internal node to a certain value. The best example of high controllability is a node that can be settable through an input pad. A circuit having nodes with poor controllability takes extremely long to get it into a specific state, and, sometimes, it is not even possible.

- *Observability* can be seen as a measure of the ease of observing a specific circuit node at the output of the integrated circuit. Ideally, it should be possible to observe every single gate output either directly or indirectly (within some clock cycles).

Design For Testability is a design technique, of which the objective is to improve controllability and observability in the circtuit. The two main approaches are:

- *BIST (Built In Self Test)*

- *Scan Design*

Scan design is the approach used in this thesis and will be discussed in the next section. An explanation about BIST and another aspects of the DFT technique can be found in [7].

2.5.1 Scan Design

Scan design is a DFT strategy that consists in replacing the normal registers by scannable registers. The scan registers can operate in two modes: in *normal mode* the register operates as normal registers while in *scan mode* they are connected in order to create a *scan chain*. A scan chain can be seen as a shift register where the designer can shift data in and out. As will be shown in Sections 2.5.1.1 to 2.5.1.3 three extra IO ports are needed in the design. In this thesis we will use the "DFT Compiler" capability from Design Compiler to implement a simple scan chain in the design. DFT Compiler supports the following scan models:

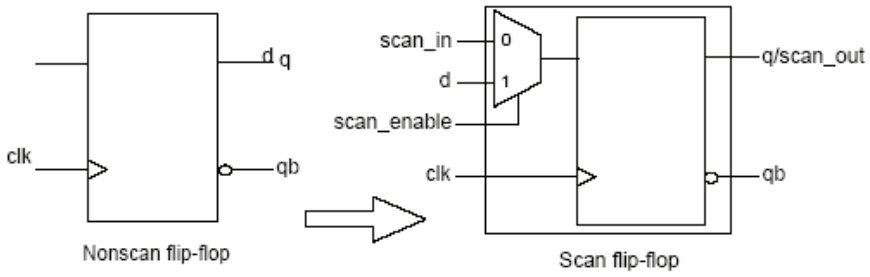


Figure 2.5: Multiplexed scan style

2.5.1.1 Multiplexed Flip-Flop Style

The most basic scannable flip-flop can be achieved combining a normal D flip-flop with a multiplexer as shown in Figure 2.5. The main advantage of this style is its low area overhead while the main disadvantage is the additional delay introduced by the multiplexer, which can become really critical when the register is part of the critical path. The multiplexed scan style is the most commonly supported in technology libraries (Faraday library used in this thesis also provides specific scannable D flip-flops) and requires the following test pins:

- Scan input
- Scan output
- Scan enable

In Section 5.3 is explained how to implement a simple multiplexed style scan chain. A more complete explanation about all the details involving scan insertion can be found in [8] and [9].

2.5.1.2 Clocked Scan Style

The difference with the previous style is that, now, instead of a selection signal an additional clock is presented. When this clock is active, the test data is shifted into the register. Figure 2.6 shows an example of clocked scan style, which also needs three test pins (the scan enable pin is now substituted for the test clock pin). The difference with the previous style is that, preserving the low area overhead, a better performance is obtained with the cost of an additional clock.

2.5.1.3 LSSD Style

The Level Sensitive Scan Design (LSSD) style consists in two latches working as master-slave pair. It has the best performance in comparison with the previous styles at the expense of a high area overhead and an extra test pin. There are three variations of the LSSD scan style: single-latch, double-latch and clocked that are explained in [8]. Figure 2.7 shows the substitution of a normal latch for a single-latch LSSD style.

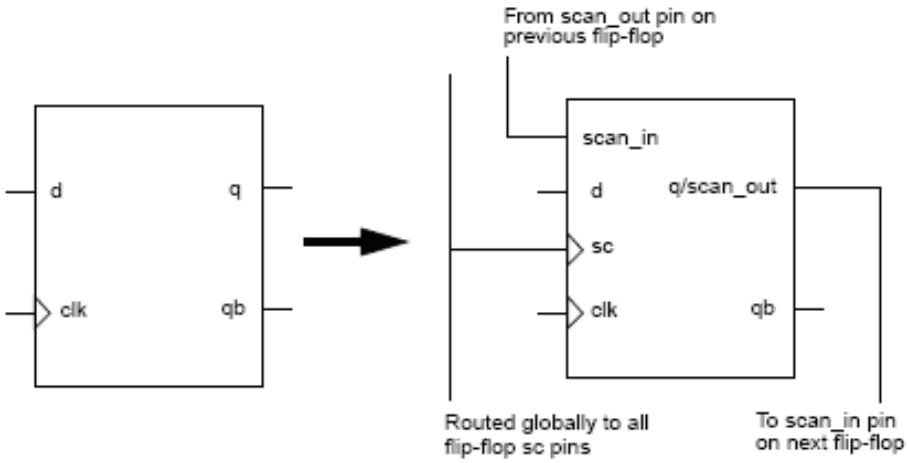


Figure 2.6: Clocked Scan Style

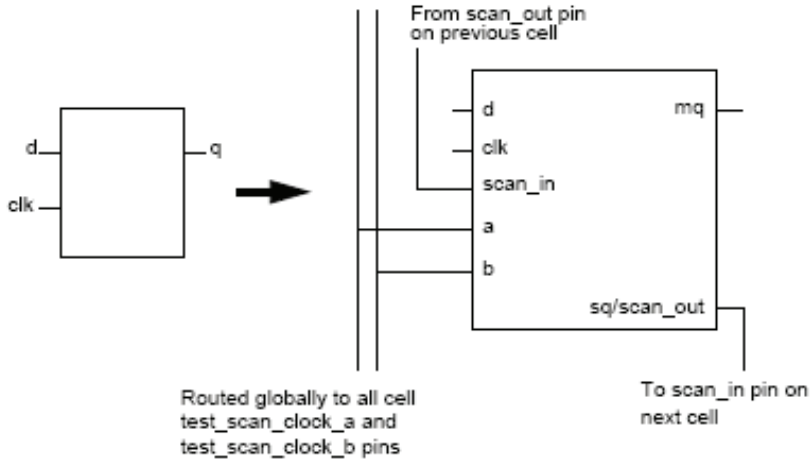


Figure 2.7: Single-latch LSSD Style

2.6 Wishbone Bus Interface

Wishbone Bus Interface is a System on Chip (SoC) interconnection method that allows integrating digital circuits together in a chip and it is used by a lot of designs in the OpenCores[3] project. The Wishbone Bus is a flexible, simple and portable way of interconnecting "IP Cores" (Intellectual Property Cores) to our circuit, which means that a new functionality can be added quickly and easily in the design. It can be seen as a standard for the IP Cores interfaces, which is very useful for independent development of the cores but assuring a final successful integration. The Wishbone Bus has been specified as a "logical bus", that is to say, in terms of "signals", clock cycles and high and low levels. The ambiguity in electrical information or bus topology lets designers more freedom to combine different designs. It presents a Master/Slave architecture with four different types of interconnection:

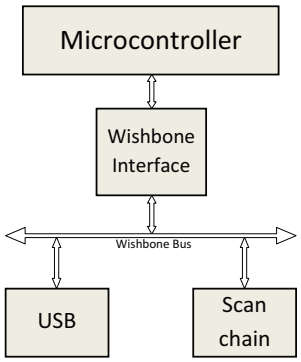


Figure 2.8: Wishbone shared bus interconnection

- *Point to Point*: It is the simplest interconnection. Allows only one master to communicate with only one slave.
- *Data Flow Interconnection*: It is used when data can be processed in a sequential way. Every IP core has both a master and a slave interface and the data "flows" through the line of interconnected cores. It is a kind of pipelining because exploits parallelism and, therefore, decreasing execution time.
- *Shared Bus Interconnection*: It is a multi-master/multi-slave interconnection, that is to say, more than one master and more than one slave can be connected to the bus at the same time. In this situation, when more than one master is connected, it is necessary arbitration. Both the arbiter and the shared-bus implementation are entirely specified by the system integrator. A possible example could be a PCI bus with round-robin.
- *Crossbar Switch Interconnection*: It is the last and most complex way of interconnection in the Whisbone bus. Crossbar switch interconnection allows two or more master to communicate at the same time with two or more slaves (each slave can only be addressed by one master, that is to say, two masters cannot addressed the same slave, but it is possible different slaves at the same time). In this case, the arbiter indicates when a master can gain access to a specific slave. The average data transfer rate is higher than in the shared bus scheme but the requirements in interconnection logic and routing resources are also higher.

In this thesis will be used a "shared bus interconnection" to communicate one master, a microcontroller implementing the Wishbone interface, and two slaves, an USB (see Section 3.2.4) and a scan chain prototype (see Section 3.2.5). This architecture can be seen in Figure 2.8

2.7 AVR Microcontroller

The microcontroller implemented in this thesis is based on the "AVR ATmega103" from Atmel. The ATmega103 is an 8-bit microcontroller with the following main features:

- RISC architecture with 121 instructions and 32 8-bit general purpose registers
- 128KB PM, 4KB DM and 4KB EEPROM

- SPI, UART, Watchdog, two 8-bit timers, PWM, ADC and 32 programmable I/O lines

Some of this features has been changed in the design implemented in this thesis, (for instance, the new sizes of the memories are 32K for the PM, 8KB for the DM and 256bytes EEPROM), another ones have been removed (Watchdog, PWM, ADC) and some new features have been included. All these changes can be seen in Chapter 3.

2.8 JTAG interface

JTAG (Join Test Action Group) was an industry group founded in 1985 to create a method to test circuit boards that were becoming smaller and smaller and, in this way, much more complex to be tested. The industry standard developed became in 1990 an IEEE standard, 1149.1 IEEE Standard Test Access Port and Boundary-Scan Architecture. The method devised by this group was how to perform a boundary-scan testing at the IC level, which means that a large debugging can be performed through a small numbers of test pins, and it is the most popular and used "design-for-test" technique nowadays. The JTAG interface consists at least of three inputs ports and one output (there is an optional extra input for asynchronous initialization of the test logic).These set of pins are known in JTAG terminology as the Test Access Port (TAP) and they are the following:

- *TDI* (Test Data In) is sampled at the rising edge of TCK and shifted into the device's test or programming logic.
- *TDO* (Test Data Out) represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK.
- *TCK* (Test Clock) synchronizes the internal state machine operations
- *TMS* (Test Mode Select) is sampled at the rising edge of TCK to determine the next state
- *TRST* (Test Reset, optional) resets the internal state machine when it is driven low

The operation of the JTAG interface is controlled by a 16 state-machine called the *TAP controller*. The different states are selected with the TMS signal (it is interested to point out that five consecutive ones always get the "Test Logic Reset" state). The state transition diagram can be seen in Figure 2.9. A typical test process using the JTAG interface is as follows:

1. The diagnostic data is set on the input pins.
2. The input data is saved in the boundary scan registers monitoring the input pins.
3. The output data is scanned from the TDO pin.
4. The output data is compared with the expected values according to the input data to check the correctness of the circuit.

Apart from the described functionality, the JTAG has been extended by Atmel to include the following additional functionalities in the latest AVR microcontrollers:

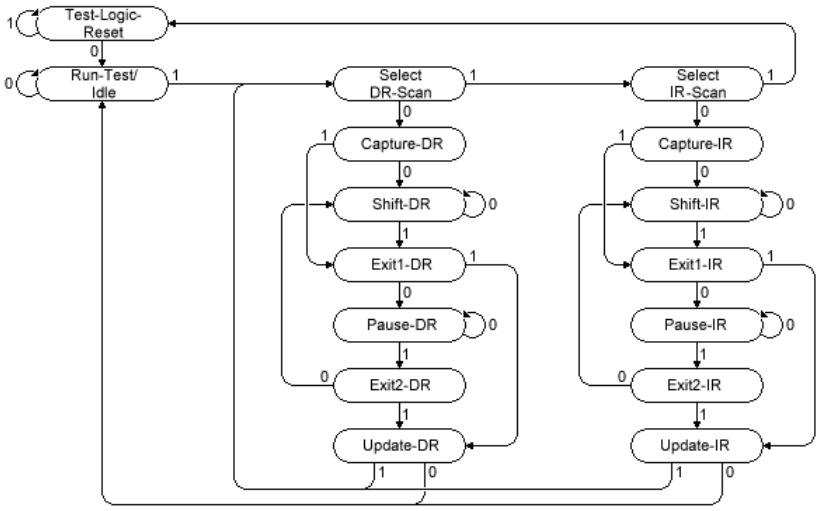


Figure 2.9: TAP controller state transition diagram

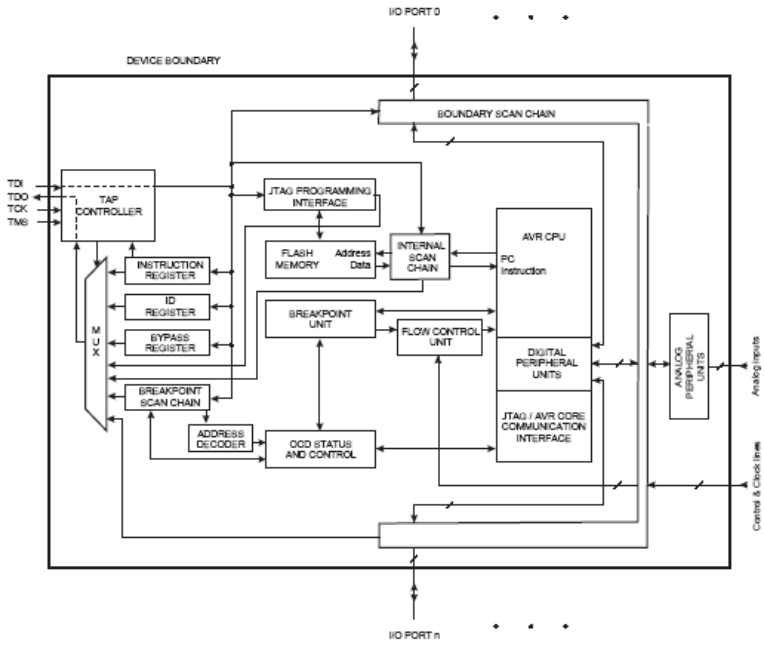


Figure 2.10: AVR JTAG block diagram

- Programming the non-volatile memories, fuses and lock bits.

- On-chip debugging with the AVR Studio.

In this thesis, only the programming capabilities for the PM using the JTAG interface have been implemented. Figure 2.10 shows the block diagram of the extended JTAG.

2.9 Summary

In this chapter a briefly introduction to the background concerning the scope of this thesis was given; including topics such as methodology employed throughout this VLSI design, tools that have been used, necessity of DFT techniques, Wishbone and JTAG interfaces, etc.

Next chapter will describe the modifications applied to the design and the new components included.

3 RTL behavioural description

As it was mentioned before, the starting point is a tested FPGA implementation of the AVR Atmega 103 including the Wishbone interface. In that case, the slave attached was an LCD controller and the design was tested successfully in a Xilinx Avnet xc3s2000 FPGA. From this, it can be assumed that the RTL behavioral description of both, the core and the Wishbone interface, is synthesisable and acceptably verified.

3.1 Modifications

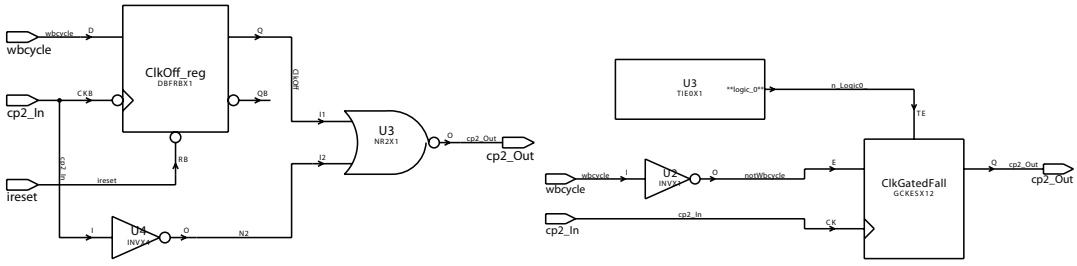
From the initial design, the following components are not necessary anymore:

- *Debouncer*: Filters mechanical switch bounces from the FPGA reset push button.
- *CPUWaitGenerator*: It is used when the core is connected with low speed memories.
- *Xilinx memories*: the program, data an EEPROM memories were implemented using the specific memory blocks from Xilinx.

The Xilinx memories have been replaced by the following embedded Faraday memories:

- 3 single port SRAM of 32K, 8K and 256 bytes for the PM, DM and EEPROM.
- 10 synchronous two-port register files for implementing the 10 FIFOs in the *USB Wishbone slave*.

Furthermore, the component *ClockSwitch*[1] has been implemented using a specific clock gate cell from Faraday in order to avoid "risky" implementations as depicted in Figure 3.1.



(a) Normal register and logic gates (b) Specific clock gate cell

Figure 3.1: ClockSwitch implementation

[1]This component is used to stop the clock of the AVR core when communicating with slower Wishbone slaves.

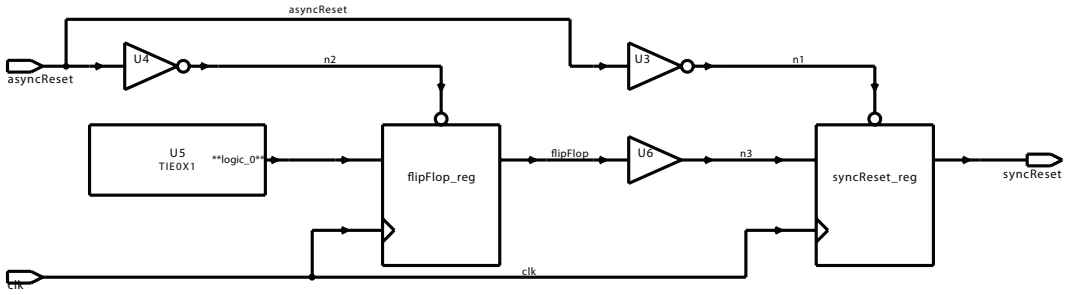


Figure 3.2: Reset Synchronizer

3.2 New components

The next list presents the new modules in the design, which are briefly explained in the following subsections:

- *Reset synchronizer*
- *Reset chain*
- *MyJtag*
- *USB 1.1 IP Core*
- *Scan chain prototype wishbone slave*
- *I/O pads*

3.2.1 Reset synchronizer

With the exception of the SPI and the USB, the rest of the components have an asynchronous reset, which means that some mechanism is needed to avoid metastability in the circuit. Metastability occurs when the reset removal violates the "reset recovery time", that is, the reset signal is (de)asserted too close to the falling/rising edge of the clock.

A possible solution is to insert a "reset synchronizer" as depicted in Figure 3.2, where only the first flip-flop has potential metastability problems since the second one has the same input and output when the reset is removed. In this way, the circuit is brought into the reset state asynchronously but the reset removal is performed synchronously in two clock cycles.

3.2.2 Reset Chain

In Section 3.2.1 was presented how to avoid metastability because of the reset removal. Another critical issue concerning this aspect is to generate a valid sequence of reset removal, that is, most of the times, not all the components in the design can be taken out the reset state at the same time. This is due to the fact that not all the components need the same number of clock cycles to recover from the reset state. For example, component A is using data generated from component B and in both of them the reset is removed at the same time. In this case, if B needs more time than A in order to present valid data after the reset has been removed, A could use wrong data from B and leads the whole circuit to malfunction.

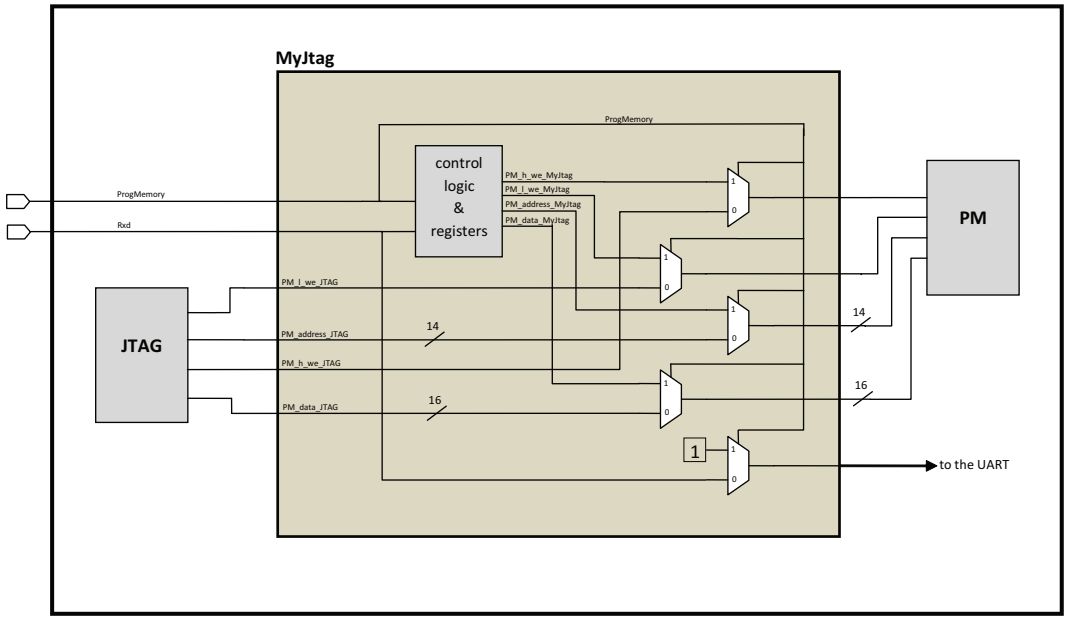


Figure 3.3: MyJtag simplified diagram block

In this thesis, the only component that presents this potential problem is the AVR core, so the reset sequence is really simple: the reset signal for all the components is removed at the same time apart from the one going to the AVR core, which is delayed 16 clock cycles.

In addition, this component also keeps the AVR core in reset state while using the JTAG or MyJtag (see Section 3.2.3) capabilities.

3.2.3 MyJtag

The function of this component is to give an alternative way of programming the Program Memory (PM) in the case the JTAG interface is not working properly due to some manufacturing defect. It is using three IO ports:

- *clkExt* (external clock pin): The purpose is to avoid synchronisation problems when reading the data into the shift register.
- *progMemory* (control pin): when set to 1 it means that this component is going to be used to program the PM.
- *rxd* (data input pin): this pin is being shared with the UART to reduce the total number of IO ports needed.

MyJtag block diagram is shown in figure Figure 3.3. Essentially, it works as follows:

1. The rxd signal has to be set high because a low starting bit is used.
2. The progMemory signal is set high so as to take the AVR core into the reset state and to select the internal data and address registers for the PM.
3. After the starting bit, data is read into the shift register with each rising edge of the external clock.

4. Every 16 bits, the address register is incremented by one and the data is written in the PM.

It is assumed that the external data is being generated by the external clock, so no synchronisation mechanism is needed. About the maximum speed of this external clock, the I/O buffer delay limits the frequency to approximately 1 GHz, and, taking into account that an instruction is written in the memory every 16 bits, it yields a PM writing rate of 62.5 MHz, which is feasible.

Anyway, there is no need in squeezing the design up to this point: with a 32K bytes PM and an external clock frequency in the range of the AVR core, for example 150 MHz, the PM will be "filled" completely in:

$$\frac{32K * 8}{15 \cdot 10^7 bps} \approx 2ms$$

which is a more than acceptable value.

3.2.4 USB 1.1 IP Core

As its name suggests, this IP Core, successfully proven in an FPGA, implements the USB 1.1 standard with the following features:

- 8-bits Wishbone slave interface
- Include low (1.5 Mbps) and high (12 Mbps) speed capability.
- Control, bulk, interrupt and isochronous transfers are supported.
- Four endpoints with independent 64 bytes FIFOs for each one.

The USB also presents 9 interruptions lines that have to be attached to the microcontroller. Unfortunately, there is only one external interruption available that is already been used for the wishbone interface. In order to solve this problem, the 9 interruptions can be "xored" in one line and, to find out what interruption has been activated, the interruption status register has to be read. Figure 3.4 shows this change in the architecture.

In addition to the previous modification, special memories from Faraday have been used to implement the 10 FIFOs used in the USB (one transmission and one reception FIFO for each endpoint and also another pair for the host mode).

Furthermore, it has to be taken into account that the USB require the following clocks:

- USB logic clock: It is the internal clock of the USB. It has to run at 48MHz with a tolerance of +/- 0.25%.

- Wishbone clock: This is the clock use for the Wishbone interface implemented in the USB. It can be asynchronous to the internal clock but it is limited to the following frequency range: 24MHz <= Wishbone clock <= 240MHz.

Further information about this IP core can be found in [5].

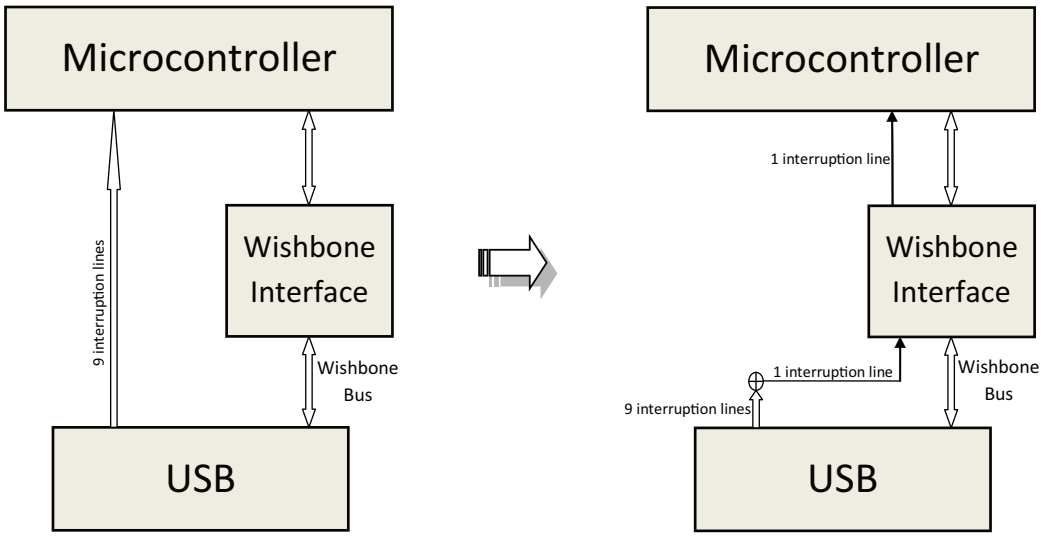


Figure 3.4: USB architecture modification

3.2.5 Scan Chain prototype in a Wishbone slave

This component has a double objective: implements a basic Scan Chain that consists of four registers and shows how the microprocessor can handle different Wishbone slaves

Concerning the Wishbone interface, the intention is to create a component that can be used to verify if the microprocessor can handle correctly more than one Wishbone slave. With that goal, a simple slave that adds two 8-bits number was implemented. It consists of four 8-bits registers: two for the addends, one for the result and one control register. The addition is carried out when the LSB of the control register is written to one. The four register can be written and read from the microcontroller.

Regarding the Scan Chain objective, the importance of Design for Testability was explained in section Section 2.5. For that reason, a first approach to this technique is put into practice as can be seen in figure Figure 5.7, where the four register used in the Wishbone slave were included in the scan chain. The idea is to set up a basic scene where this technique can be tested without affecting the performance of the other components.

3.2.6 I/O pads

Some way must be provided to communicate data between the chip and the external circuitry. In order to accomplish this function, in [3] can be seen that I/O pads must have the following properties:

- *Protects against over voltage damage*: Extremely high voltages can be put on an input pin just by touching it with a finger.
- *Drives large capacitance*: Typical values for off-chip signals are between 2 and 50 pF.
- *Protects the circuit against electrostatic discharge (ESD[2])*

[2]The term "electrostatic discharge" is used to denote the unexpected and temporary electric current that flows between two objects at different electrical potentials. This current can damage the semiconductor and insulating materials of the circuit.

- *Provides level conversion*: Voltage level must be compatible with external devices.
- *Has a small number of pins (low cost)*
- *Limits slew rates to control high-frequency noise*

A brief explanation about the I/O cells from Faraday used in this thesis was given in Section 2.3.1, where the different programmable capabilities were detailed. For our design, a pad-limited version with 8mA output driving strength has been selected due to SSO[3] (simultaneous Switching Output) limitations. With this strength, the maximum output load that can be driven is limited to 41.48pF at 200MHz and to 82.96pF at 100MHz due to electromigration effects (see Section 6.1.2 for more details).

3.3 Summary

The different modifications in order to convert an initial FPGA into an ASIC have been explained in this chapter. Also the new components and functionalities were discussed in Section 3.2, were, among other features, an additional method to program the PM was described, and the new component implementing the USB 1.1 standard was introduced.

Next, the test cases used to verify the design and how to apply them using Modelsim will be explained.

[3]In Section 6.1.2 can be seen that 16 I/O power/ground cells have been employed for 54 I/O signal cells, which yields in ratio of almost 7 I/O cells per power/ground pair. According to Faraday documentation, a higher value for the drive strength will result in important SSO noise effects.

4 Verification with Modelsim

As it was mentioned in Section 2.2, Modelsim is used to verify the correct behaviour of the design in three different phases:

- *RTL Behavioural Description*: At this stage, Modelsim is used to check that the design accomplishes with the specifications and product requirements. It has to be pointed out that only the operation is verified, i.e., no timing information is taken into account.
- *Netlist after synthesis*: Once the design is synthesized, all the test benches used in the previous phase has to be rerun in order to check that they produce the same output. The difference is that now, besides checking the correct behavioural of the circuit, the delay of the logic gates is annotated into the simulation to verify the timing requirements.
- *Netlist after place & routing*: When the layout of the circuit has been generated, interconnection delays and more accurate timing information of the gates can be used to rerun again the test benches.

The test benches that have been employed are the same in every phase and are briefly explained in Section 4.1. Afterwards, in Sections 4.2 and 4.3 is explained how to apply a test bench in the three phases. The last section, Section 4.4, discuss the difference when testing the component *MyJtag*.

4.1 Test Benches

A test bench is a piece of HDL (mainly VHDL or Verilog) code that is used to verify the functional correctness of a HDL model. It can be seen as wrapper where the top entity of the design under test (DUT) is instantiated in order to apply stimulus to the DUT and verify the corresponding outputs. In this thesis, the DUT is a microcontroller, so, apart from generating stimulus for the inputs, a program should be load in the PM. How to generate this code is explained in Section 4.1.2 Regarding the use of Modelsim, we refer to [10] for all the details about how to use this tool for simulating a design, but there are two features that deserve to be stressed:

- *Loading the PM*: Loading instructions into the PM can be performed with the following instruction:

  mem load -infile "file.hex" -format hex "MemoryInstanceName"

  Where *file.hex* is the file containing the instructions in hexadecimal. This command is very useful, for instance, when the simulation has to be restarted.

- *Tcl/Tk scripting*: Modelsim supports Tcl/Tk scripting which allows us to save a significant amount of time when using the tool more than once. In general, several "simulation

```
vsim -t ps work.avr_wb_top_tbUART

#New "maximized" wave window
set newWindow [view wave -undock]
set waveTopLevel [winfo toplevel $newWindow]
wm geometry $waveTopLevel [winfo screenwidth .]x905+0+0

#Add all signals to wave
add wave -r /*

run 200ns

#Load the program into the PM
mem load -infile uartCcode.mem -format hex /
    /avr_wb_top_tbUART/dut/pm/progmem_inst/vitalbehavior/memorycore

run 800us
```

Figure 4.1: Modelsim script

+ correction" cycles have to be executed before obtaining the desired result, where most of the time some steps are repeated without any change.

The combination of these two features can be seen in Figure 4.1. In this example a simulation is executed just typing:

source simularUART

in the command window. Throughout this chapter the different steps to generate and apply a test bench to the design will be illustrated using one of the test cases explained in Section 4.1.1; we will use, for instance, the UART test case.

4.1.1 Test cases

Verifying that a circuit is 100% correct is almost impossible; consequently, some assumption must be taken in order to limit the number of cases to test. In this thesis, the microcontroller core and the USB wishbone slave are functionally correct and synthesisable for an FPGA environment. For that reason, it has been assumed to be sufficient the following test cases:

- *UART*: This test receives a value through the UART and retransmits again the same value multiplied by 3.
- *SPI*: The same test case as before is repeated, but now with the SPI.
- *MyJtag*: The PM is programmed with this component.
- *USB*: The USB wishbone slave is set and some data is sent.
- *Scan Chain*: Data has been written to and read from the *Scan Chain Wishbone slave* using the already mentioned scan chain.
- *I/O ports*: Some data is write to and read from the three 8-bits I/O ports.

4.1.2 Generating the program code

As have been said, for testing a microcontroller, it is necessary to generate the instructions that have to be load into the PM. Fortunately, there exists a complete tool chain for AVR microcontrollers that can be used, among other purposes, to generate the assembly code of the

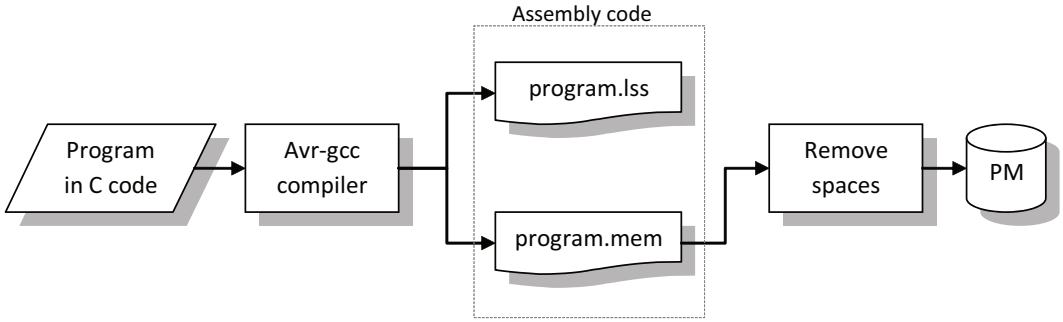


Figure 4.2: From C to Assembly code

program. One of these tools is the *avr-gcc*, a free-ware C compiler and assembler that allows us to use a high-level programming language like C to generate the assembly instructions as shown in Figure 4.2. The *avr-gcc* generates several files, the most relevant of which for us are the ".lss" and ".mem" files. The ".mem" file contains the operation codes and is used to load the program into the PM, while the ".lss" shows the assembly instructions mnemonics, which is really useful for debugging purposes. An example can be seen in Figure 4.3 for the UART test case. It is important to remember to delete the blank spaces in the ".mem" file because Modelsim does not complain about it, but inserts "00" in every place where it finds a blank space, creating two wrong instructions in place of only one correct instruction:

$$\text{“␣90␣0c”} \xrightarrow{To\ PM} \text{“0094” \& “000c”}$$

instead of:

$$\text{“940c”} \xrightarrow{To\ PM} \text{“940c”}$$

4.2 RTL Behavioural verification

According to Figure 2.2, at this point of the VLSI design flow, the correct behaviour of the circuit should be checked. In a normal procedure, every component should be tested separately before integrate them all together, but, as was stated in Section 4.1.1, it will be assumed that the microcontroller core (described in VHDL) and the USB wishbone slave (described in Verilog) are functionally correct. Regarding the other components, *MyJtag*, reset syncrhonizer, reset chain and the *Scan Chain Wishbone slave*, they have been included and tested one by one in the overall design. The first task to perform in order to simulate the DUT is to generate a "test bench file" in VHDL or Verilog[1], which provides the stimulus. The test bench file used for the UART test case can be seen in Figure 4.4[2]. In order to understand the simulation results in Figure 4.5a, the following has to be taken into account:

- The *UART* has been configured with a baud rate equal to 62500bps which yields in a symbol period of 16us.

[1]Modelsim supports multi-language simulation, consequently it is not a problem that the AVR core is described in VHDL and the USB in Verilog

[2]Only the last part of the file is shown (the stimulus).

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include "uartCcode.h"
#include "uart.h"

void
main (){
  init_uart ();       // Initialize UART
  sei();              // Enable interruptions
  char x,y;

  x = UART_receive();
  x = x*3;

  printf("%c",x);

  for(;;);
}

//Wait for a character
unsigned char UART_receive (void) {

  while ( !(USR & (1<<RXC))) ;

  return UDR;
}
```

(a)

```
000000a4 <UART_receive>:

//Wait for a character
unsigned char UART_receive (void) {

  while ( !(USR & (1<<RXC))) ;
  a4:   5f 9b        sbis    0x0b, 7    ; 11
                <UART_receive>
  a6:   fe cf        rjmp    .-4        ; 0xa4

  return UDR;
  a8:   8c b1        in   r24, 0x0c ; 12
}
  aa:   99 27        eor r25, r25
  ac:   08 95        ret

000000ae <main>:
                <init_uart>
  ae:   0e 94 71 01  call    0x2e2    ; 0x2e2
  b2:   78 94        sei
  b4:   5f 9b        sbis    0x0b, 7    ; 11
                <main+0x6>
  b6:   fe cf        rjmp    .-4        ; 0xb4
  b8:   8c b1        in   r24, 0x0c ; 12
  ba:   8f 5f        subi    r24, 0xFF ; 255
  bc:   99 27        eor r25, r25
                <putchar>
  be:   0e 94 d1 01  call    0x3a2    ; 0x3a2
                <main+0x14>
  c2:   ff cf        rjmp    .-2        ; 0xc2
```

(b)

Figure 4.3: UART C code (a) and part of the .lss file (b)

```
reset <= '1' after 0 ns,'0' after 200 ns;
sys_clk <= not sys_clk after 5 ns;   -- 100 MHz

--Sending a character to the uart at 62500bps (16us bit period)
rxdChar:process
begin
   rxd <= '1'; wait for 400 us;
   rxd <= '0'; wait for  16 us;
   rxd <= '1'; wait for  32 us;
   rxd <= '0'; wait for  32 us;
   rxd <= '1'; wait for  32 us;
   rxd <= '0'; wait for  32 us;
   rxd <= '1'; wait;
end process;
```

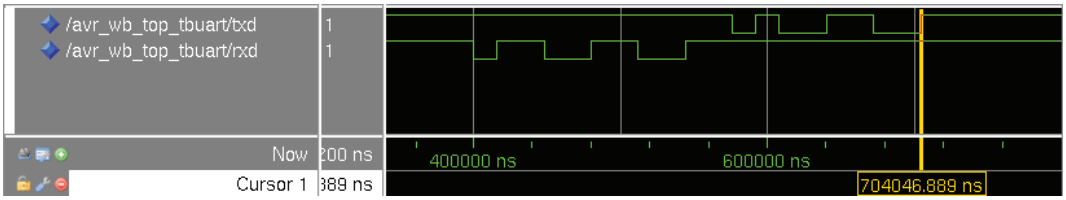
Figure 4.4: Stimilus example in a test bench

- The microcontroller is waiting for a character (33 in hexadecimal in the example) that is going to be retransmitted multiplied by 3.
- The frame format is configured with a starting bit (always low) and one stop bit (always high). Furthermore, the transmission starts with the LSB (less significant bit); for that reason, when transmitting a 33 to the *UART* the "rxd" line shows the following binary sequence:

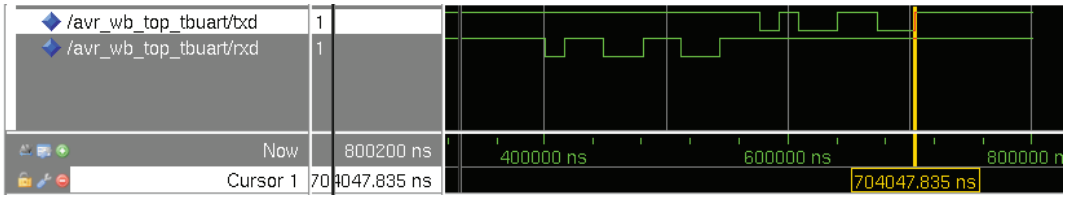
$$33 \rightarrow \overbrace{0}^{start}\ \underbrace{11001100}_{\text{data}}\ \overbrace{1}^{stop}$$

and in the same way, the expected 99 will be:

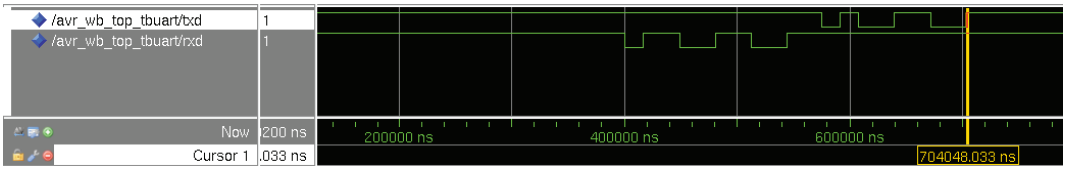
$$99 \rightarrow \overbrace{0}^{start}\ \underbrace{11001100}_{\text{data}}\ \overbrace{1}^{stop}$$



(a) RTL simulation



(b) Post-synthesis simulation



(c) Post-layout simulation

Figure 4.5: Simulation of the UART

4.3 Synthesised netlist and post-layout netlist verification

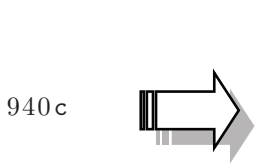
The simulation procedure for the synthesised and post-layout netlist is similar to the RTL simulation, but now we have the possibility of including the cell delays and, in the post-layout simulation, the interconnection delays. In order to perform an annotated simulation the SDF files generated by Design Compiler and SOC Encounter (see Section 5.6 and Section 6.7) have to be read into Modelsim. This can be done by adding the following options to the first line of the script presented in Figure 4.1:

vsim -sdftyp "topInstanceNam"="pathToSDFfile" -sdfnoerror ...

Furthermore, the following replacements in the SDF file generated by SOC Encounter have to be done:

RB Q ()	→	(negedge RB) Q ()
SB Q ()	→	(negedge SB) Q ()
RB QB	→	(negedge RB) QB
SB QB	→	(negedge SB) QB
SB Q	→	(negedge SB) Q

otherwise, the timing information related with the "set" and "reset" signal will not be annotated. This is due to some differences in the syntax of the SDF generated by SOC Encounter. For further information about SDF syntax check [10].




```
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '1'; wait for 130 ns;
rxd <= '1'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '1'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '1'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '0'; wait for 130 ns;
rxd <= '1'; wait for 130 ns;
```

Figure 4.6: Opearation code from hexadecimal to VHDL signals

Figure 4.5 shows the repetition of the test bench performed in the previous section. It can be seen the extra delay due to the cell delays in Figure 4.5b and some additional delay in Figure 4.5c due to the interconnections.

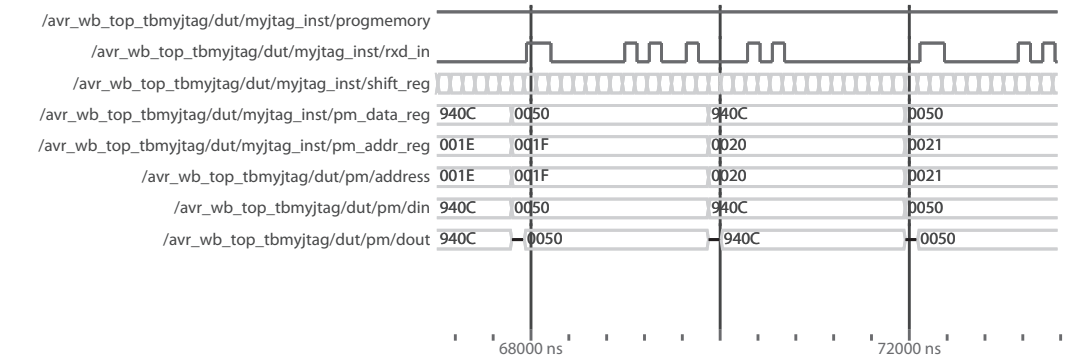
4.4 Testing the component MyJtag

Unlike the other test cases, in this one the PM cannot be loaded as before, because that is the purpose of this component. Therefore, in order to test *MyJtag*, a small application has been developed in C. It receives a file containing the instruction codes in hexadecimal and returns the corresponding VHDL file to test this component. An example of the input and output files can be seen in Figure 4.6. According to this, to prove that this additional programming capability works properly, the other test cases will be loaded into the PM using this method. For instance, the transmission of a character with the SPI. Figure 4.7b shows how the data is being loaded into the PM and in Figure 4.7a it can be seen that everything is working as expected.

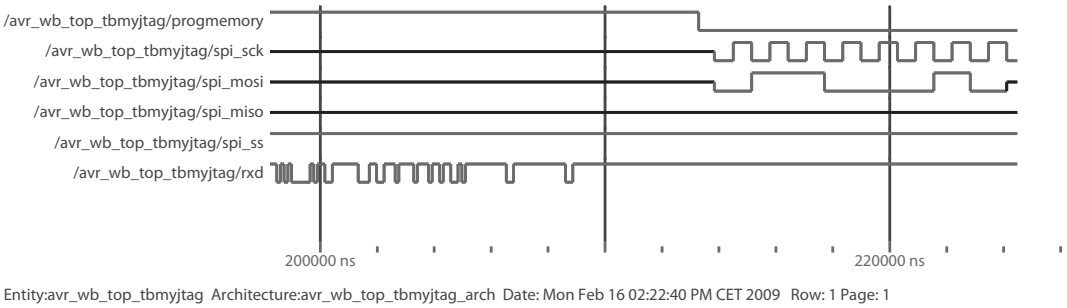
4.5 Summary

Throughout this chapter the test cases applied to design have been described. It has been discussed also how to generate this test cases and the differences in the different phases of the design flow (RTL behavioural description, post-synthesis and post-layout). Finally, the method employed to test the component *MyJtag* was explained.

In the following chapter will be discussed how to synthesize the design using Design Compiler and some solutions to increase the clock frequency and reduce the power.



(a) Loading data into the PM



(b) Transmitting a character

Figure 4.7: MyJtag test

Synthesis with Design Compiler

5

In this chapter will be discussed some issues related with performing the design synthesis step using Design Compiler. A nice definition of synthesis can be found on [11]:

> *"Synthesis is the process of taking a design written in a hardware description language, such as VHDL, and compiling it into a netlist of interconnected gates which are selected from a user-provided library of various gates."*

5.1 Constraining the design

Among the numerous issues that arises during the synthesis of a design using Design Compiler, the most important ones for the scope of this thesis have been selected and will be detailed in the following subsections.

5.1.1 Compile strategy

In order to obtain the best results from Design Compiler, a two-pass compilation strategy should be apply. First, the design has to be compiled with the appropriate options depending on the requirements. In this thesis the timing requirements are more restrictive than the area requirements, therefore an example of a first compilation could be:

compile_ultra -timing[1]

Then, the design can be optimized using:

compile_ultra -incremental

5.1.2 Input delay

The synchronizers used in the general digital I/O ports are triggered by a falling edge. If Design Compiler is not aware of this fact, it will assign an input delay of half of the clock period to that pins. To avoid this situation, it has to be specified the maximum input delay and that the delay is relative to the falling edge of the clock:

set_input_delay -clock "clkName" -max "delay" -clock_fall ["listOfports"]

5.1.3 Special buffers and inverters for the clock

Normal buffers and inverters have smaller area than the specific ones for the clock, but the second ones are better for balancing and optimizing the clock tree as can be seen in the delay comparison shown in Table 5.1. According to this, clock buffers and inverters should not be used during optimization of the design. This can be achieve using the following command before the incremental compilation:

[1] "Compile_ultra" is only available with DC Ultra license

	Load	1.200 ff		3.012 ff		7.560 ff		18.97 ff		47.62 ff		119.5 ff	
	Path	tplh	tphl	tplh	tphl	tplh	tphl	tplh	tphl	tplh	tphl	tplh	tphl
BUFX1	I-O	29.80	42.98	36.69	47.73	53.27	56.32	94.54	74.18	197.5	114.6	455.5	214.0
BUFCKX1	I-O	31.92	35.61	37.26	41.17	49.22	53.41	77.82	82.09	148.8	153.1	327.2	330.9

Table 5.1: Normal and clock buffer delays (ps)

set_dont_use "libname"/"cell_list"

5.1.4 Generated clocks

In Section 5.4 will be explained that in some situations, a clock divider can be used in this design. This fact has to be specified in Design Compiler; otherwise, the generated clock will be treated as a normal data signal. For instance, a divided by two clock can be specified by:

create_generated_clock -divide_by 2 -source "source_clock" [get_pins "root_pin"]

5.2 Synchronous and asynchronous reset

This section is not intended to discuss the advantages and disadvantages of synchronous and asynchronous resets in a design as its name suggests (for that purpose, check [4]). In its place, we will focus on the design issues that involved the synthesis of both kinds of resets, and subsequently in Section 6.5 the "reset tree generation" procedure with *SOC Encounter* will be described.

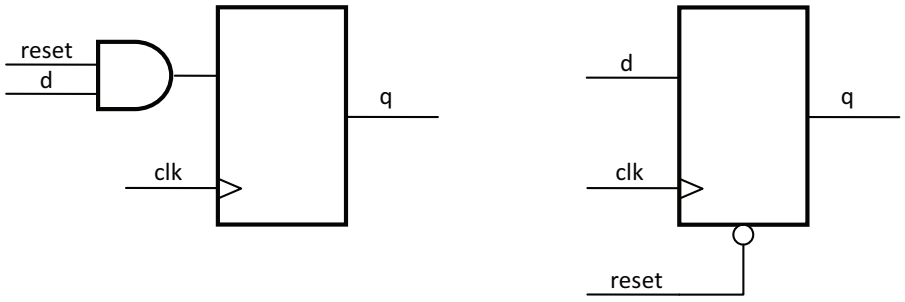
5.2.1 Asynchronous reset

In this thesis, the whole design is using an asynchronous reset with the exception of the *SPI* and the *USB Wishbone slave*. The main advantage of the asynchronous reset is that not interfere in the data path, i.e., no logic is inserted in the data path due to the reset. When trying to synthesize a design with asynchronous reset, it is necessary to be aware of the following matters:

- *Metastability*: This issue was already discussed in Section 3.2.1 and a solution was also proposed.
- *DFT insertion*: If the design is implemented with any of the techniques explained in Section 2.5, the reset of the components involved in the DFT should be controlled directly from an I/O pin, i.e., the reset cannot pass through the required *Reset Synchronizer*. In the present design, only the *Scan Chain Wishbone slave* has this problem, that can be easily solved adding a multiplexer controlled for the "scan enable" signal. Nevertheless, due to the extra risk that means additional logic in the reset tree and the fact that the scan chain is just a prototype, the reset to this slave is directly driven from the I/O pin reset at the expense of possible metastability.

5.2.2 Synchronous reset

As have been said, the *SPI* and the *USB Wishbone slave* have been implemented using a synchronous reset. This kind of reset can be seen as a normal data signal that can be implemented as depicted in Figure 5.1a. This implementation has two disadvantages:



(a) (b)

Figure 5.1: Flip-flop without(a) and with(b) built-in synchronous reset

- Extra logic in the data path: The extra delay introduced by the AND gate can be serious when is part of the critical path. It has to be taken into account that this situation can occurred several times throughout the critical path, multiplying the number of times by the gate delay.
- Potential problems in simulation: Depending on the implementation of the reset, it can be masked by 'X's during simulation, which makes impossible the verification of the design. There are many ESNUG articles (E-mail Synopsys User Group) about this question.

Fortunately, the standard cell library from Faraday has flip-flops with built-in synchronous reset as depicted in Figure 5.1b, which allows to avoid the previous problems. Assuming a good coding style for synthesis, Design Compiler automatically recognizes a synchronous reset and infers the appropriate logic, but sometimes (as is the case) it is necessary to specify with some compiler directives that a synchronous reset is being used. For a Verilog design, the following directive should be included in every module[2] where a synchronous reset is present:

//synopsys sync_set_reset "reset_signal_name"

In a VHDL design the procedure is basically the same. The only differences are that instead of "directive", in the Presto VHDL compiler it is called "attribute", and that it has to be specified at the beginning of the file that that attribute is going to be used. The attribute has to be included in the entity part, after the port declaration as shown in Figure 5.2. Figure 5.3 illustrates two different implementation of part of a design with synchronous reset.

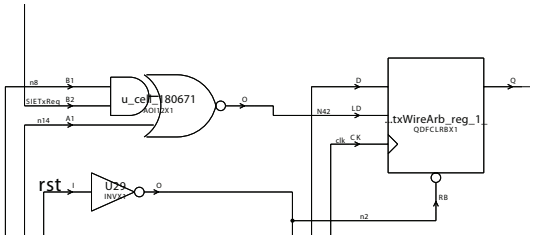
5.2.2.1 Correcting the USB reset

In the last section has been stated that, most of the times, Design Compiler detects the synchronous reset when dealing with proper synthesizable code. What have not been said is that a good synthesizable code is sometimes mandatory to obtain a correct synchronous

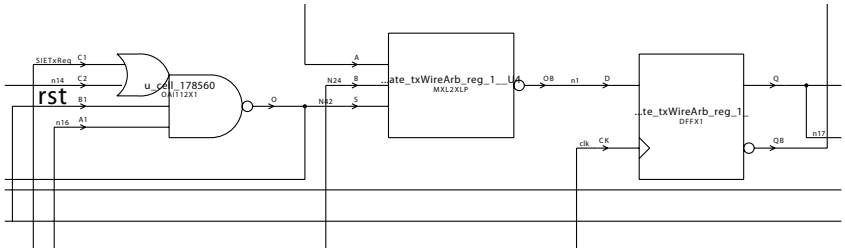
[2]This is important because this directive does not propagate through the hierarchy.

```
library synopsys;
use synopsys.attributes.all;

entity ... is
   port (
      ...
   );
   attribute sync_set_reset of "reset_signal_name": signal is "true"
end ...;
```

Figure 5.2: Sync_set_reset VHDL attribute



(a) Flip-flop with built-in synchronous reset



(b) Synchronous reset implemented with logic

Figure 5.3: Different synchronous reset implementations

reset implementation, even using the directive (or attribute in VHDL language) explained before. In order to accomplish with this premise, some changes had to be made throughout the reset path in the *USB Wishbone slave* as depicted in Figure 5.4. Furthermore, some registers were not correctly initialized and it was necessary to include them into the initial reset state, otherwise, the USB would not operate properly[3].

5.3 Scan Chain insertion

In Section 2.5 an overview about some DFT techniques and their necessity in actual designs was given. This section will focus in the use of Design Compiler to implement a scan chain with the multiplexed flip-flop style. Figure 5.5 shows the DFT insertion flow that is going to be followed. Nevertheless, the insertion procedure is intended to minimize the impact in the

[3]After reading this, a question immediately arises: how was then the USB able to operate correctly in a FPGA? Because an FPGA has a general reset that initialize the whole board, what makes the FPGA immune from this problem

```
always @(posedge busClk) begin

  if (rstFromWire == 1'b1 || rstFromBus == 1'b1)
    rstShift <= 6'b111111;
  else
    rstShift <= {1'b0, rstShift[5:1]};

end
```

Original

```
always @(posedge busClk) begin
  if (rstFromWire == 1'b1)
    rstShift <= 6'b111111;
  else begin
    if (rstFromBus == 1'b1)
    rstShift <= 6'b111111;
    else
       rstShift <= {1'b0, rstShift
            [5:1]};
  end
end
```

Corrected

Figure 5.4: Correction of the USB Verilog code for synthesis

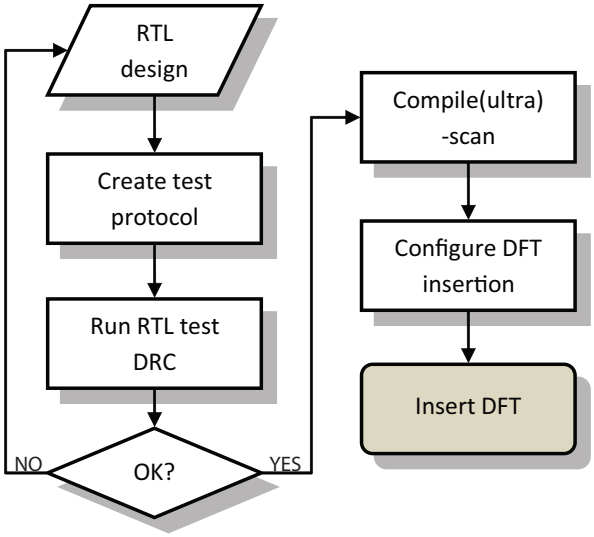


Figure 5.5: DFT insertion flow

rest of the design, i.e., due to the prototype nature of the scan chain, the insertion has to be as "clean" as possible.

5.3.1 Create test protocol

The first task to perform is to define the test protocol, i.e., the signals that are going to be used in the DFT have to be specified. When specifying a DFT signal, there are two options: use an existing signal as a DFT signal or define a new one that Design Compiler will create automatically while performing the DFT insertion. The problem with the second option is that it becomes more difficult to attach an I/O cell to the new signal; therefore, the easiest way is to define in advance the necessary signals with the corresponding I/O cells in the top level file of the design. The specification of the DFT signals has to be done just before to perform the first compilation, i.e., after creating the clock, setting the design constraints, etc. In the multiplexed flip-flop style, the required signals are the following:

- *Clock*
- *Reset*
- *Scan data input*

- *Scan data output*
- *Scan enable*

An example of how to specify a scan enable signal that is active high is:

set_dft_signal -view existing_dft -type ScanEnable -port "signal_name" -active_state 1

Next, it is necessary to indicate the elements that are going to be part of the scan chain:

set_scan_element true {*"element_list"*}

Finally, the test protocol has to be created:

create_test_protocol

Once the signals and the elements that are going to be used are correctly specified in the corresponding test protocol, we can proceed with the next step.

5.3.2 RTL Design Rule Checking

The Design Rule Checking (DRC) process checks the design to determine if there is any design rule violation in the design (see [9] for further information about the test design rules). The DRC process can be invoked just by typing:

dft_drc

When the process finishes , the "Violation Browser" window appears showing the detected violations. In the present design, the following violation were found:

- *DFF set/reset line not controlled*: This violation arise because the reset is not driven from an I/O pin. This issue was already discussed in Section 5.2.2.
- *Bus gate failed contention ability check*: Due to the shared bus interconnection that is being used for the Wishbone interface, bus contention can occur if both of the slaves try to communicate with the wishbone interface at the same time. As can be seen in Figure 5.6, there is a case (*when => "111")* where all the slaves are enabled. Therefore, this violation can be solved just by removing that case, but it can also be neglected without affecting the DFT performance.
- *Bus gate failed 'Z' state ability check*: The situation in this case is similar to the previous one. The DRC process is complaining because there is no way to force a 'Z' state in the bus being shared by the *SPI*, the *EEPROM Interface* and the *Wishbone Interface*. This violation, as the previous one, is not a problem and it can be also neglected.

```
--Slave selection
case wbctrl (2 downto 0) is
  when ``00" =>
    wb_enable_o <= ``0000001";
  when ``01" =>
    wb_enable_o <= ``0000010";
  when ``10" =>
    wb_enable_o <= ``0000100";
  when ``11" =>
    wb_enable_o <= ``0001000";
  when ``00" =>
    wb_enable_o <= ``0010000";
  when ``01" =>
    wb_enable_o <= ``0100000";
  when ``10" =>
    wb_enable_o <= ``1000000";
  when ``11" =>
    wb_enable_o <= ``1111111";
  when others =>
    wb_enable_o <= ``0000000";
end case;
```

Figure 5.6: RTL code that generates DRC DFT violation

5.3.3 Compile, configure and insert the DFT

Once the violations have been corrected (or ignore if possible), the design must be compile including the "-scan" option among the compile parameters. This option tells Design Compiler to replace the elements being part of the DFT for scannable elements. After that, the last step before inserting the scan chain is to configure the way that it is going to be inserted. Among the multiple options that are detailed in [9], the most relevants for our design are:

- Three-state buffers and bidirectional ports: By default, DFT Compiler infers logic to bring all the three-state buffers and bidirectional ports into 'Z' state. This behaviour is not necessary in our design, and consequently, has to be deactivated:

  set_dft_configuration -fix disable -fix_bidirectional disable

- Ordered scan chain: The order of the scan chain can be defined with the following command:

  set_scan_path "chainname" { "ordered_list"}

Finally, the insertion of the scan chain is executed with:

insert_dft

A scan segment can be seen in Figure 5.7

5.4 Increasing the clock frequency

The initial design was intended to run at 50MHz in order to have only one external clock[4] in the design, due to the fact that the internal clock of the *USB Wishbone slave* has to run at 50MHz[5] (see Section 3.2.4 for a brief explanation about the USB clocks). Therefore, the

[4]The JTAG clock and the external clock used by the component *MyJtag* are not taking into account in this discussion.

[5]Being more precisely, the USB internal clock has to run at 48MHz, but for simplicity we will keep the discussion with 50MHz and multiples. Besides, if the design can run at 50 or 100MHz, it can also do it at 48 or 96.

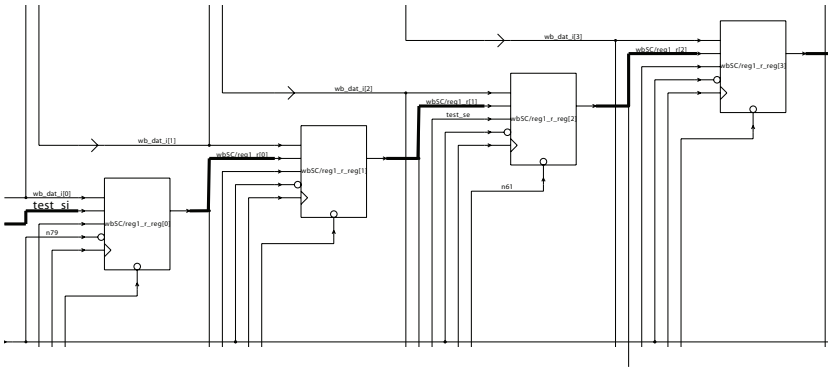


Figure 5.7: Initial part of the scan chain

first synthesis attempt was made with a clock of 50MHz, resulting in the critical path of Figure 5.8. This critical path shows a significant improvement margin, but, before starting to increase the frequency, we should take a look to the overall design to identify the factors that limit the clock speed apart from, obviously, the critical path:

- *I/O cell*: The standard cell library used in this thesis does not provide specific I/O cells for the clock. The generic I/O cell employed almost has 1ns delay in the worst case, which limits the frequency to approximately 1GHz.
- *Memories*: In Table 2.1 were shown the timing values for the PM, which is the most restrictive of the memories used in terms of timing. It can be seen that the "Read cycle time" in the worst case (WC) situation is 3.21ns, which yields in a maximum frequency slightly higher than 300MHz.
- *USB*: As explained in Section 3.2.4, the "wishbone clock" of the *USB Wishbone slave* has to run between 24 and 240MHz.

According to this, none of these elements are, for the time being, an obstacle when trying to increase the speed of the circuit. On the other hand, keeping the intention of having only one external clock determine the next attempt to 100MHz with the use of a clock divider for the USB internal clock (see Section 5.1.4). The resulting critical path can be seen in Figure 5.9, showing that the limitation for further increases in frequency is due to the fact that the memories are being triggered with an inverted clock.

Before trying to find a solution in order to jump now into 150MHz, we should consider again the constraint of having only one external clock. This restriction limits the possible frequencies to multiples of fifty; it could be the case that the microcontrollers could work at 140MHz, but not at 150MHz, and therefore the frequency should be kept to 100MHz, throwing away a potential improvement of 40%. Consequently, at the expense of an additional clock, this constraint will be removed in order to have "scale down" possibilities concerning the speed of the microcontroller.

Coming back to the problem of Figure 5.9, it is clear that the only reason for having an inverted clock triggering the memories is to accomplish with "setup times" requirements. Table 2.1 shows that the most restrictive setup value is 0.25ns, which is much smaller than half of the clock period (5ns at the moment). For that reason, two solutions are proposed:

1. Use a double speed inverted clock for the memories.

```
Startpoint: DM/Mem_Inst
            (rising edge-triggered flip-flop clocked by sys_clk_pad')
Endpoint: AVR_core/AVR/main/pc_low_reg_1_
          (rising edge-triggered flip-flop clocked by sys_clk_pad)
Path Group: sys_clk_pad
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------------
clock sys_clk_pad' (rise edge)                          10.00      10.00
clock network delay (ideal)                              0.00      10.00
DM/Mem_Inst/CK (SHAA90_8192X8X1CM8)                      0.00      10.00 r
DM/Mem_Inst/DO3 (SHAA90_8192X8X1CM8)                     2.73      12.73 f
DM/dout[3] (DataRAM)                                     0.00      12.73 f
AVR_core/RAMD_OUT[3] (avr_wb)                            0.00      12.73 f
AVR_core/EXT_MUX/ram_data_out[3] (external_mux)          0.00      12.73 f
AVR_core/EXT_MUX/U17/O (AOI22X1)                         0.12      12.84 r
AVR_core/EXT_MUX/U52/O (OAI12X1)                         0.07      12.92 f
AVR_core/EXT_MUX/dbus_out[3] (external_mux)              0.00      12.92 f
AVR_core/AVR/dbusin[3] (AVR_Core)                        0.00      12.92 f
AVR_core/AVR/io_dec/dbusin_ext[3] (io_adr_dec)           0.00      12.92 f
AVR_core/AVR/io_dec/U20/O (INVX1)                        0.04      12.96 r
AVR_core/AVR/io_dec/U17/O (OAI112X1)                     0.16      13.12 f
AVR_core/AVR/io_dec/dbusin_int[3] (io_adr_dec)           0.00      13.12 f
AVR_core/AVR/BP_Inst/dbusin[3] (bit_processor)           0.00      13.12 f
AVR_core/AVR/BP_Inst/U43/O (AOI222X1)                    0.24      13.36 r
AVR_core/AVR/BP_Inst/U42/O (NR2X1)                       0.05      13.41 f
AVR_core/AVR/BP_Inst/U41/O (AOI13X1)                     0.09      13.50 r
AVR_core/AVR/BP_Inst/U40/O (OAI112X1)                    0.12      13.62 f
AVR_core/AVR/BP_Inst/U39/O (INVX1)                       0.06      13.67 r
AVR_core/AVR/BP_Inst/U38/O (OAI23X1)                     0.08      13.75 f
AVR_core/AVR/BP_Inst/U25/O (OAI112X1)                    0.10      13.86 r
AVR_core/AVR/BP_Inst/bit_test_op_out (bit_processor)
                                                         0.00      13.86 r
AVR_core/AVR/main/bit_test_op_out (pm_fetch_dec)         0.00      13.86 r
AVR_core/AVR/main/U461/O (ND3X1)                         0.13      13.99 f
AVR_core/AVR/main/U700/O (INVX1)                         0.38      14.36 r
AVR_core/AVR/main/u_cell_177816/O (NR3X1)                0.07      14.43 f
AVR_core/AVR/main/u_cell_177817/O (OAI112X1)             0.37      14.80 r
AVR_core/AVR/main/U210/O (INVX1)                         0.18      14.98 f
AVR_core/AVR/main/U557/O (AOI22XLP)                      0.17      15.14 r
AVR_core/AVR/main/U509/O (OAI112X1)                      0.08      15.22 f
AVR_core/AVR/main/pc_low_reg_1_/D (QDFERBX1)             0.00      15.22 f
data arrival time                                                  15.22

clock sys_clk_pad (rise edge)                           20.00      20.00
clock network delay (ideal)                              0.00      20.00
clock uncertainty                                       -0.20      19.80
AVR_core/AVR/main/pc_low_reg_1_/CK (QDFERBX1)            0.00      19.80 r
library setup time                                      -0.18      19.62
data required time                                                 19.62
-----------------------------------------------------------------------------
data required time                                                 19.62
data arrival time                                                 -15.22
-----------------------------------------------------------------------------
slack (MET)                                                        4.40
```

Figure 5.8: Critical path at 50MHz

2. Introduce a fixed delay in the clock wire with some delay cells.

The first solution has the advantage of providing an accurate "delay" of a quarter of the clock period, but limits the frequency to 150MHz (the maximum frequency for the memories

```
Startpoint: DM/Mem_Inst
            (rising edge-triggered flip-flop clocked by sys_clk_pad')
Endpoint: AVR_core/AVR_main/pc_low_reg_2_
          (rising edge-triggered flip-flop clocked by sys_clk_pad)
Path Group: sys_clk_pad
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------------
clock sys_clk_pad' (rise edge)                           5.00       5.00
clock network delay (ideal)                              0.00       5.00
DM/Mem_Inst/CK (SHAA90_8192X8X1CM8)                      0.00       5.00 r
DM/Mem_Inst/DO6 (SHAA90_8192X8X1CM8)                     2.73       7.73 f
DM/dout[6] (DataRAM)                                     0.00       7.73 f
AVR_core/RAMD_OUT[6] (avr_wb)                            0.00       7.73 f
AVR_core/EXT_MUX/ram_data_out[6] (external_mux)          0.00       7.73 f
AVR_core/EXT_MUX/U29/O (AOI22X1)                         0.12       7.84 r
AVR_core/EXT_MUX/U28/O (OAI12X1)                         0.07       7.92 f
AVR_core/EXT_MUX/dbus_out[6] (external_mux)              0.00       7.92 f
AVR_core/AVR_io_dec/dbusin_ext[6] (io_adr_dec)           0.00       7.92 f
AVR_core/AVR_io_dec/U8/O (INVX1)                         0.04       7.96 r
AVR_core/AVR_io_dec/U1/O (OAI112X1)                      0.19       8.15 f
AVR_core/AVR_io_dec/dbusin_int[6] (io_adr_dec)           0.00       8.15 f
AVR_core/AVR_BP_Inst/dbusin[6] (bit_processor)           0.00       8.15 f
AVR_core/AVR_BP_Inst/U21/O (AOI222X1)                    0.19       8.34 r
AVR_core/AVR_BP_Inst/U42/O (NR2X1)                       0.05       8.39 f
AVR_core/AVR_BP_Inst/U41/O (AOI13X1)                     0.10       8.49 r
AVR_core/AVR_BP_Inst/U73/O (OAI112X2)                    0.10       8.59 f
AVR_core/AVR_BP_Inst/U39/O (INVX1)                       0.05       8.64 r
AVR_core/AVR_BP_Inst/U38/O (OAI23X1)                     0.08       8.72 f
AVR_core/AVR_BP_Inst/U22/O (OAI112X1)                    0.13       8.85 r
AVR_core/AVR_BP_Inst/bit_test_op_out (bit_processor)
                                                         0.00       8.85 r
AVR_core/AVR_main/bit_test_op_out (pm_fetch_dec)         0.00       8.85 r
AVR_core/AVR_main/U461/O (ND3X3)                         0.10       8.95 f
AVR_core/AVR_main/U193/O (INVX4)                         0.12       9.08 r
AVR_core/AVR_main/U4/O (NR3X1)                           0.05       9.13 f
AVR_core/AVR_main/u_cell_178846/O (OAI112X1)             0.11       9.23 r
AVR_core/AVR_main/U186/O (BUFX3)                         0.13       9.37 r
AVR_core/AVR_main/U80/O (INVX4)                          0.04       9.41 f
AVR_core/AVR_main/U664/O (AOI22XLP)                      0.13       9.54 r
AVR_core/AVR_main/U660/O (OAI112X1)                      0.08       9.62 f
AVR_core/AVR_main/pc_low_reg_2_/D (QDFERBX1)             0.00       9.62 f
data arrival time                                                   9.62

clock sys_clk_pad (rise edge)                           10.00      10.00
clock network delay (ideal)                              0.00      10.00
clock uncertainty                                       -0.20       9.80
AVR_core/AVR_main/pc_low_reg_2_/CK (QDFERBX1)            0.00       9.80 r
library setup time                                      -0.17       9.63
data required time                                                  9.63
-----------------------------------------------------------------------------
data required time                                                  9.63
data arrival time                                                  -9.62
-----------------------------------------------------------------------------
slack (MET)                                                         0.00
```

Figure 5.9: Critical path at 100MHz

is 300 MHz) and a clock divider has to be used to generate the rest of the clock tree. On the other hand, using some delay cells from the Faraday library, a more optimal delay can be

```
Startpoint: AVR_core/AVR/main/adiw_st_reg
            (rising edge-triggered flip-flop clocked by sys_clk_pad)
Endpoint: AVR_core/AVR/main/pc_low_reg_1_
          (rising edge-triggered flip-flop clocked by sys_clk_pad)
Path Group: sys_clk_pad
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------------
clock sys_clk_pad (rise edge)                            0.00       0.00
clock network delay (ideal)                              0.00       0.00
AVR_core/AVR/main/adiw_st_reg/CK (QDFFRBX1)              0.00       0.00 r
AVR_core/AVR/main/adiw_st_reg/Q (QDFFRBX1)               0.24       0.24 f
AVR_core/AVR/main/U165/O (NR2X1)                         0.19       0.43 r
AVR_core/AVR/main/U299/O (AN4B1XLP)                      0.15       0.58 r
AVR_core/AVR/main/U146/O (AN3X1)                         0.15       0.73 r
AVR_core/AVR/main/U657/O (INVX3)                         0.05       0.77 f
AVR_core/AVR/main/U225/O (NR2X1)                         0.23       1.00 r
AVR_core/AVR/main/U649/O (ND2X1)                         0.13       1.13 f
AVR_core/AVR/main/U147/O (NR2X1)                         0.11       1.25 r
AVR_core/AVR/main/U148/O (INVX1)                         0.06       1.30 f
AVR_core/AVR/main/U187/O (NR2X1)                         0.31       1.61 r
AVR_core/AVR/main/u_cell_159069/O (AOI22XLP)             0.13       1.75 f
AVR_core/AVR/main/u_cell_159071/O (OAI112X1)             0.12       1.86 r
AVR_core/AVR/main/adr[0] (pm_fetch_dec)                  0.00       1.86 r
AVR_core/AVR/U1/O (BUFX1)                                0.60       2.46 r
AVR_core/AVR/adr[0] (AVR_Core)                           0.00       2.46 r
AVR_core/WishboneInterface/adr[0] (wishbone_interface)
                                                         0.00       2.46 r
AVR_core/WishboneInterface/u_cell_159051/O (NR3X1)       0.10       2.56 f
AVR_core/WishboneInterface/U10/O (OR4B2XLP)              0.47       3.03 r
AVR_core/WishboneInterface/U16/O (AOI12X1)               0.17       3.20 f
AVR_core/WishboneInterface/out_en (wishbone_interface)
                                                         0.00       3.20 f
AVR_core/EXT_MUX/io_port_en_bus_7_ (external_mux)        0.00       3.20 f
AVR_core/EXT_MUX/U96/O (NR2X1)                           0.12       3.33 r
AVR_core/EXT_MUX/U95/O (OR3B1XLP)                        0.11       3.44 f
AVR_core/EXT_MUX/U87/O (NR2X1)                           0.12       3.56 r
AVR_core/EXT_MUX/U84/O (AN3B1X1)                         0.46       4.02 r
AVR_core/EXT_MUX/U56/O (AOI22X1)                         0.13       4.16 f
AVR_core/EXT_MUX/U55/O (OAI112X1)                        0.11       4.26 r
AVR_core/EXT_MUX/U21/O (AOI112X1)                        0.06       4.33 f
AVR_core/EXT_MUX/U52/O (OAI12X1)                         0.12       4.44 r
AVR_core/EXT_MUX/dbus_out[3] (external_mux)              0.00       4.44 r
AVR_core/AVR/dbusin[3] (AVR_Core)                        0.00       4.44 r
AVR_core/AVR/io_dec/dbusin_ext[3] (io_adr_dec)           0.00       4.44 r
AVR_core/AVR/io_dec/u_cell_158926/O (INVX1)              0.04       4.48 f
AVR_core/AVR/io_dec/u_cell_158929/O (OAI112X1)           0.25       4.74 r
AVR_core/AVR/io_dec/dbusin_int[3] (io_adr_dec)           0.00       4.74 r
AVR_core/AVR/BP_Inst/dbusin[3] (bit_processor)           0.00       4.74 r
AVR_core/AVR/BP_Inst/U43/O (AOI222X1)                    0.16       4.89 f
AVR_core/AVR/BP_Inst/U42/O (NR2X1)                       0.09       4.98 r
AVR_core/AVR/BP_Inst/U41/O (AOI13X1)                     0.05       5.03 f
AVR_core/AVR/BP_Inst/U40/O (OAI112X1)                    0.14       5.17 r
AVR_core/AVR/BP_Inst/U38/O (OAI23X1)                     0.12       5.29 f
AVR_core/AVR/BP_Inst/U25/O (OAI112X1)                    0.12       5.41 r
AVR_core/AVR/BP_Inst/bit_test_op_out (bit_processor)
                                                         0.00       5.41 r
AVR_core/AVR/main/bit_test_op_out (pm_fetch_dec)         0.00       5.41 r
AVR_core/AVR/main/U466/O (ND3X2)                         0.10       5.51 f
AVR_core/AVR/main/U259/O (INVX1)                         0.36       5.87 r
AVR_core/AVR/main/U226/O (OR3X1)                         0.38       6.25 r
AVR_core/AVR/main/U215/O (INVX1)                         0.13       6.38 f
AVR_core/AVR/main/U214/O (AOI22XLP)                      0.16       6.53 r
AVR_core/AVR/main/U212/O (OAI112X1)                      0.08       6.62 f
AVR_core/AVR/main/pc_low_reg_1_/D (QDFERBX1)             0.00       6.62 f
data arrival time                                                   6.62

clock sys_clk_pad (rise edge)                            7.00       7.00
clock network delay (ideal)                              0.00       7.00
clock uncertainty                                       -0.20       6.80
AVR_core/AVR/main/pc_low_reg_1_/CK (QDFERBX1)            0.00       6.80 r
library setup time                                      -0.18       6.62
data required time                                                  6.62
-----------------------------------------------------------------------------
data required time                                                  6.62
data arrival time                                                  -6.62
-----------------------------------------------------------------------------
slack (MET)                                                         0.01
```

Figure 5.10: Critical path at 142MHz

	f_{ck} = 100MHz	f_{ck} =200MHz
$f_{ck}/32$	3.125M	6.25M
$f_{ck}/64$	1.5M	3.125M
$f_{ck}/128$	781k	1.5M
$f_{ck}/256$	390k	781k

Table 5.2: Baud rates SPI

achieved[6]. The drawback of this second solution is the inaccuracy introduced in the reset tree by the delay cells; nevertheless, placing different cells delays just in front of the clock port of every memory minimize this inaccuracy to the point that can be neglected for our purposes. Therefore, the final solution will consist in three delay cells attached just in front of every clock port for the three memories. If the synthesis of the design is performed again with the proposed solution and a clock period of 7ns (142MHz), the critical path show in Figure 5.10, which meets the requirements, will be achieved.

The next step in order to reach the maximum clock frequency of the design is to use a back-annotated synthesis approach. As explained in Chapter Section 7, back-annotated synthesis uses the interconnection delays and more accurate timing information of the gates generated for the SOC Encounter. Therefore, it is considered more appropriated to postpone this discussion to Chapter Section 7.

5.4.1 Higher frequency consequences

Higher clock speeds are desired most of the time. The only reason to keep the clock speed under its maximum is in order to save power. In Table 5.4 can be seen the power consumption for 50, 100 and 200 MHz estimated by Design Compiler.

On the other hand, the advantages are clear: faster processing capacity and higher transmission rates with external devices using the SPI or the UART. Table 5.2 shows the different transmission rates for the SPI and in Table 5.3 for the UART. In the second one, the UBRR column ("Uart baud rate register") refers to value that has to be written in that register to achieve the corresponding baud rate. The "%Error" refers to the difference between the real baud rate and the one shown in the table.

5.5 Clock gating

Clock gating is a powerful power-saving technique that consists in disabling parts of the circuit that are not being used by "pruning" the clock tree through some additional logic. Design Compiler perform this technique automatically just by including *-gate_clock* among the compile options in any (or both) of the two compile passes explained in Section 5.1.1[7].

Table 5.4 shows the results of applying clock gating to the present design. As can be seen, the power reduction in the AVR core is between 50 and 60% depending on the frequency[8].

[6] A quarter of the clock period can be still a really conservative setup time for the memories.

[7] Version Z-2007.03-SP4 of Design Compiler crashes if the -gate_clock option is active in both of the compilations passes. This problem has been solved in version B-2008.09-SP1-1

[8] The reduction in the USB is in the same order as the AVR core if the 10 memories implementing the FIFOs are not taken into account for the power consumption

	100 MHz		200 MHz	
Baud rates	UBRR	%Error	UBRR	%Error
28800	216	0.00	-	-
38400	162	-0.15	-	-
57600	108	-0.45	216	0.00
76800	80	0.47	162	-0.15
115200	53	0.47	108	-0.45
250k	24	0.00	49	0.00
0.5M	12	-3.85	24	0.00
1M	5	4.17	12	-3.85
2M	2	4.17	5	4.17
6.25M	0	0.00	1	0.00
12.5M	-	-	0	0.00

Table 5.3: Baud rates UART

	50MHz		100MHz		200MHz	
	Gated Clock	No GC	Gated Clock	No GC	Gated Clock	No GC
AVR Core	0.273	0.543	0.470	1.005	0.886	1.956
USB	2.784	2.806	5.126	5.181	9.813	9.930
EEPROM	0.399	0.399	0.718	0.718	1.358	1.358
PM	2.157	2.157	3.543	3.543	6.314	6.314
DM	1.297	1.297	1.992	1.992	3.384	3.384
Total	**7.061**	**7.371**	**12.130**	**12.726**	**22.289**	**23.47**

Table 5.4: Power consumption (mW) at different frequencies

5.6 Preparing data for simulation and P&R

Once the synthesis step has been performed, the next task is to prepare the data required for the simulation and layout generation tool, which are the following files:

- The Verilog netlist containing the interconnected gates.
- SDC file (Synopsys design constraints) with all the design constraints (input delays, output loads, etc.) and clock information.
- SDF file (Standard delay format) where the cell delays are annotated.

For simulation purposes, it is only necessary the netlist and the sdf file as explained in Section 4.3.

The script for generating these files can be seen in Figure 5.11 where the first three lines are required for three-state nets and naming style issues.

5.7 Summary

The present chapter has explained the different issues concerning the synthesis of this design using Design Compiler. In addition, how to implement a scan chain, increase the clock

```
set verilogout_no_tri true
change_names -rules verilog -hierarchy
set bus_naming_style %s\[%d\]
write -hierarchy -format verilog -output $GATE_PATH/$TOPLEVEL-$STAGE.v

write_sdf $GATE_PATH/$TOPLEVEL-$STAGE.sdf
write_sdc $GATE_PATH/$TOPLEVEL-$STAGE.sdc
```

Figure 5.11: DC script for generating the netlist, SDC and SDF files

frequency and reduce power using clock gating was discussed.

The next chapter will cover the different steps to perform in order to generate the layout of the circuit.

6 Layout generation

Throughout this chapter will be detailed the layout generation flow depicted in Figure 6.1. The final goal is to create the GDSII file, which is the *de facto* standard for describing mask geometry, i.e., the file that is sent to the foundry for the manufacture of the design.

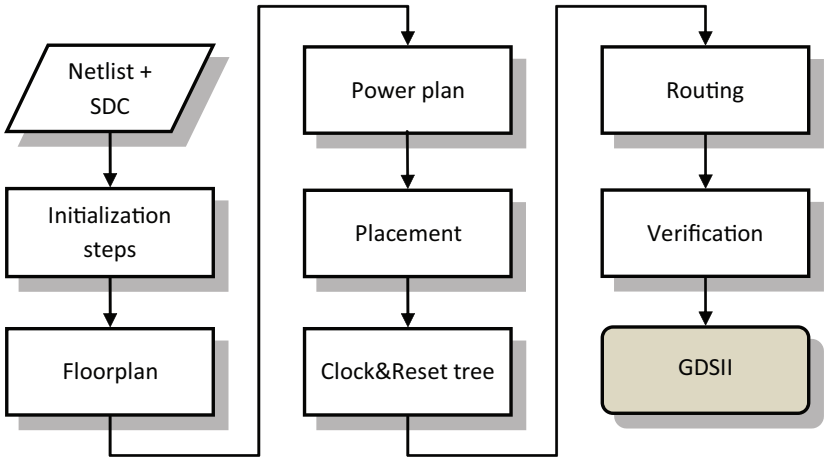


Figure 6.1: Layout generation flow

6.1 Initialization steps

The very first step is to specify to SOC Encounter what and where are the files needed for the layout generation. Basically, the following information is needed:

- Verilog netlist and SDC file (see Section 5.6)
- Timing libraries for "max", "min" and "common" conditions, i.e., the information related to the best, worst and typical case (bc, wc and tc respectively) of the PVT conditions (Process Voltage Temperature)[1].
- "Library Exchange Format" files (LEF files) containing the dimensions and all the physical information about the standard cells for the target process, which in this thesis is a 9 layers process. [2]

We refer to [12] to import this physical and timing information in SOC Encounter in order to focus in more advanced issues in the following subsections.

[1] Apart from transition times and capacitive loadings, cells delays are also a function of voltage and temperature conditions: low voltages and high temperatures yield in higher delays.

[2] The target processes supported by the Faraday library have 6, 7, 8 and 9 layers. The 6 and 7 layers processes are available with different thickness for the layers.

```
# footPrint: I+0I    nrCell: 26
Library:fsd0a_a_generic_core_bc
BUFXLP   I+0I   1      22.779     11.041
     0=I
BUFCKX1  I+0I   2      16.003     18.963
     0=I
BUFX1    I+0I   3      15.956     7.988
     0=I
BUFCKX1PI+0I    4      11.652     12.066
     0=I
BUFX1P   I+0I   5      11.622     5.401
     0=I
BUFCKX2  I+0I   6      8.131      9.094
     0=I
BUFX2    I+0I   7      8.115      3.837
     0=I
DELDX3   I+0I   8      5.717      6.183
     0=I
DELCX3   I+0I   9      5.644      6.073
     0=I
DELBX3   I+0I   10     5.561      5.949
     0=I
DELAX3   I+0I   11     5.496      5.853
     0=I
BUFCKX3  I+0I   12     5.455      6.037
     0=I
BUFX3    I+0I   13     5.435      2.522
     0=I
BUFCKX4  I+0I   14     4.115      4.469
     0=I
BUFX4    I+0I   15     4.106      1.882
     0=I
BUFX5    I+0I   16     3.289      1.498
     0=I
BUFCKX6  I+0I   17     2.760      2.954
     0=I
BUFX6    I+0I   18     2.752      1.249
     0=I
...
```

(a)

```
# footPrint: DEL    nrCell: 4
Library:fsd0a_a_generic_core_tc(min)
DELDX3   DEL  1      8.495      9.333
     0=I
DELCX3   DEL  2      8.348      9.121
     0=I
DELBX3   DEL  3      8.180      8.886
     0=I
DELAX3   DEL  4      8.049      8.702
     0=I
# footPrint:   BUFCK    nrCell: 10
Library:fsd0a_a_generic_core_bc
BUFCKX1    BUFCK1      16.003     18.963
     0=I
BUFCKX1P   BUFCK2      11.652     12.066
     0=I
BUFCKX2    BUFCK3      8.131      9.094
     0=I
BUFCKX3    BUFCK4      5.455      6.037
     0=I
BUFCKX4    BUFCK5      4.115      4.469
     0=I
BUFCKX6    BUFCK6      2.760      2.954
     0=I
BUFCKX8    BUFCK7      2.073      2.201
     0=I
BUFCKX12   BUFCK8      1.386      1.462
     0=I
BUFCKX16   BUFCK9      1.040      1.093
     0=I
BUFCKX20   BUFCK10     0.832      0.874
     0=I
# footPrint:   BUF    nrCell: 12
Library:fsd0a_a_generic_core_bc
BUFXLP     BUF  1      22.779     11.041
     0=I
BUFX1      BUF  2      15.956     7.988
     0=I
...
```

(b)

Figure 6.2: Original footprint file (a) and the modified version (b)

6.1.1 Footprints

A "footprint" is a common name assigned to all the cells with the same functionality but different drive strengths. SOC Encounter requires the footprint name for the buffers, the inverters and the delay cells in order to fixed setup and hold timing violations (see Section 6.4.1 and Section 6.5.1 respectively). Unfortunately, for some incompatibility between the Faraday library and SOC Encounter there are the two problems with the footprints:

1. The footprints for all the cells are replaced by an "unfriendly" nomenclature.

2. The normal buffers and inverters and the specific ones for the clock are identified with the same footprint.

Regarding the first issue, if the timing libraries are checked, footprints such as "AN2" for an AND gate of two inputs can be found. Nevertheless, the footprint used for SOC Encounter for this situation will be "I1+I2+OI1I2", which represents the ports of the cell ("I1+I2+O") and the functionality (I1I2). Once this is known, the footprints from the timing library can be ignored in order to specify the correct name for the buffer, inverter and delay cells based on the new nomenclature created by SOC Encounter. Proceeding with this solution, the second issue will be found soon afterwards in the design due to the "pre-place" optimization performed by SOC Encounter during placement (see Section 6.4). For the moment, the only thing we need to know is that all the normal buffer, inverter and delay cells are deleted in that

```
loadfootprint -infile footprints.cfp
setInvFootPrint INV
setBufFootPrint BUF
setDelayFootPrint DEL
```

Figure 6.3: Fixing the footprint nomenclature

optimization process and re-inserted later on. Therefore, if the clock buffers and inverters have the same footprint as the normal ones, two potential problems are found:

1. Special clock cells could be inserted in normal data path.
2. Normal cells could be used in place of clock cells! [3]

In Section 5.1.3 was explained that the first situation is not desired. Also in that section was discussed that the clock cells are necessary to balance and optimize the clock tree, so the second issue should be always avoided. Consequently, it is essential to split the normal cells and the clock cells following generating a correct footprint nomenclature with the next procedure:

1. Once the design has been imported in SOC Encounter, report all the footprints in a file with:

   reportFootPrint -outfile "fileName".cfp

2. Split manually the normal and clock buffers and inverters and assign a different and more understandable footprint.
3. Reload the modified footprint file into SOC Encounter and specify the footprints for the normal buffer and inverter cells and for the delay cells.

Figure 6.2 shows and example of how to modify the footprint file and Figure 6.3 shows the script to reload the footprint file and set the buffer, inverter and delay cells[4].

6.1.2 I/O file

This file collects the information about the I/O cells and their position in the layout. It also determines the location and number of power/ground pads used for the I/O cells and the core. Table 6.1 contains the list of I/O pads used in this design and a small part of the file is shown in Figure 6.4. Regarding the I/O pads dedicated to I/O power supply, in Section 3.2.6 was briefly explain the effect of the amount of I/O power/ground pads, which, as have been said, limits the maximum output load that can be driven without suffering from Electro-Migration effect.

The value for the output load depending on the I/O to power/ground ratio[5] can be calculated as follows:

[3] Even if the clock tree has not be generated yet, some clock inverters can be present in the clock tree, or, more likely, in the reset tree (in Section 6.5 can be seen that the reset tree is also generated using special clock cells). Furthermore, it can be the case that the clock tree is already generated and, in this situation the whole clock tree structure will be lost.

[4] Although it is not necessary, the delay footprint has been also changed for a more readable scripting

[5] The I/O to power/ground ratio determines how many I/O buffers can be connected to one power/ground pad pair.

I/O Pads	
Power supply I/O (2.5 V)	16
Power supply Core (1.0 V)	8
UART	2
USB	12
Scan Chain	3
MyJtag	2
JTAG	5
SPI	4
GPIO	24
Clock and reset	2
Total	**78**

Table 6.1: I/O pads

```
RowMargin: W 2 77.0
RowMargin: N 2 77.0
RowMargin: E 2 77.0
RowMargin: S 2 77.0

Spacing: 15.68

Skip: 0
Orient: R180
Row: 2
Pad:    io_corner_lb    SW  CORNERGB
Skip: 0
Orient: R270
Row: 2
Pad:    io_corner_rb    SE  CORNERGB
Skip: 0
Orient: R0
Row: 2
Pad:    io_corner_rt    NE  CORNERGB
Skip: 0
Orient: R90
Row: 2
Pad:    io_corner_lt    NW  CORNERGB

Skip: 0
Row: 2
Pad:    io_ipTms    W
Row: 2
Pad:    io_ipTdi    W
Row: 2
Pad:    io_ipTck    W
Row: 2
Pad:    io_vssio_w0 W   GND2IOGB
Row: 2
Pad:    io_vddio_w0 W   VCC2IOGB
...
```

Figure 6.4: I/O File

$$C = \frac{I}{N \cdot V \cdot f}$$

where "$I = 140mA$"[6] is the current-carrying capacity of one pair of power/ground pads, "$V = 2.5V$" is the operation voltage, "f" the operating frequency and "$N = 54/8 = 6.75$" the I/O to power/ground ratio. This formula yields in 82.96pF for 100 MHz and 41.48pF for 200MHz [7]. On the other hand, in Section 6.3 is explained that the number of I/O pads dedicated to core power supply has been overdimensioned.

Finally, concerning the total number of I/O pads, it is necessary to point out that the more I/O pads the smaller the bonding pitch. The "North" and "South" sides of the layout have 20 I/O pads per side, which yields in a challenging bonding pitch of 72.24 μm.

6.1.3 Excluded net

As expected, SOC Encounter does not perform any optimization in the clock net during placement, as explained in Section 6.4, but unfortunately, this is not the case with the reset because SOC Encounter does not support the command "set_ideal_network" use in the SDC file generated by Design Compiler. In order to prevent buffer insertion and "false" timing values during static timing analysis before the reset tree has been generated[8], all the nets being part of the reset tree should be declared as "excluded nets". In order to do this, a file with the following format has to be created:

"netName" "netDelay"ns "netOutputTran"ns "netLoad"pf

with one line per net that has to be excluded. To treat the nets as ideal nets, the values selected for the net delay, the output transition and the net load should be set to "0", unless that some estimations were done to reflect the impact of the reset tree before its generation. This file has to be read with following command:

setExcludeNet -file "fileName"

and subsequently removed before generating the reset tree:

cleanupExcludeNet

6.2 Floorplan

In this step, the physical dimension of the chip has to be specified and the placement of the memories has to be done. The available die for this design has a size of 1875 square μm but, of course, not all the area can be used for the core because of the pad, I/O and power ring as can be seen in Figure 6.6a. Then, in order to supply the power to the I/O cells, filler cells has to inserted in between. Finally, a "halo"[9] for the memories should be specified before place them. It is important to be sure that the specified coordinates are multiples of the routing

[6] Value from Faraday documentation

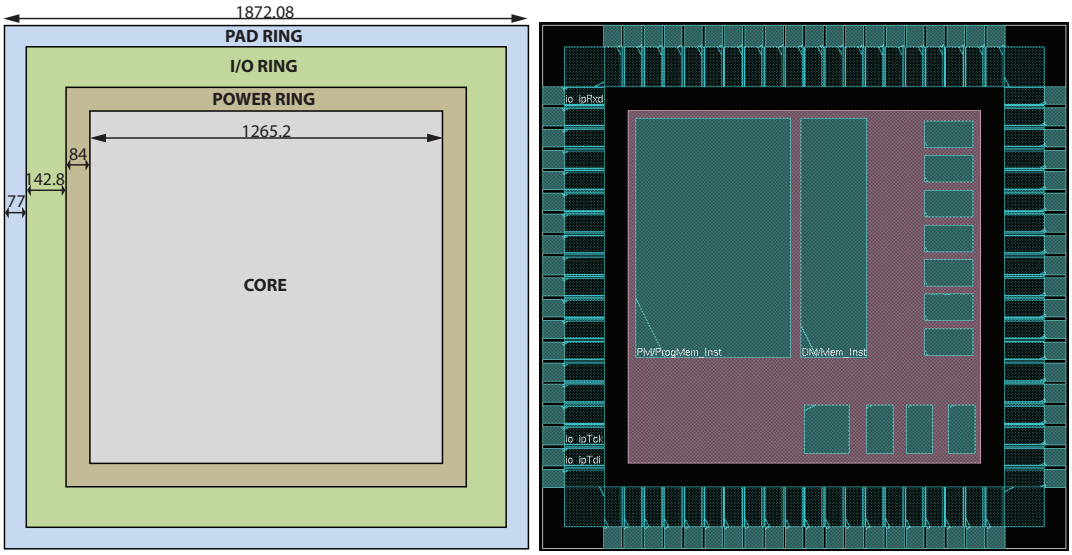
[7] It has to be taken into account that the "operating frequency" of a I/O buffer is likely half of the clock frequency or less, i.e., these are really conservatives values.

[8] The reset net is extremely long and presents a high fanout, therefore, the delays that can produce (before the reset tree is generated) are even bigger than the clock period.

[9] A halo is a perimeter around the memories where the standard cells cannot be placed in order to reduce congestion. Checking the data sheets from the Faraday memories, a halo requirement of 10 μm on each side can be found.

```
floorPlan -b 0.0 0.0 1872.08 1872.08 \
             219.8 219.8 1652.28 1652.28 \
             303.8 303.8 1568.28 1568.28

addIoFiller -row 2 -cell EMPTY16GB -prefix io_fillperi
addIoFiller -row 2 -cell EMPTY8GB -prefix io_fillperi
addIoFiller -row 2 -cell EMPTY4GB -prefix io_fillperi
addIoFiller -row 2 -cell EMPTY2GB -prefix io_fillperi
addIoFiller -row 2 -cell EMPTY1GB -prefix io_fillperi

addHaloToBlock  10 10 10 10 -allBlock

placeInstance DM/Mem_Inst 1300.8800 683.4800 R0
placeInstance PM/ProgMem_Inst 331.8000 683.4800 R0
placeInstance EEP1/EEPROMX1  1300.8800 352.8000 R270
...
```

Figure 6.5: Floorplan specification



(a) (b)

Figure 6.6: Layout dimenesions (a) and floorplan (b)

grid (0.28μm)[10] An example can be seen inFigure 6.5 and the layout after the floorplan is shown in Figure 6.6b.

6.3 Power plan

Once the floorplan has been generated, the next task is to create the "power plan" of the layout, which purpose is to correctly distribute the power supply to all the cells without IR

[10]The placement of the memories can also be done manually and SOC Encounter automatically place the instances in routing grid positions, but taken into account that 13 memories has to be placed, this can be quite tedious.

drop violations[11]. This task is not critical in this design due to the over dimensioning of the power supply. Assuming:

- A clock frequency of 200MHz
- A toggle probability of 1.
- A worst case condition concerning voltage supply, i.e., 0.9V.

SOC Encounter estimates a power consumption of 167.5mW that yields in a current requirement of 186.1mA. The 4 I/O power/ground pairs can supply 416mA according to SOC Encounter, i.e., only 44% of the total current is being used. The IR drop analysis performed in Section 6.7 corroborates these calculations.

6.4 Placement

Now, the final position of the standard cells in the layout has to be determined. With this aim, it is necessary to select a timing-driven placement algorithm to improve the placement of instances on timing critical paths. It is also important to specify the buffer, inverter and delay cells footprints as explained in Section 6.1.1 because we are going to ask SOC Encounter to perform a "pre-place optimization". In this optimization all the buffer, inverter and delay cells are deleted in order to calculate the real capacitance and resistance value associated to every net. With these values, SOC Encounter can estimate the optimum driving capabilities for every cell to fulfil timing requirements, i.e., some cells can be "upsize" or "downsize" depending on the requirements. These two options can be set as follows:

setPlaceMode -timingDriven 1

placeDesign -prePlaceOpt

and the result can be seen in Figure 6.8. Sometimes, SOC Encounter place two cells too close to each other, yielding in a "spacing violation". These violations can be fixed performing geometry verification with: then using the "Vioalation browser" to find the exact location and, finally, spacing the cell manually. Figure 6.7a illustrates an example of "spacing violation".

6.4.1 Setup timing violations

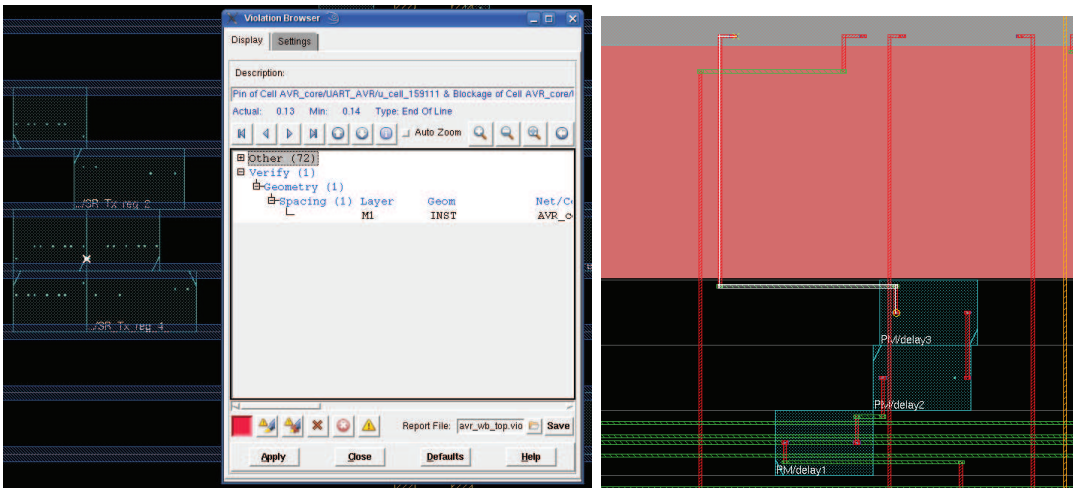
If a static setup timing analysis[12] is performed at the moment, hundreds, or even thousands, of violating paths can be found. This is due to the fact that all the buffers and inverters were deleted and now have to be inserted again. As have been said before, SOC Encounter optimizes the driving strength of the cells and reinserts the buffers and inverters in the optimal position to achieve timing requirements. Before executing the optimization, the delay cells used to increase the clock frequency as explained in Section 5.4 must be preserved from being eliminated:

set_dont_touch true "instanceCellDelaysList"

It is also necessary to check that the delay cells were placed next to clock pin of the memories as depicted in Figure 6.7b. Then the optimization can be performed just by typing:

[11] IR drop is the voltage drop through the layout, i.e., the real voltage that a cell could have is for example 0.98V instead of 1V, yielding in a IR drop of 20mV

[12] The "setup time" is the amount of time **before** the clock edge that data input must be stable.



(a) (b)

Figure 6.7: "Spacing violation" (a) and delay cells placement (b)

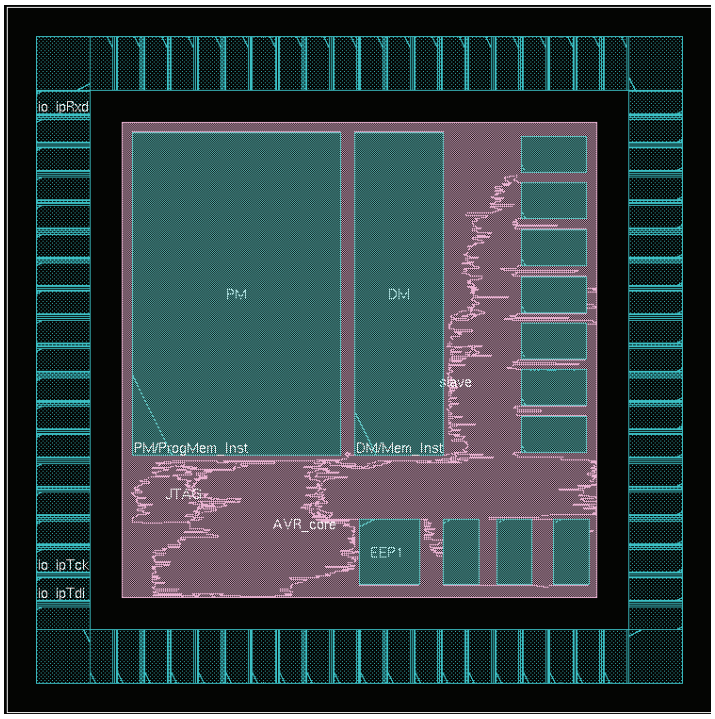


Figure 6.8: Layout after placement

optDesign -preCTS -setup

If the previous analysis is repeated, no setup timing violations should be found now.

When the optimization process has been performed, the "exclude net" property should be removed with:

cleanupExcludeNet

6.5 Clock and reset tree generation

A properly synthesized clock and reset tree is essential to achieve a functional design. The clock should be able to reach all the points of the design at the same time, otherwise, data from the past clock cycle can be used in the current cycle. The requirements for the reset are not so strict because it is "only" necessary that reaches all the logic within a clock cycle. In order to performed this task with SOC Encounter, a "clock tree specification file" has to be provided containing the synthesis information for every clock and reset in the design and the use the following command:

specifyClockTree -clkFile "fileName"

Next is shown an example for the clock of the AVR core. The option "NoGating" has to be set to "No" to allow SOC Encounter to trace the clock tree through the clock gating cells.

```
AutoCTSRootPin io_ipSys_clk/O

NoGating No
Buffer BUFCKX1 BUFCKX1P BUFCKX2 ...
MaxDelay 10ps
MinDelay 0ps
MaxSkew 100ps

End
```

Regarding the reset tree, two main issues have to be solved:

- As have been said in Section 5.2, this design presents two synchronous resets in the SPI and USB components. By default, SOC Encounter treats the synchronous set and reset pins as excluded pins, preventing the generation of the reset tree in this two components.
- The logic used throughout the reset tree (reset synchronizer, reset chain, USB logic to generate the internal synchronous reset, etc) is an obstacle for SOC Encounter when trying to synthesize the reset tree.

The first problem can be solved configuring SOC Encounter to treat the data pins as synchronous pins with:

setCTSMode -setDPinAsSync

and the use *ExcludePin* inside the CTS specification file to exclude the non desire pins in the tree synthesis process. The option *setDPinAsSync* is also necessary to make possible the generation of the internal USB reset, otherwise SOC Encounter does not allow to use *AutoCTSRootPin* in the synchronous pin that is the origin of the internal USB synchronous reset. Regarding the second issue, the option *ThroughPin* has to be used to trace the reset tree through the logic of, for example, the reset synchronizer or the reset chain. An example illustrating all this commands can be seen in Figure 6.9 and the complete clock tree in Figure 6.10

```
AutoCTSRootPin io_ipReset/O

ThroughPin
+ rstSynchronizer/syncReset_reg/SB
+ rstChain/avrReset_reg/RB
slave/wpip/USBwrapper_inst/u_hostSlaveMux/
    u_hostSlaveMuxBI_rstSyncToUsbClkOut_reg/D

ExcludePin
AVR_core/SPI_AVR/U16/A1

NoGating     No
Buffer   BUFCKX1 BUFCKX1P BUFCKX2 ....
#MaxDelay    10ps
#MinDelay    0ps
MaxSkew      100ps

End

AutoCTSRootPin slave/wpip/USBwrapper_inst/u_hostSlaveMux/
            u_hostSlaveMuxBI_rstSyncToUsbClkOut_reg/Q
...
```

Figure 6.9: Reset specifications

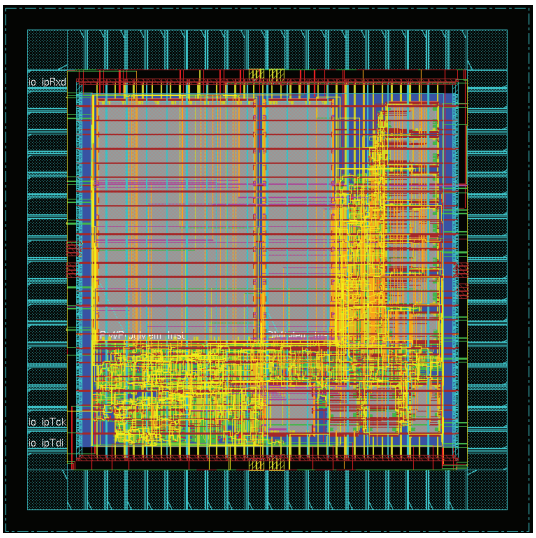


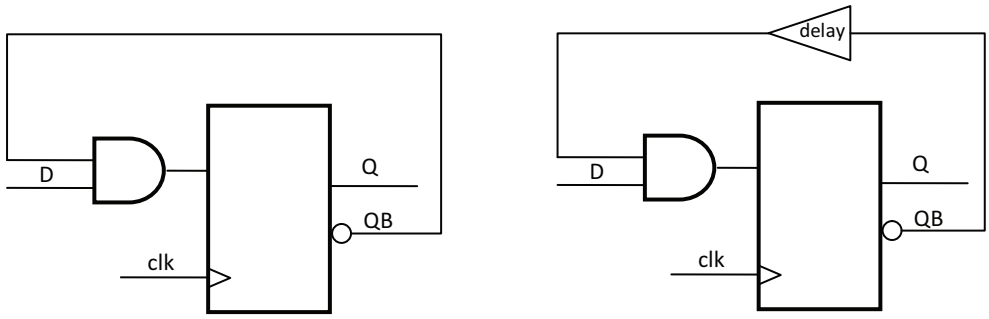
Figure 6.10: Clock tree

6.5.1 Hold timing violations

As soon as the clock tree has been synthesized, a hold timing [13] analysis has to be done. Now, the necessity of a correct specification for the delay footprint is clearly illustrated in Figure 6.11; SOC Encounter needs to use the delay cells to fixed situations as depicted in Figure 6.11a, which are quite common throughout the design. Like before, the design needs to be optimized using:

optDesign -postCTS -hold

[13]The "hold time" is the amount of time **after** the clock edge that data input must be held stable



(a) (b)

Figure 6.11: Potential hold timing violation (a) and the corresponding solution (b)

```
selectNet -allDefClock
setNanoRouteMode -quiet route_selected_net_only true
globalDetailRoute

setNanoRouteMode -quiet route_selected_net_only false
globalDetailRoute
```

Figure 6.12: Routing script

6.6 Route

The routing step creates the real interconnections between all the cells. Depending on the density of the design and the layers available for routing, congestion can be a problem during this process. Due to the medium die utilization in this design (approximately 60%) and the fact that the target process selected has 9 layers, SOC Encounter is able to achieve a successfully routed design without too much effort. Nevertheless, even if the routing step is not a critical issue in the present thesis, it should be taken into account that the clock and reset tree have to be routed in first place in order to get the best clock and reset tree synthesis possible. In Figure 6.12 can be seen a basic script to route first the clock networks (the reset tree is seen by SOC Encounter as a clock) and then the rest of the networks. The first line of the script calls another script to insert the filler cells in the design in order to supply power properly to all the standard cells. Once the design has been routed, the final step in the layout generation is to include the pads [14] around the design. Figure 6.13 shows the final result.

6.7 Verification and results

Finally, the DRC (Design Rule Check) process has to be run in order to find possible layout rule violations. This can be performed just by typing:

verifyGeommetry

[14] A "pad" is a simple piece of metal were the bonding wires are attached

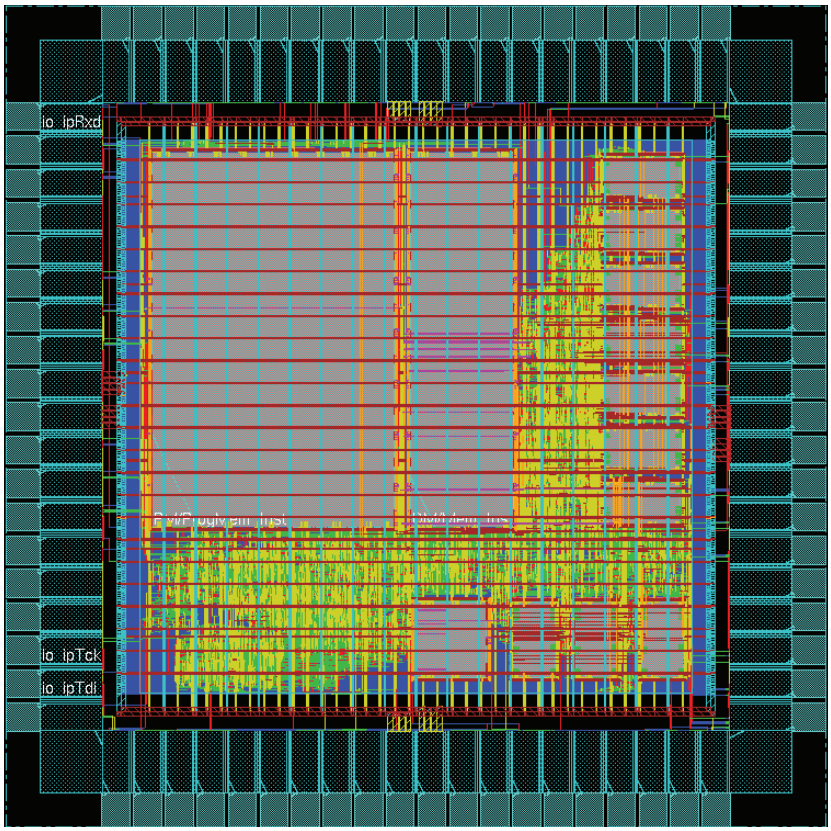


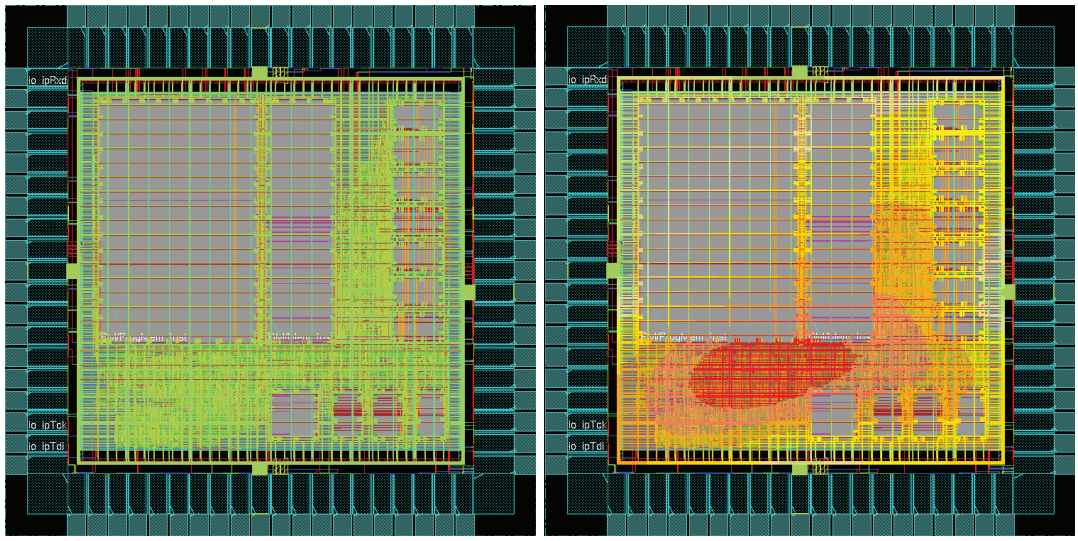
Figure 6.13: Final layout

```
#Netlist
saveNetlist avr_wb_top.v

#SDF and SPEF
setExtractRCMode -detail
extractRC
delayCal  -sdf avr_wb_top.sdf -version 2.1

#GDSII
streamOut ./RESULTS/tud_d3lab_c1_0810.gds \
          -mapFile streamOut.map \
          -libName DesignLib \
          -structureName tud_d3lab_c1_0810 \
          -stripes 1 \
          -units 1000 \
          -mode ALL
```

Figure 6.14: GDSII extraction

If there are no violations, the GDSII file can be extracted. Figure 6.14 shows an example where also the SPEF, SDF and verilog netlist files are extracted to use in simulation and back-annotated synthesis.

Now, also the IR drop throughout the circuit can be performed. Figure 6.15 displayed two IR Drop analyses with different thresholds. As predicted in the discussion about the over



(a) IRD threshold = 90mV (b) IRD threshold = 10mv

Figure 6.15: IR Drop for two different thresholds

dimension of the power supply in Section 6.3, the voltage drop is negligible[15] due to the extra power applied to the core.

6.8 Summary

The gap from the RTL code to the GDSII file has been explained in this chapter. Different issues concerning the layout generation using SOC Encounter were discussed: footprint specification, timing optimization, clock and reset tree synthesis, etc.

The following chapter will give a short introduction to the back-annotated synthesis process.

[15]It has to be taken into account that this IR drop analyses were executed with a power consumption estimation using a toggle probability of 1, i.e., the IR Drop threshold of 10mV is really pessimistic.

7 Back-annotated synthesis

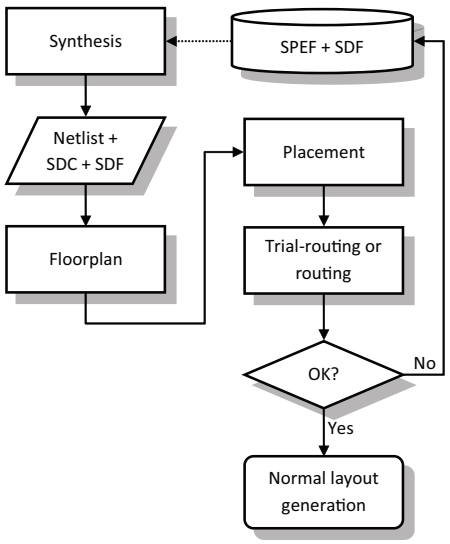


Figure 7.1: Back-annotated synthesis flow

In this chapter will be briefly introduced the back-annotated synthesis process following the discussion initiated in Section 5.4 about how to increase the clock frequency. The term "back-annotated synthesis" refers to the use of parasitic and timing information generated after place and routing to re-synthesis the design. In the first synthesis pass, Design Compiler uses an estimation of the capacitance and the resistance of the wires to calculate delays in the interconnection and the gates. This estimation is provided by the library vendor (Faraday Technology in this thesis) in form of "wire load models" (WLM) but the problem is the dependence of the capacitance and resistance on the length of the wires since this can only be known with enough accuracy after P&R.

Figure 7.1 illustrates the flow used in this thesis to perform a back-annotated synthesis. It can be seen that the information is back-annotated into Design Compiler using the SPEF and SDF files generated after P&R, and subsequently forward-annotated in SOC Encounter through the new SDF file, which is used to perform a timing driven placement based on this timing information. Figure 7.2 shows how to execute the back-annotatoin in Design Compiler and Figure 7.3 how to generate the data required in SOC Encounter. The step called "Trial or Normal route" refers to the two options available at this point, i.e., perform a "trial route", which is faster and less accurate, or a "normal route", which provides more precision at the expense of running time.

Once the timing requirements have been met, or no further optimization is possible, the layout generation cycle is repeated, as detailed in the previous chapter, with the improved netlist.

Figure 7.4a and Figure 7.4b illustrates the different mapping performed by Design Com-

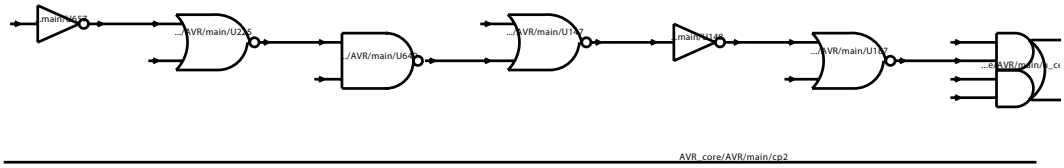
```
read_verilog -netlist avr_wb_top.v

#Read the .spef file
current_design $TOPLEVEL
redirect readSPEF.log { read_parasitics avr_wb_top.spef}

#Read the .sdf file
current_design $TOPLEVEL
redirect readSdf.log { read_sdf avr_wb_top.sdf }

#Read the .sdc file
current_design $TOPLEVEL
redirect readSdc.log { read_sdc avr_wb_top.sdc }

set_fix_hold [all_clocks]
set_dont_use  $LIBNAME/BUFCK*
set_dont_use  $LIBNAME/INVCK*

redirect compile-inc.log  { compile_ultra  -inc}
```

Figure 7.2: Back annotated synthesis script for Design Compiler

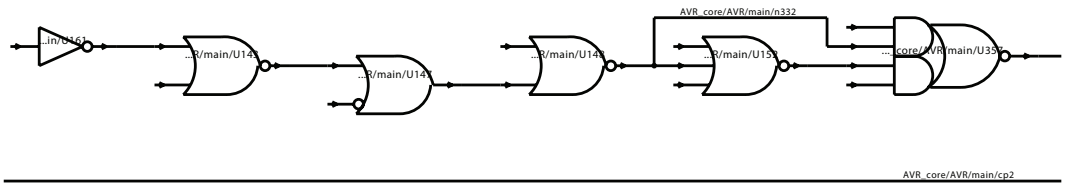
```
source avr_wb_top-init.tcl
read_sdf avr_wb_top.sdf
setExcludeNet -file excludeNet.exc
source avr_wb_top-floorplan.tcl
source placeMemories.tcl
source avr_wb_top-place.tcl
trialRoute

exec mkdir -p RESULTS
saveNetlist ./RESULTS/avr_wb_top.v
setExtractRCMode -detail
extractRC
rcOut -spef ./RESULTS/avr_wb_top.spef
delayCal  -sdf ./RESULTS/avr_wb_top.sdf -version 2.1
```

Figure 7.3: Back annotated synthesis script for SOC Encounter

piler due to the back-annotation after P&R. Figure 7.5 shows the improvement in terms of timing with respect to the previous implementation shown in Figure 5.10. This improvement is small due to the fact that Design Compiler stops optimizing the design when the timing requirements are achieved in order to save area and power. So, in order to obtain the best result, more restrictive timing requirements need to be set and several iterations of back-annotation synthesis have to be performed. Following this procedure, the final frequency can be increased up to 200 MHz.



(a) Part of the critical path before improvement



(b) Part of the critical path after back annotation

Figure 7.4: Optimization in the critical path

```
Timing delays computed with First Encounter Delay Calculator delay calculator.
 * Some/all delay information is back-annotated.
 # A fanout number of 1000 was used for high fanout net computations.
  Startpoint: AVR_core/AVR/main/adiw_st_reg
              (rising edge-triggered flip-flop clocked by sys_clk_pad)
  Endpoint: AVR_core/AVR/main/pc_low_reg_1_
            (rising edge-triggered flip-flop clocked by sys_clk_pad)
  Path Group: sys_clk_pad
  Path Type: max
  Point                                                    Incr       Path
  ---------------------------------------------------------------------------
  clock sys_clk_pad (rise edge)                            0.00       0.00
  clock network delay (ideal)                              0.00       0.00
  AVR_core/AVR/main/adiw_st_reg/CK (QDFFRBX1)              0.00       0.00 r
  AVR_core/AVR/main/adiw_st_reg/Q (QDFFRBX1)               0.24       0.24 f
  AVR_core/AVR/main/U312/O (NR2X1)                         0.20       0.44 r
  AVR_core/AVR/main/U299/O (AN4B1XLP)                      0.15       0.59 r
  AVR_core/AVR/main/U215/O (AN3X1)                         0.13       0.72 r
  AVR_core/AVR/main/U161/O (INVX1)                         0.07       0.79 f
  AVR_core/AVR/main/U143/O (NR2X1)                         0.19       0.98 r
  AVR_core/AVR/main/U147/O (OR2B1XLP)                      0.27       1.25 r
  AVR_core/AVR/main/U148/O (NR2X1)                         0.17       1.42 f
  AVR_core/AVR/main/U152/O (NR3X1)                         0.34       1.76 r
  AVR_core/AVR/main/U357/O (AOI22XLP)                      0.10       1.86 f
  AVR_core/AVR/main/U359/O (OAI112X1)                      0.09       1.95 r
  AVR_core/AVR/main/U233/O (BUFX1)                         0.60       2.55 r
  AVR_core/AVR/main/adr[0] (pm_fetch_dec)                  0.00       2.55 r
  AVR_core/AVR/adr[0] (AVR_Core)                           0.00       2.55 r
  AVR_core/WishboneInterface/adr[0] (wishbone_interface)
                                                           0.00       2.55 r
  AVR_core/WishboneInterface/u_cell_159051/O (NR3X1)       0.10       2.65 f
  AVR_core/WishboneInterface/U1/O (OR4B2XLP)               0.47       3.12 r
  AVR_core/WishboneInterface/U16/O (AOI12X1)               0.17       3.29 f
  AVR_core/WishboneInterface/out_en (wishbone_interface)
                                                           0.00       3.29 f
  AVR_core/EXT_MUX/io_port_en_bus_7_ (external_mux)        0.00       3.29 f
  AVR_core/EXT_MUX/U96/O (NR2X1)                           0.12       3.41 r
  AVR_core/EXT_MUX/U95/O (OR3B1XLP)                        0.11       3.53 f
  AVR_core/EXT_MUX/U87/O (NR2X1)                           0.12       3.65 r
  AVR_core/EXT_MUX/U83/O (AN4B2X1)                         0.45       4.10 r
  AVR_core/EXT_MUX/U9/O (AOI22X1)                          0.13       4.23 f
  AVR_core/EXT_MUX/U13/O (OAI112X1)                        0.10       4.34 r
  AVR_core/EXT_MUX/U21/O (AOI112X1)                        0.06       4.40 f
  AVR_core/EXT_MUX/U52/O (OAI12X1)                         0.12       4.52 r
  AVR_core/EXT_MUX/dbus_out[3] (external_mux)              0.00       4.52 r
  AVR_core/AVR/dbusin[3] (AVR_Core)                        0.00       4.52 r
  AVR_core/AVR/io_dec/dbusin_ext[3] (io_adr_dec)           0.00       4.52 r
  AVR_core/AVR/io_dec/u_cell_158926/O (INVX1)              0.04       4.56 f
  AVR_core/AVR/io_dec/U1/O (OAI112X1)                      0.25       4.81 r
  AVR_core/AVR/io_dec/dbusin_int[3] (io_adr_dec)           0.00       4.81 r
  AVR_core/AVR/BP_Inst/dbusin[3] (bit_processor)           0.00       4.81 r
  AVR_core/AVR/BP_Inst/U2/O (AOI222X1)                     0.16       4.97 f
  AVR_core/AVR/BP_Inst/U1/O (NR2X1)                        0.09       5.06 r
  AVR_core/AVR/BP_Inst/U41/O (AOI13X1)                     0.05       5.11 f
  AVR_core/AVR/BP_Inst/U40/O (OAI112X1)                    0.14       5.24 r
  AVR_core/AVR/BP_Inst/U38/O (OAI23X1)                     0.12       5.37 f
  AVR_core/AVR/BP_Inst/U25/O (OAI112X1)                    0.10       5.47 r
  AVR_core/AVR/BP_Inst/bit_test_op_out (bit_processor)
                                                           0.00       5.47 r
  AVR_core/AVR/main/bit_test_op_out (pm_fetch_dec)         0.00       5.47 r
  AVR_core/AVR/main/U243/O (ND2X1)                         0.10       5.57 f
  AVR_core/AVR/main/U259/O (INVX1)                         0.37       5.94 r
  AVR_core/AVR/main/U140/O (OR2X2)                         0.25       6.18 r
  AVR_core/AVR/main/U144/O (INVX1)                         0.11       6.29 f
  AVR_core/AVR/main/U214/O (AOI22XLP)                      0.15       6.44 r
  AVR_core/AVR/main/U212/O (OAI112X1)                      0.08       6.52 f
  AVR_core/AVR/main/pc_low_reg_1_/D (QDFERBX1)             0.00       6.52 f
  data arrival time                                                   6.52

  clock sys_clk_pad (rise edge)                            7.00       7.00
  clock network delay (ideal)                              0.00       7.00
  clock uncertainty                                       -0.20       6.80
  AVR_core/AVR/main/pc_low_reg_1_/CK (QDFERBX1)            0.00       6.80 r
  library setup time                                      -0.18       6.62
  data required time                                                  6.62
  ---------------------------------------------------------------------------
  data required time                                                  6.62
  data arrival time                                                  -6.52
  ---------------------------------------------------------------------------
  slack (MET)                                                         0.10
```

Figure 7.5: Critical path improvement

7.1 Summary

How to back-annotate the design with data extracted from the P&R process was explained in this chapter. It has been proved that more strict timing requirements can be achieved using this methodology.

The next and last chapter will summarize the results obtained in this thesis.

8 Results

Throughout this thesis the following results were achieved:

- A VLSI design flow explained in Section 2.2 has been successfully completed in 90nm technology for a complex design such as an 8-bit microcontroller with the features described in Table 8.1. The different issues related with the synthesis and layout generation of the circuit have been solved in order to reach the final goal of a VLSI design flow: the generation of the GDSII file, i.e., the file that is sent to the foundry for the manufacturing of the circuit.

- The clock frequency of the microcontroller was initially 20MHz but now, the speed can be increased up to 200MHz with the resulting improvement in terms of processing capacity and higher transmissions rates in the peripherals, such as the UART or the SPI. This achievement has been obtained through the architectural modifications explained in Section 5.4 and advanced design techniques such as back-annotated synthesis.

- DFT techniques has been successfully integrated in the design as explained in Section 5.3.

- The integration of a complex transmission interface such as the USB 1.1 into the microcontroller using the Wishbone bus has been achieved. The additional complexity introduced by the necessity of synthesize a reset tree involving synchronous and asynchronous reset has been accomplished.

- A clock gating technique have been applied in order to obtained a power consumption reduction of a 50% [1]

- A back-annotated synthesis approach to meet timing requirements was successfully performed.

- An additional programming interface has been developed in order to overcome a potential malfunction of the JTAG interface after manufacturing.

8.1 Future work

In this section will be briefly pointed out some future work suggestions related with the present design:

- The first and obvious task is the manufacturing of the design to verify and test the microcontroller.

[1] Without taking the memories into account

Features		
Memories	Program Memory	16k x 16 bits
	Data Memory	8k x 8 bits
	Parameter Memory	256 x 8 bits
	10 FIFO's (USB)	64 x 8 bits
Peripherals	USB 1.1	
	UART	
	SPI	
	GPIO	
	Programmable I/O lines	24
Design data	Die size	1875 x 1875 μm
	Speed Grade	0 - 200 MHz
	Power	160 mW
	Core voltage	1.0V
	I/O voltage	2.5V
	I/O pads	78
	Process	UMC 90 SP CMOS

Table 8.1: Microcontroller features

- As have been said in Section 6.3, the power supply has been over dimensioned. It is strongly recommended to try to optimize the I/O power pads in order to save I/O pins. Therefore, a design with only one I/O power/ground pair could be study and see how the lack of symmetry in terms of power supply affect to power distribution, specially taking care about IR drop issues.

- In Section 2.5 was stated the necessity of DFT insertion in the design and, according to this, a simple scan chain approach was implemented. The next step concerning this aspect would be the insertion of a more sophisticated DFT technique in the design.

- The Wishbone slave created to implement the scan chain is ready to include any functionality that requires high computational effort (such as a high speed multiplier, a FFT , etc.) in order to be performed in parallel with the microcontroller.

- Following with the previous suggestion, an ISE (Instruction Set Extension) oriented to DSP (Digital Signal Processing) applications could be also investigated.

- Finally, the back-annotated synthesis approach explained in Chapter 7 should be analyze more in depth to make the most of this optimization technique.

List of Abbreviations

ADC Analog to Digital Converter
BC Best Case
BIST Buit In Self Test
CTS Clock Tree Synthesis
DFT Design For Testability
DM Data Memory
DRC Desing Rule Check
DUT Design Under Test
EDA Electronic Design Automation
EEPROM Electrically-Erasable Programmable Read-Only Memory
FFT Fast Fourier Transform
FIFO First-In First-Out
FPGA Fiel Programmable Gate Array
GDSII Graphic Data System II
GPIO General Purpose Input Output
HDL Hardware Description Language
I/O Input/Output
IC Integrated Circuit
IEEE Institute of Electrical and Electronics Engineers
IP Intellectual Property (Core)
ISE Instruction Set Extension
JTAG Join Test Action Group
LEF Library Exchange Format
LSB Less Significant Bit
LSI Large Scale Integration
LSSD Level Sensitive Scan Design
MSI Medium Scale Integration
P&R Place and Routing
PCI Peripheral Component Interconnect
PM Program Memory
PVT Process Voltage Temperature
PWM Pulse-Width Modulator
RAM Random Access Memory

RISC Reduced Instruction Set Computer
RTL Register Transfer Level
RVT Regular Voltage Threshold
SDC Synopsys Design Constraints
SDF Standard Delay Format
SoC System on Chip
SPI Serial Peripheral Interface
SRAM Static Random Access Memory
SSI Small Scale Integration
SSO Simultaneous Switching Output
TC Typical Case
TCK Test Clock
TDI Test Data In
TDO Test Data Out
TMS Test Mode Select
TPAP Test Access Port
TRST Test Reset
UART Universal Asynchronous Receiver-Transmitter
USB Universal Serial Bus
VLSI Very Large Scale Integration
WC Worst Case
WLM Wire Load Models

Bibliography

[1] Qing, K. Z. (2003) *High Speed Clock Networks Design.*Boston: Kluwer Academic Publishers.

[2] Rabaey, J. M., A. Chandrakasan & B. Nikolic (2003) *Digital Integrated Circuits: A Design Perspective (2nd edn.)* Upper Saddle River: Pearson Education.

[3] Weste, N. H. B. & D. Harris (2005) *CMOS VLSI Design: A Circuits and Systems Perspective. (3rd edn.)* Pearson Education Addison-Wesley.

[4] Golson, E., D. Mills & E. C. Clifford (2003) *Asynchronous & Synchronous Reset Design Techniques - Part Deux.* SNUG Boston 2003

[5] Fielding, S. (2008) *USBHostSlave IP Core Specification Rev. 1.2.* OpenCores.Org

[6] Moore, G. E. (1965) *Cramming more components onto integrated circuits.* Electronics Magazine

[7] Jha, N.K. & S. Gupta (2003) *Testing of digital systems.* Cambridge: Cambridge University Press

[8] Synopsys (2007) *DFT Overview. User guide*

[9] Synopsys (2007) *DFT Compiler. User guide: Scan*

[10] Mentor Graphics (2007) *Modelsim User's Manual*

[11] Digital Asic Group Lund University (2005) *Digital ASIC Design. A Tutorial on the Design Flow*

[12] Cadence 2007 *Encounter User guide*