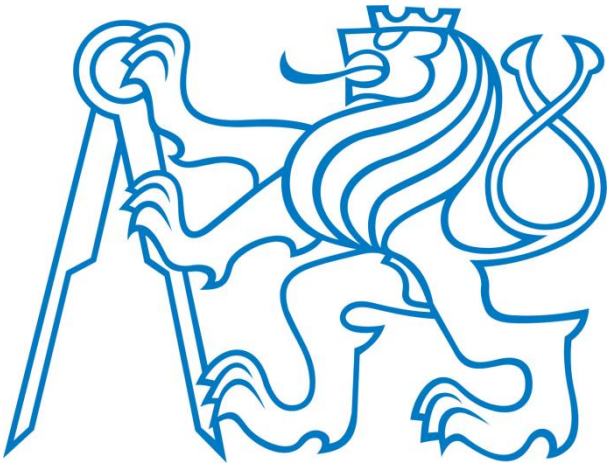
Faculty of Information Technology

Czech Technical University in Prague

Department of Digital Design



Master's thesis

Design of a digital I2C slave IP block

Jan Vošalík

Supervisor: Ing. Jan Schmidt, Ph.D.

Acknowledgement

I would like to express thanks to Ing. Stanislav Trojan and Ing. Jan Schmidt PhD. for their help, guidance and leadership during my work on my Master's thesis.

Statement

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded an agreement with the Czech Technical University in Prague, on the basis of which the Czech Technical University in Prague has waived its right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60(1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prague on …… ……...

Abstrakt

Nízká spotřeba se stala velice důležitou součástí návrhu dnešních čipů. Cílem této diplomové práce je návrh zařízení pro přenos dat mezi I2C a APB sběrnicemi za použití technik pro nízkou spotřebu. Verifikace je též součástí práce.

Práce nejprve srovnává různé techniky návrhu zařízení s nízkou spotřebou. Jako výsledek tohoto porovnání bylo v návrhu užito techniky hradlování hodin. Byla provedena analýza s patřičným odůvodněním popisující, na které registry bylo hradlování hodin použito.

Jednotlivé kroky postupu začínají od specifikace a pokračují až po fyzický design. Verifikace byla provedena samokontrolními testy. Pokrytí kódu je v práci rovněž užito společně s grafickou ukázkou pokrytí stavových strojů.

Pro možnost srovnání více výsledků bylo užito více metod hradlování hodin, kterými jsou: hradlování nepoužito, automatické hradlování (provedeno během syntézy), manuální hradlování (manuálně vloženy hradlovací buňky) a kombinovaná metoda manuálního a automatického hradlování.

Odhad spotřeby (nástroji k tomu určenými) byl proveden jak po syntéze, tak po fyzickém návrhu. Odhady, které byly provedeny po fyzickém návrhu, byly provedeny pro mód nečinnosti a komunikační mód zařízení. Výsledky odhadu spotřeby jsou porovnány a ukázány jsou i případy užití a spotřeba u těchto případů.

Klíčová slova: RTL, I2C, APB, low power design, clock gating, odhad spotřeby.

Abstract

Low power has become a very important part of designing today's chips. The goal of this thesis is to design a device for transmitting data between I2C and APB buses while considering low power techniques in the design. Verification is also a part of this thesis.

This thesis first compares the different techniques used for low power design. As a result of the comparison, clock gating technique is used in the design. An analysis was done to describe the registers that the clock gating is used for, and the reasons to use clock gating at these registers.

The work flow goes from specification to physical design. Verification was done using self-checking tests and code coverage is also used in the thesis, along with graphical examples of FSM coverage.

Four different methods of clock gating were used to compare different results. These methods are: no clock gating use, automatic clock gating (placed during synthesis), manual clock gating (manually placed cells), and manual clock gating, combined with automatic clock gating.

Power estimations were done and compared after the synthesis, as well as after the physical design. The power estimations done after the physical design, were done for idle and communication mode of the device. The results of the power consumption estimation are compared and use cases are shown, as well with their power consumption.

Keywords: RTL, I2C, APB, low power design, clock gating, power estimation.

Content

Figure index

Table index

Used abbreviations

Abbreviation	Explanation
AVS	Adaptive voltage scaling
CG_AUTO	Automatic clock gating (used during Synthesis)
CG_MAN	Manual clock gating
CG_MAN_AUTO	Manual clock gating combined with automatic clock gating during synthesis
CG_NONE	No clock gating
COR	clear on read
CTS	Clock Tree Synthesis
DP	Diploma project
DP device	Diploma project device
DVFS	Dynamic voltage and frequency scaling
DVS	Dynamic voltage scaling
Fm	Fast-mode
Fm+	Fast-mode Plus
Hs	High-Speed mode
Multi Vdd, MSV	Multiple supply voltages
Multi Vt	Multi-Threshold
RO	Read only
RW	Read/Write
S&RPG	Save and restore power gating
Sm	Standard-mode
SOC	System on chip
SRPG	State retention power gating
WO	write only

1 Introduction

1.1 The purpose and goals of this document

This document is the documentation to my Master's theses. The goal of this thesis was to design an IP core that will be able to communicate with I2C and APB bus as a Slave device with use the of low-power techniques. It was intended to design a device for physical layer only – the protocols for a particular use (e.g. if the I2C Master wants an answer from CPU or if data are only being sent to CPU and to answer is expected) would have to be designed according to the use.

Let's assume that from now on the, the abbreviation DP device will be used for this device, standing for Diploma project device.

I2C is a bit serial bus. It is often used in pad-limited design, where the speed can be limited. It has the advantage in using only two signals for communication (SDA, SCL signals).

APB bus is a parallel bus; in this case, it is used as an 8-bit bus. APB bus is used to connect peripheral devices with a CPU. One of the first activities of the project was to study how the protocols work. Therefore there is also a brief description of these protocols.

The overall connection of the device is shown in Figure 1. The DP device is connected to I2C using pads (on the left side of the picture) and connected to a CPU using APB bus (right side of Figure 1).

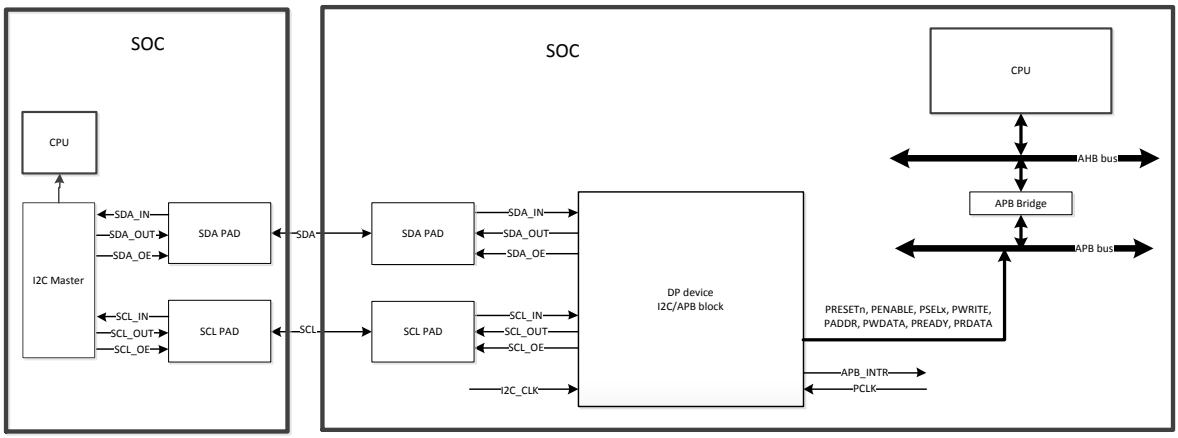


Figure 1: Connection of the DP device among other devices in a system

Low-power techniques were supposed to be described and used in the design. I researched of these techniques and described them in the document. After taking in count their characteristics and use, I decided to use clock gating, as it would be the most suitable technique for this design. The use of clock gating is also part of the assignment.

Development of the IP on RTL level was the next step in the project. This was first designed as schemas, which are also shown and described in this document. I then wrote the RTL in Verilog 2001. Clock gating is included in the Verilog coded as an option through defines, which gives the option of using or not using the clock gating cells I manually placed in the design.

There were four different alternatives of clock gating that were used in order to compare the power consumption – no clock gating, automatic clock gating (done during synthesis), manual clock gating (placing manually clock gating cells) and manual clock gating combined with automatic clock gating. These four different alternatives were measured and compared.

The overall goal was to use low power aware design and compare the consumption results with and without the use of these techniques. The assignment says to compare the consumption estimation after synthesis, however because these estimations are not very accurate and usually differ by 30-50%, I went further and continued with physical design and measured the consumption after the physical design was done. That gave very accurate power consumption estimations which gave adequate results.

1.2 Brief overview of each chapter

1.2.1 Chapter 1 - Introduction

This chapter contains an introduction to the topic with description of the overall project as well as its goals.

1.2.2 Chapter 2 - Protocols descriptions

This chapter briefly describes I2C and APB protocols that were used in the design.

1.2.3 Chapter 3 - Low-Power techniques

This chapter describes all the different kinds of techniques for low-power design as well as the reasoning why clock gating was used in the design.

1.2.4 Chapter 4 - Design and Verification flow

This chapter describes the whole design and verification flow that was used for the development of the IP. It contains the RTL description of the device, description of verification and the verification tests that were used, descriptions of FSMs, the description and reasoning for what registers clock gating was used for. It describes also the different phases of physical design such as Floorplan, Cell place, Clock tree synthesis and Routing.

1.2.5 Chapter 5 - Power consumption results

This chapter contains final consumption results and explanations why in different modes are different power consumptions. This chapter also describes use cases of the design and the power consumption in those cases.

1.2.6 Chapter 6 - Summary

This chapter contains the summary of this whole document and describes the results that were reached in this thesis.

2 Protocols descriptions

2.1 I2C Protocol description

This device communicates with the I2C standard rev. 03. The device is an I2C Slave device operating in Sm, Fm and Fm+ modes with 7-bits addressing. The explanations of these terms follow. The description of the I2C protocol is not complete in this document, but is focused on these characteristics. The complete documentation of the I2C Standard can be found in (B.V., 2007).

I2C is a bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. (B.V., 2007)

Two wires, serial data (SDA) and serial clock (SCL), carry information between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a transmitter or receiver, depending on the function of the device. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfer. A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

2.1.1 Speed modes

All devices are downward compatible – any device may be operated at a lower bus speed. Sm, Fm and Fm+ modes have the same bus protocol and data format. The data format of Hs mode, however is different.

- **Standard-mode (Sm)** – up to 100 kbit/s
- **Fast-mode (Fm)** – up to 400 kbit/s
- **Fast-mode Plus (Fm+)** – up to 1 Mbit/s
- **High-speed mode (Hs)** – up to 3.4 Mbit/s

2.1.2 SDA and SCL Signals

- SDA (serial data line) - serves for transferring data
- SCL (serial clock line) – used as a logical clock for I2C

2.1.3 Reserved addresses

Table 1: Reserved addresses

Slave address	R/W bit	Description
0000 000	0	general call address[1]
0000 000	1	START byte[2]
0000 001	X	CBUS address[3]
0000 010	X	reserved for different bus format[4]
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	reserved for future purposes
1111 0XX	X	10-bit slave addressing

2.1.4 Data transfer example

Figure 2 shows a complete data transfer in a block level. After the START condition (S), a slave address is sent. This address is seven bits long followed by an eighth bit which is a data direction bit ($R/\overline{W}$) — a 'zero' indicates a transmission (WRITE), a 'one' indicates a request for data (READ). A data transfer is always terminated by a STOP condition (P) generated by the master. However, if a master still wishes to communicate on the bus, it can generate a repeated START condition (Sr) and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such a transfer.

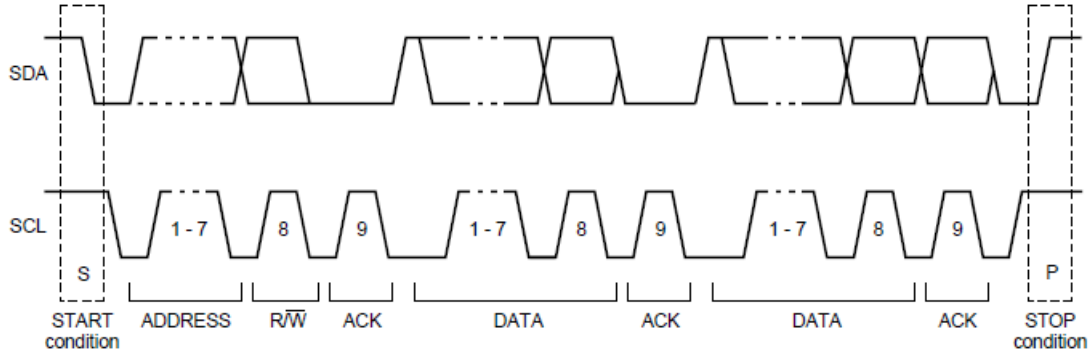


Figure 2: Complete data transfer

2.1.5 Start and Stop condition

All transactions begin with a START (S) and are terminated by a STOP (P) condition. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition. The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition. In this respect, the START (S) and repeated START (Sr) conditions are functionally identical.

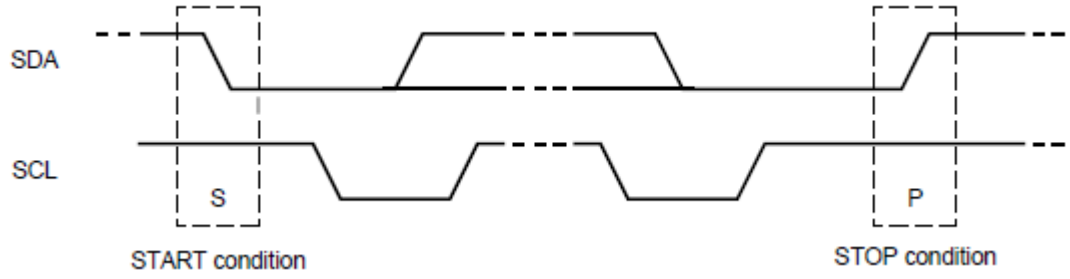


Figure 3: START and STOP conditions

2.1.6 Data validity

The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Figure 4). One clock pulse is generated for each data bit transferred.

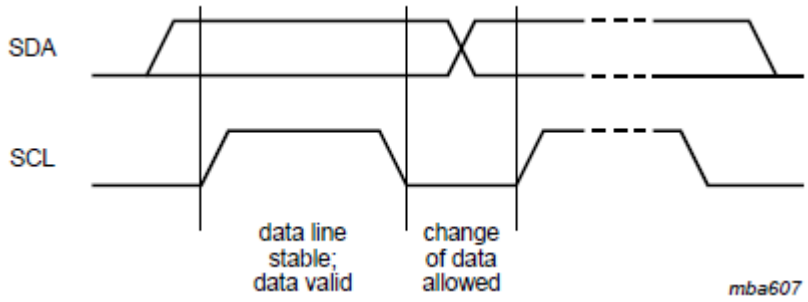


Figure 4: Bit transfer on I2C bus – data validity

2.1.7 Clock stretching

Clock stretching pauses a transaction by holding the SCL line LOW. The transaction cannot continue until the line is released HIGH again. Clock stretching is optional.

On the byte level, a device may be able to receive bytes of data at a fast rate, but needs more time to store a received byte or prepare another byte to be transmitted. Slaves can then hold the SCL line LOW after reception and acknowledgment of a byte to force the master into a wait state until the slave is ready for the next byte transfer in a type of handshake procedure

2.1.8 Write operation example

Figure 5 shows the I2C write operation example. It is very similar to Figure 2, where the transfer was described in general. On Figure 5 the R/$\overline{W}$ is set to 0, which means that the operation is write. The whole operation ends either with Slave sending a

NACK (for example when the Slave's memory is full) or by Master sending a STOP condition.

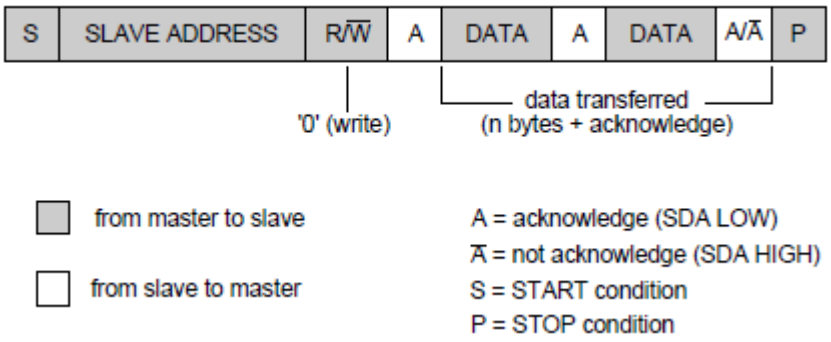


Figure 5: I2C Write operation example

2.1.9 Read operation example

Figure 6 shows the I2C Read operation example. The R/$\overline{W}$ signal is set to 1, which sets the I2C operation to read. The operation ends when the I2C Master sends NACK and Stop condition afterwards.

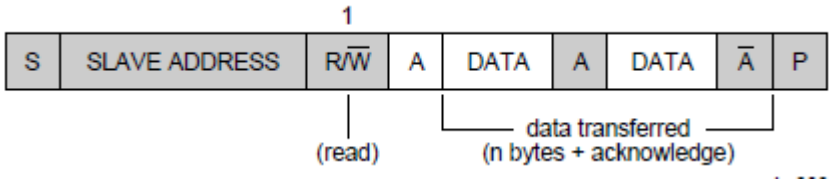


Figure 6: I2C Read operation example

2.1.10 Combined operation example

An example of two different operations is shown on Figure 7. After the first operation a Repeated Start condition is sent by the I2C Master and a new operation follows starting with the new Slave address. After all of the operations are finished, a STOP condition is sent by the I2C Master.

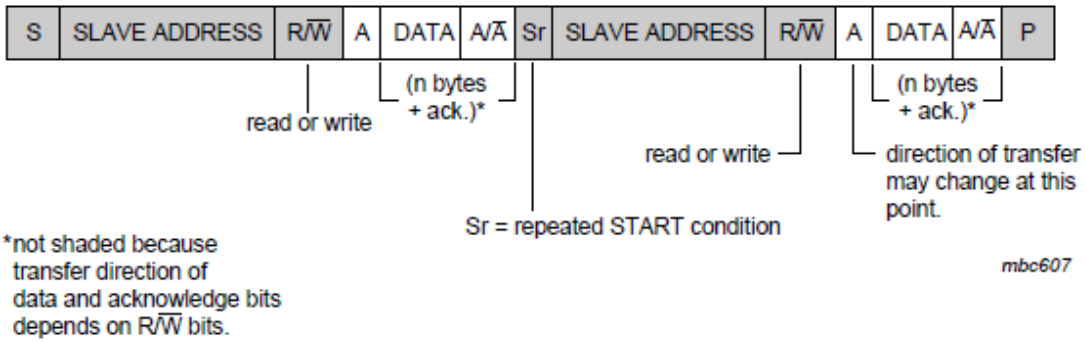


Figure 7: I2C Combined operation example

2.2 APB Protocol description

This device communicates with AMBA 3 APB Protocol. The complete documentation for this protocol can be found under (ARM, 2004).

APB is a parallel unpipelined synchronous protocol where every transfer takes at least two cycles. This APB version also includes signal PREADY which is used for extending the APB transfer by the slave device. This can be useful if the device needs more than two cycles for the transfer. Any number of extra additional cycles can be added. This means from 0 higher.

APB uses the following signals:

- Input signals: PSELx, PENABLE, PRESETn, PCLK, PWRITE, PADDR, PWDATA
- Output signals: PREADY, PSLVERR, PRDATA

2.2.1 Operating states

The APB bus can be in three different operating states as shown on Figure 8. Those states are further described under Figure 8.

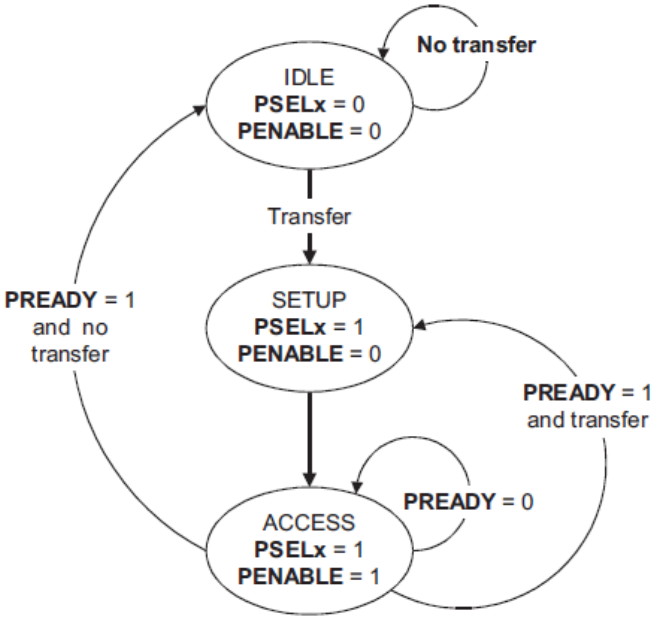


Figure 8: APB Operating states

- **IDLE** This is the default state of the APB.
- **SETUP** When a transfer is required the bus moves into the SETUP state, where the appropriate select signal, PSELx, is asserted. The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.
- **ACCESS** The enable signal, PENABLE, is asserted in the ACCESS state. The address, write, select, and write data signals must remain stable during

the transition from the SETUP to ACCESS state. Exit from the ACCESS state is controlled by the PREADY signal from the slave:

- If PREADY is held LOW by the slave then the peripheral bus remains in the ACCESS state.
- If PREADY is driven HIGH by the slave, then the ACCESS state is exited and the bus returns to the IDLE state if no more transfers are required. Alternatively, the bus moves directly to the SETUP state if another transfer follows.

2.2.2 APB Signals detailed description

Table 2: APB Signals desription

Signal	Source	Description
PCLK.	Clock source Clock	The rising edge of PCLK times all transfers on the APB.
PRESETn	System bus equivalent	Reset. The APB reset signal is active LOW. This signal is normally connected directly to the system bus reset signal.
PADDR	APB bridge	Address. This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
PSELx	APB bridge	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a PSELx signal for each slave.
PENABLE	APB bridge	Enable. This signal indicates the second and subsequent cycles of an APB transfer.
PWRITE	APB bridge	Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW.
PWDATA	APB bridge	Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH. This bus can be up to 32 bits wide.
PREADY	Slave interface	Ready. The slave uses this signal to extend an APB transfer.
PRDATA	Slave interface	Read Data. The selected slave drives this bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide.
PSLVERR	Slave interface	This signal indicates a transfer failure. APB peripherals are not required to support the PSLVERR pin. This is true for both existing and new APB peripheral designs. Where a peripheral does not include this pin then the appropriate input to the APB bridge is tied LOW.

2.2.3 Write transfer without waiting states

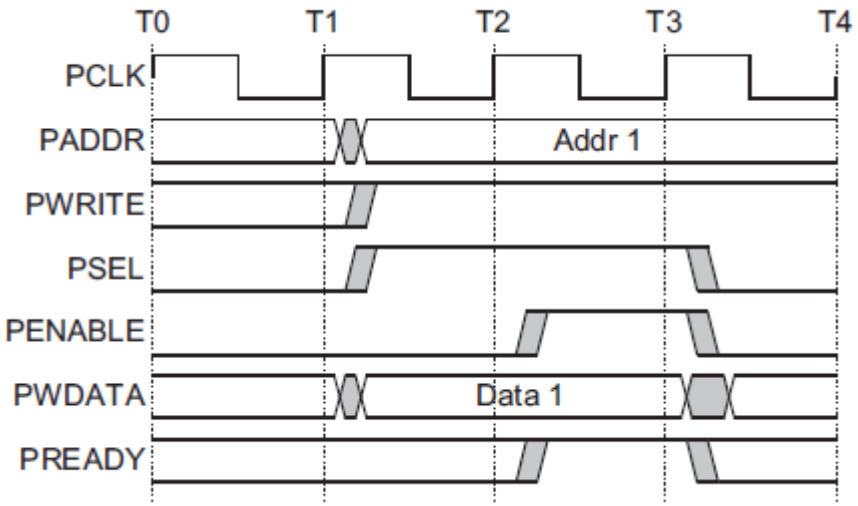


Figure 9: Write transfer without waiting states

The write transfer starts with the address, write data, write signal and select signal, which are all changing after the rising edge of the clock. After the following clock edge the enable signal is asserted, PENABLE, and this indicates that the Access phase is taking place. The address, data and control signals all remain valid throughout the Access phase. The transfer completes at the end of this cycle.

The enable signal, PENABLE, is deasserted at the end of the transfer. The select signal, PSELx, also goes LOW unless the transfer is to be followed immediately by another transfer to the same peripheral. (B.V., 2007)

2.2.4 Write transfer with waiting states

Waiting states can be used to extend the transfer. As shown on Figure 10, waiting states are used when PREADY signal is low during the transfer.

During an Access phase, when PENABLE is HIGH, the transfer can be extended by driving PREADY LOW. The following signals remain unchanged for the additional cycles:

- address, PADDR
- write signal, PWRITE
- select signal, PSEL
- enable signal, PENABLE
- write data, PWDATA.

PREADY can take any value when PENABLE is LOW. This ensures that peripherals that have a fixed two cycle access can tie PREADY HIGH.

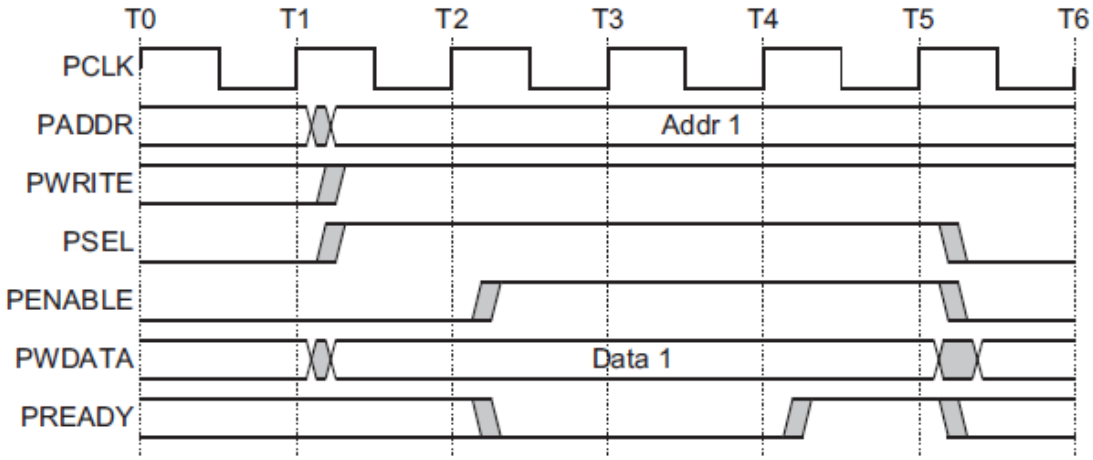


Figure 10:APB Write transfer with waiting states

2.2.5 Read transfer without waiting states

Figure 11 shows the read transfer without using wait states. The timing of the signals was already described in the write transfer paragraph above.

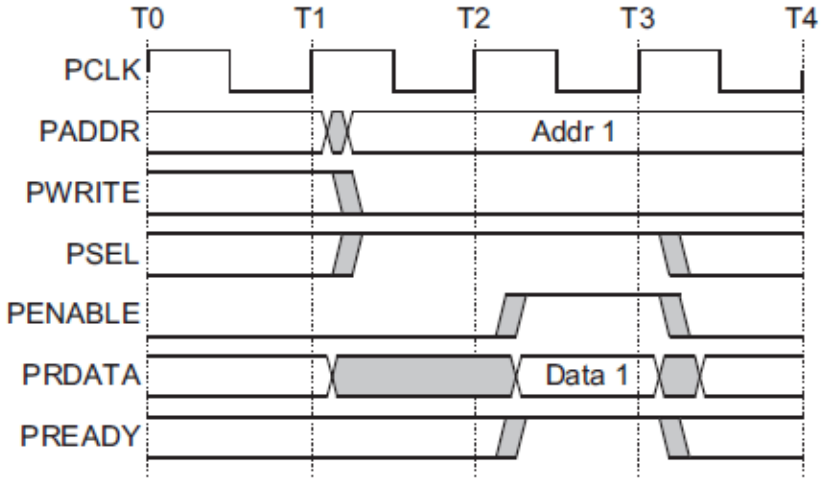


Figure 11: Read transfer without waiting states

2.2.6 Read transfer with waiting states

The transfer is extended if **PREADY** is driven LOW during an Access phase. The protocol ensures that the following remain unchanged for the additional cycles:

- address, PADDR
- write signal, PWRITE
- select signal, PSEL
- enable signal, PENABLE.

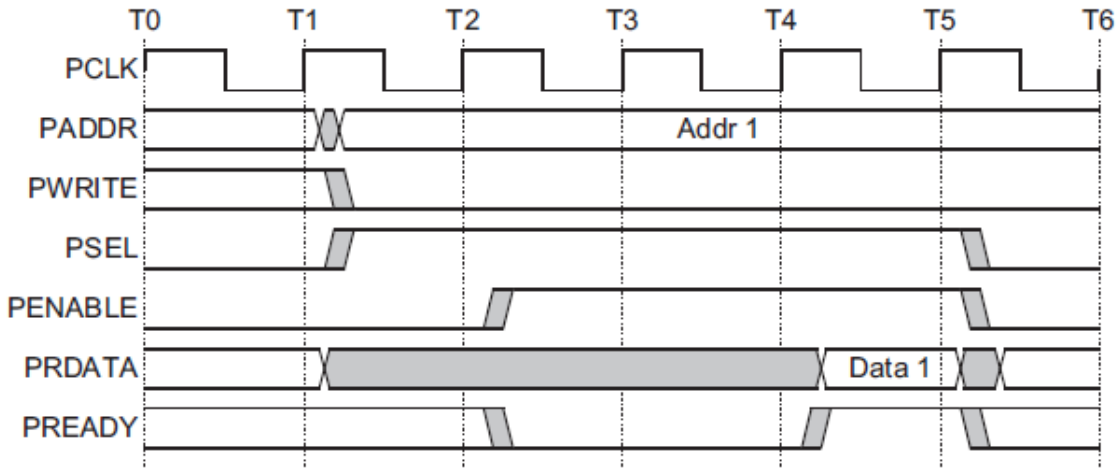


Figure 12: APB Read transfer with waiting states

3 Low-Power techniques

3.1 Low power design motivation

Challenges that cause us to deal with low power design are mainly the following:

- Increasing device density
- Increasing clock frequencies
- Lowering supply voltage
- Lowering transistor threshold voltage

High power consumption leads to higher temperatures. The goal is to keep the temperature low to avoid parasite effects. The principle of achieving this is to provide performance only when it is required.

3.2 Types of power consumption

3.2.1 Dynamic power

Dynamic power consists of internal power and switching power.

Internal power is consumed by the cells when one of the inputs changes, but the output doesn't change. Internal power results from the short-circuit (crowbar) current that flows through the PMOS-NMOS stack during a transition.

3.2.1.1 Switching power

Because the current flows only during logic transitions on the net, the long-term dynamic power consumption depends on the clock frequency (possible transitions per second) and the switching activity (presence or absence of transitions actually occurring on the net in successive clock cycles).

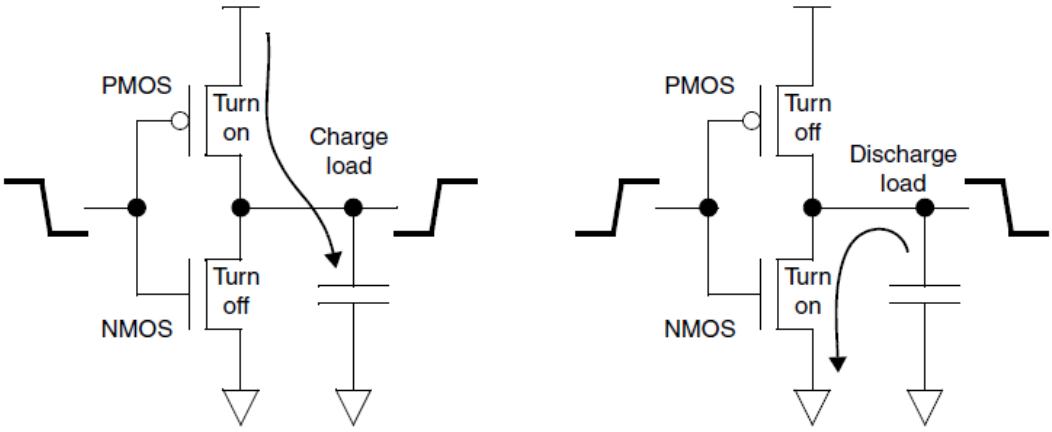


Figure 13: Switching power

The higher the clock frequency is, the more often there is activity on the transistors (change of value), because with synchronous devices activity is done with the change of clock. In other words, switching power results from the charging and discharging of the external capacitive load on the output of a cell.

These parameters can be summed in the following formula:

$P_{dyn}=C_{eff}*V_{dd}^{2}*f_{clk}$

Here we can see that the dynamic power depends on capacitance, voltage (which obviously has the greatest impact on dynamic power consumption because of the square power) and the clock frequency. The techniques described in the following text will mostly focus on how to use the voltage and frequency for lowering the power consumption.

3.2.1.2 Internal power

Internal power is consumed during the short period of time when the input signal is at an intermediate voltage level. During which, both the PMOS and NMOS transistors can be conducting. This condition results in a nearly short-circuit conductive path from VSS to ground, as illustrated in Figure 1-2. A relatively large current, called the crowbar current, flows through the transistors for a brief period of time. Lower threshold voltages and slower transitions result in more internal power consumption.

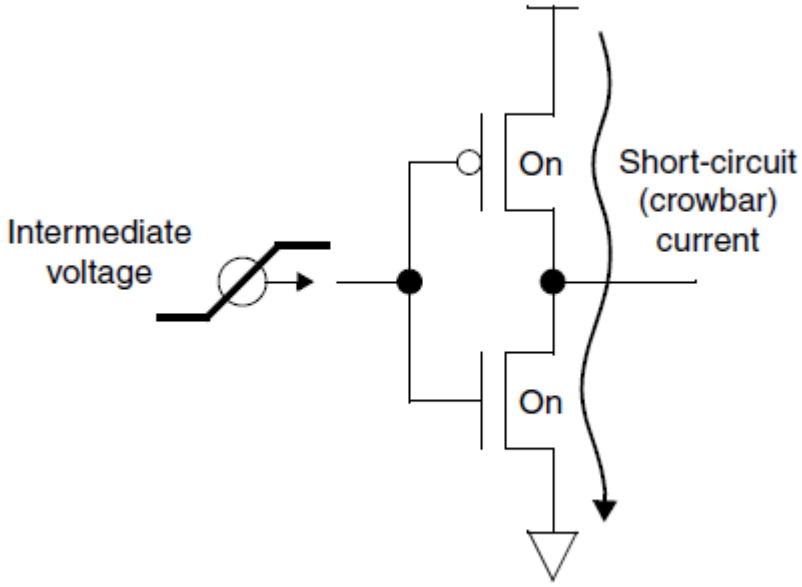


Figure 14: Internal power

(Synopsys, 2010)

3.2.2 Static (leakage) power

Static power is leakage at transistors at all times. This consumption remains at all times constant.

The main causes of leakage power are reverse-bias p-n junction diode leakage, sub-threshold leakage, and gate leakage. These leakage paths in a CMOS inverter are shown in

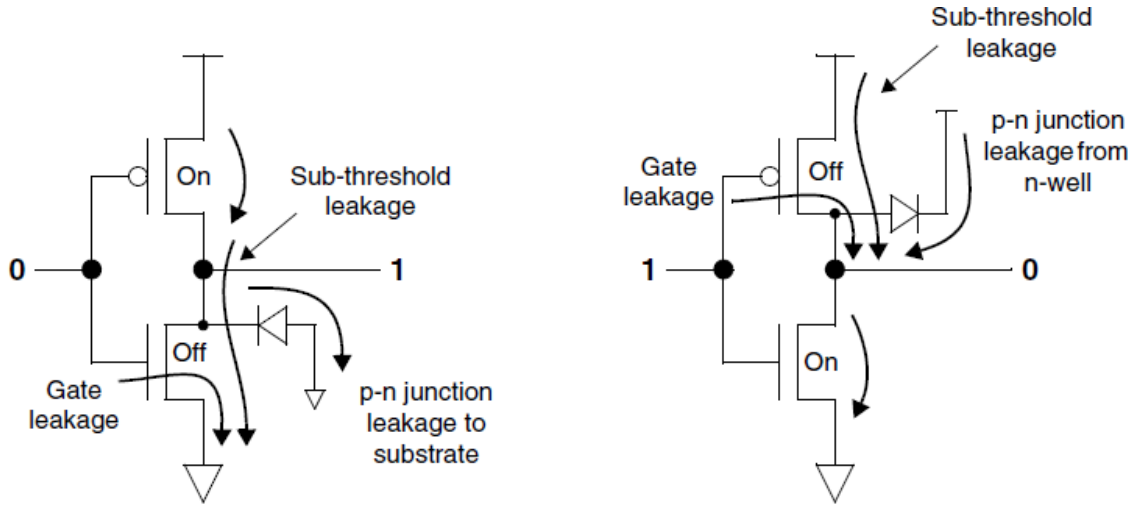


Figure 15: Static leakage currents

3.2.2.1 p-n junctions leakage

Leakage at reverse-biased p-n junctions (diode leakage) has always existed in CMOS circuits. This is the leakage from the n-type drain of the NMOS transistor to the grounded p-type substrate, and from the n-well (held at VDD) to the p-type drain of the PMOS transistor. This leakage is relatively small.

3.2.2.2 Sub-threshold leakage

Sub-threshold leakage is the small source-to-drain current that flows even when the transistor is held in the "off" state. In older technologies, this current was negligible. However, with lower power supply voltages and lower threshold voltages, "off" gate voltages are getting close to "on" threshold voltages. Sub-threshold leakage current increases exponentially as the gate voltage approaches the threshold voltage.

3.2.2.3 Gate leakage

Gate leakage is the result of using an extremely thin insulating layer between the gate conductor and the MOS transistor channel. Gate oxides are becoming so thin that only a dozen or fewer layers of insulating atoms separate the gate from the source and drain. Under these conditions, quantum-effect tunneling of electrons through the gate oxide can occur, resulting in significant leakage from the gate to the source or drain.

(Synopsys, 2010)

3.3 Low power techniques overview and comparing

There are different techniques used for low-power. The next several paragraphs are an introduction to low power techniques. The focus therefore is on comparing different techniques and their use and purpose.

Table 3: Most common low-power techniques overview

Technique	Description
Clock gating and clock tree gating	Disables blocks or clock tree parts not in use.
Multiple supply voltages (MSV, Multi Vdd), Static Voltage scaling (SVS)	Operates different blocks at different, fixed supply voltages. Also known as voltage islands. Signals that cross voltage domain boundaries are level-shifted.
Dynamic voltage scaling (DVS), Multi-level voltage scaling (MVS)	Operates different blocks at variable supply voltages. Uses look-up tables to adjust voltage on-the-fly to satisfy varying performance requirements. Signals that cross voltage domain boundaries are level-shifted.
Dynamic voltage and frequency scaling (DVFS)	Operates different blocks at variable supply voltages and frequencies. Uses look-up tables to adjust voltage and frequency on-the-fly to satisfy varying performance requirements. Signals that cross voltage domain boundaries are level-shifted.
Adaptive voltage scaling (AVS)	Operates different blocks at variable supply voltages. Uses in-block monitors to determine frequency requirements, and adjusts voltage on-the-fly to satisfy them.
Power gating or Power Shut-Off (PSO)	Turns off supply voltage to blocks not in use. Significantly reduces – but does not eliminate – leakage. Block outputs float.
Power gating with retention	Stores system state prior to power-down. Avoids complete reset at power-up, which reduces powerup/ reset delay and power consumption.
State retention power gating (SRPG)	Stores the system state in local registers. When on standby or idling, gates the clock, and the register saves the data. State retention registers use both a continuous power supply and a switchable supply. Other logic is powered only by the switchable supply, and can be powered down.
Save and restore power gating (S&RPG)	As SRPG, but uses a memory array.

(Goering, 2008)

Table 4: Low power design techniques – compared according to usage

Dynamic Power	Leakage Power	Design	Architectural
Clock gating	Multi Vt	Multi Vt	Pipelining
Variable frequency	Power gating	Clock gating	Asynchronous
Variable power supply	Back (substrate) bias	Power gating	
Multi Vdd	Use new devices-FinFet, SOI	Multi Vdd	
Voltage islands		DVFS	
DVFS			

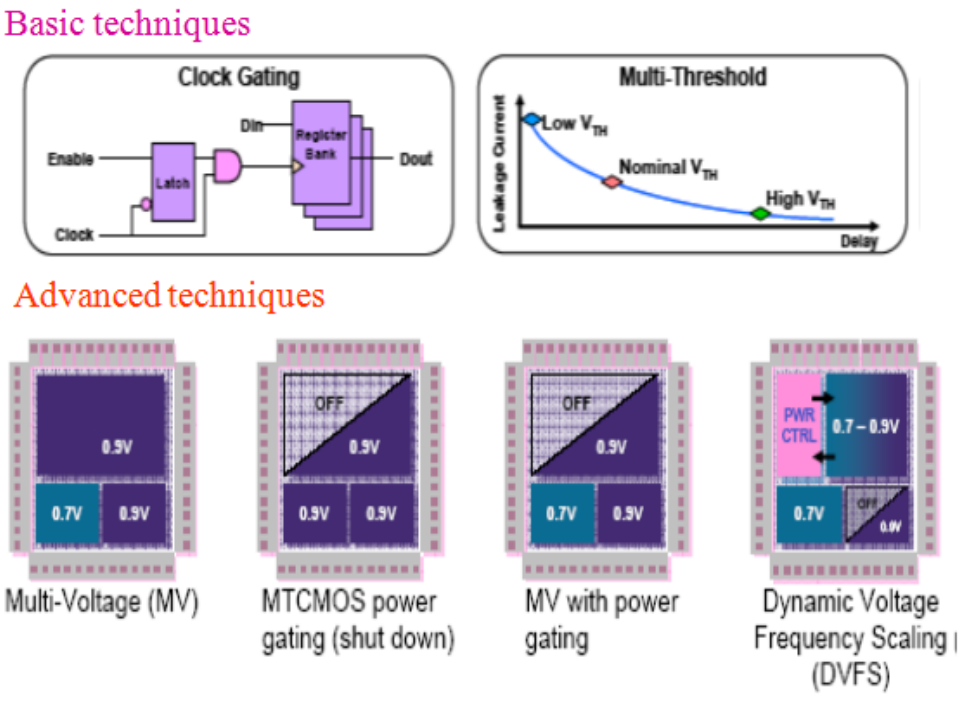


Figure 16: Low Power Techniques comparison

(Murali, 2009)

3.4 Clock-gating

RTL clock gating works by identifying groups of flip-flops which share a common enable signal. Traditional methodologies use this enable term to control the select on a multiplexer connected to the D port of the flip-flop or to control the clock enable pin on a flip-flop with clock enable capabilities. RTL clock gating uses this enable term to control a clock gating circuit which is connected to the clock ports of all of the flip-flops with the common enable term. Therefore, if a bank of flip flops which share a common enable term have RTL clock gating implemented, the flip-flops will consume zero dynamic power as long as this enable term is false.

(Frank Emnett, 2000)

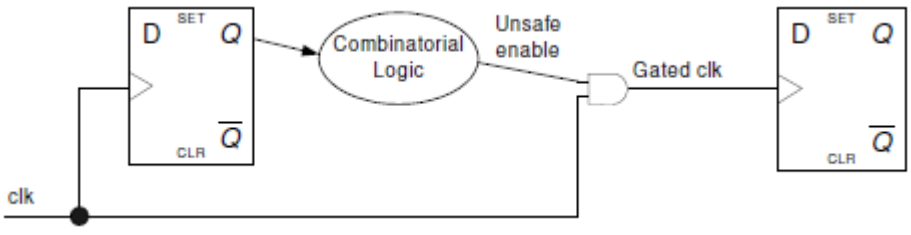
Clock gating is particularly useful for registers that need to maintain the same logic values over many clock cycles. Shutting off the clocks eliminates unnecessary switching activity that would otherwise occur to reload the registers on each clock cycle. The main challenges of clock gating are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

Clock gating is relatively simple to implement because it only requires a change in the netlist. No additional power supplies or power infrastructure changes are required.

(Synopsys, 2010)

Clock-gating lowers average power consumption; however it always increases the maximum immediate consumption. Therefore it is convenient to use clock-gating only for registers that have their enable signal mostly disabled. It is important to do an analysis of use of different registers and apply clock-gating only on those where it's suitable. Usually it is recommended to have at least 3-4 flip-flops with the same common enable signal for making clock-gating effective. In case of using clock-gating for less than 3 flop-flops with the same enable signal it can have an effect of increased consumption.

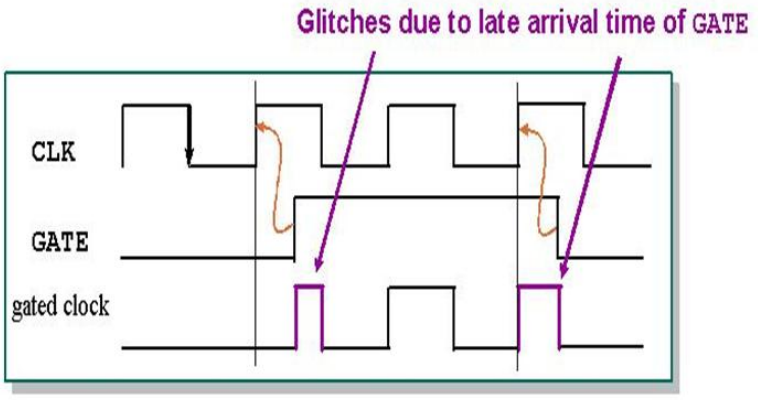
(Bečvář, 2011)



(Bečvář, 2011)

Figure 17: Principle of clock-gating connection (not completely correct)

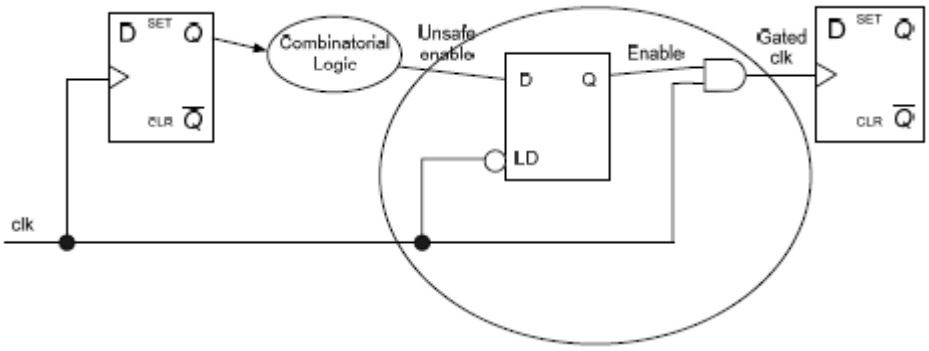
Figure 17 shows the principle of clock-gating. The AND gate is enabling the clock. This is not a correct connection though, because with having the AND gate it will cause a glitch impulse on the gated clock instead of the right clock impulse as shown on Figure 18.



(Murali, 2009)

Figure 18: Glitches in latch free clock gating

Therefore a level-sensitive latch is used with the AND gate inside the clock gating cell from a library which needs to be used. The use of the cell is shown on Figure 19. The latch holds the enable signal from the active edge of the clock until the inactive edge of the clock.



(Bečvář, 2011)

Figure 19: Correct clock-gating cell connection – connection in a dont_touch cell

Clock gating effects only dynamic power consumption as it is dependent on preventing clock activity.

3.4.1 Automatic clock gating done by Synthesis tools / Clock gating

Synthesis tools can detect low-throughput data paths where clock gating can be used with the greatest benefit, and can automatically insert clock-gating cells in the clock paths at the appropriate locations.

(Synopsys, 2010)

Automatic clock gating uses so called functional gating – input and output values of the flip flop are compared and if they are different, the clock enable signal is enabled. A big advantage of automatic clock gating during synthesis is that it only needs a change of one command to enable clock gating use.

3.4.2 Manual clock gating / Clock tree gating

Manual clock gating is done by the IP designer by manually setting the enable signal for a set of flip flops in the FSM. This enable signal is propagated through a clock gating cell. Usually different state modes are used.

3.5 Miltiple-Vt

Some CMOS technologies support the fabrication of transistors with different threshold voltages (Vt values). In that case, the cell library can offer two or more different cells to implement each logic function, each using a different transistor threshold voltage. For example, the library can offer two inverter cells: one using low-Vt transistors and other using high-Vt transistors.

A low-Vt cell has higher speed, but higher sub-threshold leakage current. A high-Vt cell has low leakage current, but less speed. The synthesis tool can choose the appropriate type of the cell to use based on the tradeoff between speed and power. For example, it can use low-Vt cells in the timing-critical paths for speed and high-Vt cells everywhere else for lower leakage power.

(Synopsys, 2010)

3.6 Multi Vdd

Different parts of a chip might have different speed requirements. For example, the CPU and RAM blocks might need to be faster than a peripheral block. A lower supply voltage reduces power consumption but also reduces speed. To get maximum speed and lower power at the same time, the CPU and RAM can operate with a higher supply voltage while the peripheral block operates with a lower voltage, as shown in Figure 20.

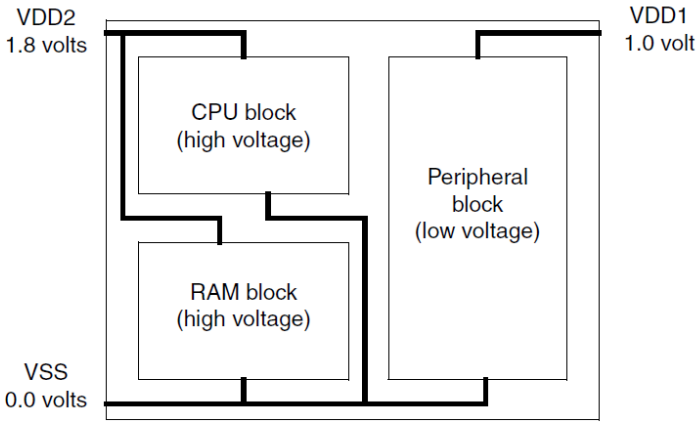


Figure 20: Multi Vdd blocks connection

(Synopsys, 2010)

3.6.1 Level Shifters

Level shifters are used for transferring data between two blocks with different power voltage as shows Figure 21.

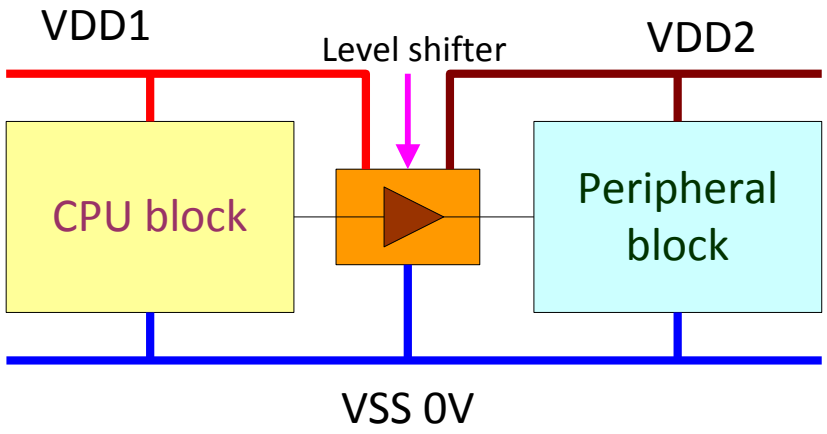


Figure 21: Blocks with different Level shifter

In any multi-voltage design, level shifters are required at the interfaces of blocks operating at different voltages. It is much easier to design one direction level shifters.

(Murali, 2009)

In theory, the bus interface of CPU can be a higher or lower voltage, for practical reason the bus is always operate at a voltage higher than or equal to the CPU. Otherwise system errors occur.

(Yang, 2008)

3.7 Multi-level voltage scaling (MVS), Dynamic voltage scaling (DVS)

This is an extension of Multi Vdd case where a block or subsystem is switched between two or more voltage levels. Only a few, fixed, discrete levels are supported for different operating modes.

3.8 Dynamic voltage and frequency scaling (DVFS)

DVFS is an extension of MVS where a larger number of voltage levels are dynamically switched between to follow changing workloads.

Timing/Voltage Values: DVFS uses a set of discrete voltage / frequency pairs. Determining which values to support is a key design decision, application dependent. Too few operating points results in systems that spend too much time ramping between levels. Too many levels results in the power supply spending too much time "hunting" between different target voltages.

Switching Times and Algorithms: Switching performance levels takes time for both voltage regulators and clock generators. Switching voltage levels is particular slow and switching frequencies is orders of magnitude faster than voltage level switching. Increase the voltage first and decrease the voltage after the frequency is lowered.

(Yang, 2008)

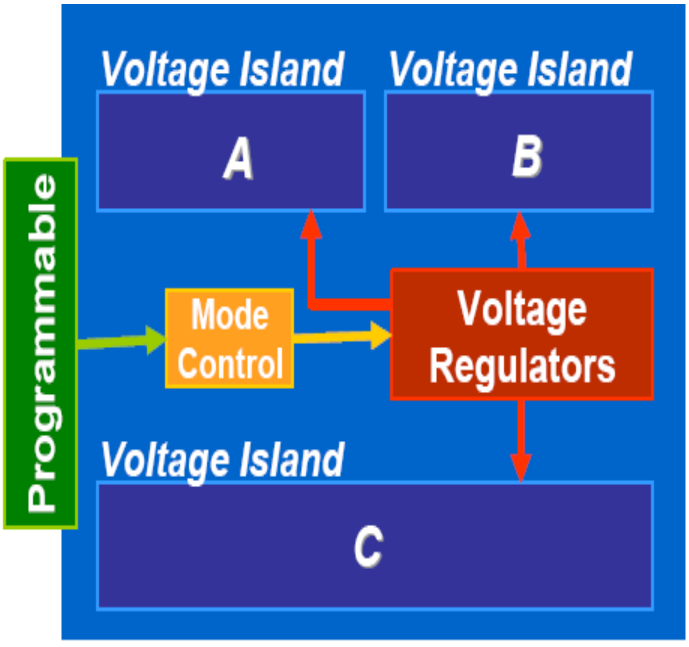


Figure 22: DVFS blocks

Mode control block - Voltage as well as frequency is dynamically varied as per the different working modes of the design.

Voltage regulators block - When high speed of operation is required, voltage is increased to attain higher speed of operation with the penalty of increased power consumption.

(Murali, 2009)

The principle of multivoltage operation can be extended to allow the voltage to be changed during operation of the chip to match the current workload. For example, a math processor chip in a laptop computer might operate at a lower voltage and lower clock frequency during simple spreadsheet computations, thereby saving power; and then at a higher voltage and higher clock frequency during 3-D image rendering when the highest performance is needed. The changing of supply voltage and operating frequency during operation to meet workload requirements is called dynamic voltage and frequency scaling.

The chip and voltage supply can be designed to use a number of established levels, or even a continuous range. Dynamic voltage scaling requires a multilevel power supply and a logic block to determine the best voltage level to use for a given task. Design, implementation, verification, and testing of the device can be especially challenging because of the ranges and combinations of voltage levels and operating frequencies that must be analyzed and accommodated.

Dynamic voltage scaling can be combined with power switching technology so that each block in the design can operate at multiple voltage levels for different performance requirements, or shut off completely when not needed at all.

(Synopsys, 2010)

3.9 Adaptive voltage scaling (AVS)

AVS is an extension of DVFS where a control loop is used to adjust the voltage. Performance Monitor is integrated with IP is monitoring to get the best thermal tracking. The performance monitor communicates with a power controller which in return sets the voltage of the power supply.

(Yang, 2008)

AVS contains voltage areas with variable software controlled VDD. Monitors in each block communicate with the mode controller that controls Voltage regulators as shows in Figure 23.

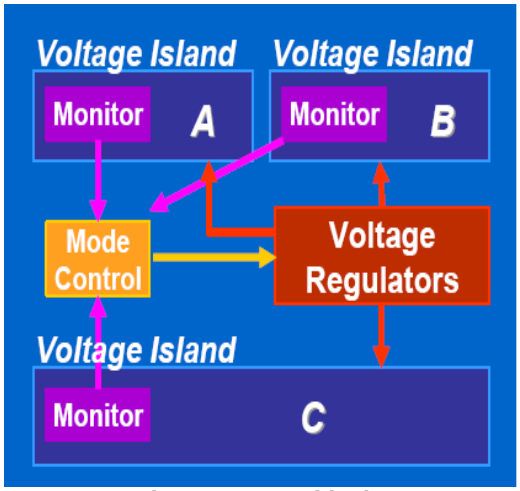


Figure 23: AVS blocks

(Murali, 2009)

3.10 Power gating (Power Switching)

3.10.1 How Power gating works

Power gating circuit blocks that are not in use are temporarily turned off. On the other hand, this increases time delays as power gated modes have to be safely entered and exited. The shutting down of these blocks is done by either hardware timers or software drivers.

(Murali, 2009)

Power switching has the potential to reduce overall power consumption substantially because it lowers leakage power as well as switching power. It also introduces some additional challenges, including the need for a power controller, a power-switching network, isolation cells, and retention registers.

(Synopsys, 2010)

3.10.2 Ways how to shut down blocks

There are different ways how to safely shut down blocks:

- Software or hardware
  - Driver software schedules the power down operations
  - Hardware timers are used
- Dedicated power management controller
- Switch off by using external power supply for long term
- Use CMOS switches for smaller duration switch off
- A power switch (either to VDD – header switch, PMOS or GND – footer switch, NMOS) is added to supply rails to shut-down logic. MTCMOS switches are used.

3.10.3 Power switches

A block that can be powered down must receive its power through a power-switching network, consisting of a larger number of transistors with source-to-drain connections between the always-on power supply rail and the power pins of the cells. The power switches are distributed physically around or within the block. The network, when switched on, connects the power to the logic gates in the block. When switched off, the power supply is effectively disconnected from the logic gates in the block.

High-Vt transistors from a Multiple-Threshold CMOS (MTCMOS) technology are used for the power switches because they minimize leakage and their switching speed is not critical. PMOS header switches can be placed between VDD and the block power supply pins, or NMOS footer switches can be placed between VSS and the block ground pins, as shown in Figure 1-8. The number, drive strength, and placement of switches should be chosen to give in an acceptable voltage drop during peak power usage in the block.

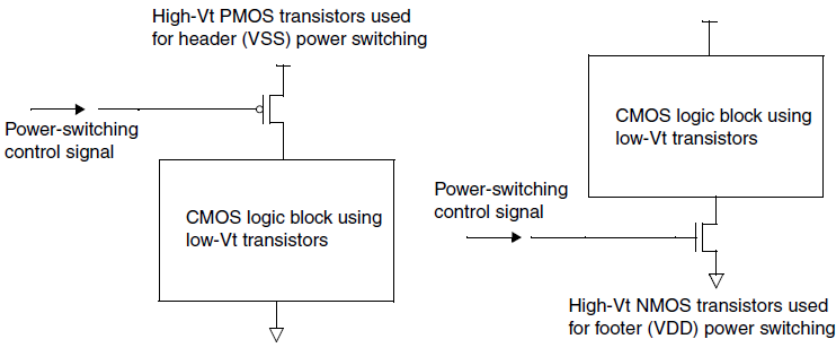


Figure 24: Power-switching Network Transistors

(Synopsys, 2010)

3.10.4 Isolation cells

Isolation cells isolate the power gated block from the always-on-block. It can hold logic 1 or logic 0 or it can hold the signal value latched at the time of the power-down event. Isolation cells must be powered during power-down periods to hold the saved value.

Any use of power switching requires isolation cells where signals leave a powered-down block and enter a block that is always on (or currently powered up). An isolation cell provides a known, constant logic value to an always-on block when the power-down block has no power, thereby preventing unknown or intermediate values that could cause crowbar currents.

One simple implementation of an isolation cell is shown in Figure 25. When the block on the left is powered up, the signal P_UP is high and the output signal passes through the isolation cell unchanged (except for a gate delay). When the block on the left

is powered down, P_UP is low, holding the signal constant going into the always-on block. Isolation cells must themselves have power during block power-down periods.

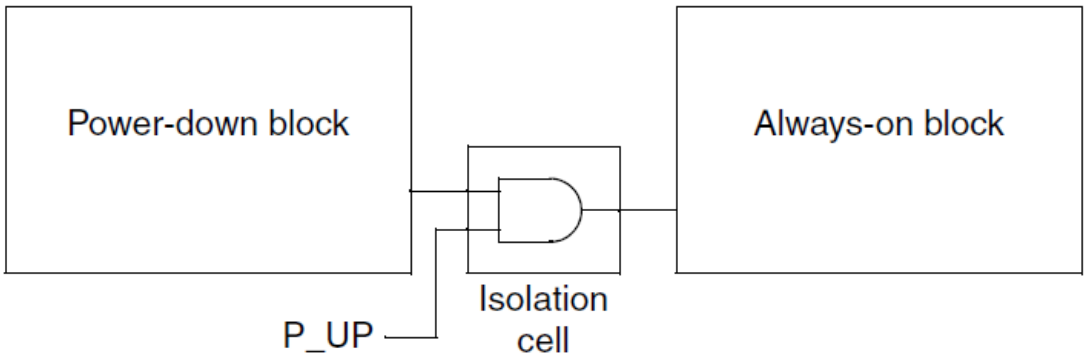


Figure 25: Use of isolation cell

(Synopsys, 2010)

3.10.5 Enable level shifter

An enable level shifter acts as a level shifter and an isolation cell at the same time. This is shown on Figure 26. That means that the interface cells between different blocks must perform both level shifting and isolation functions.

(Murali, 2009)

The power switching can be combined with multi voltage operation. Different blocks can be designed to operate at different voltages and also to be separately powered down when they are not needed. In that case, the interface cells between different blocks must perform both level shifting and isolation functions, depending on whether the two blocks are operating at different voltages or one is shut down. A cell that performs both functions is called an enable level shifter. This cell must have two separate power supplies, just like any other level shifter.

(Synopsys, 2010)

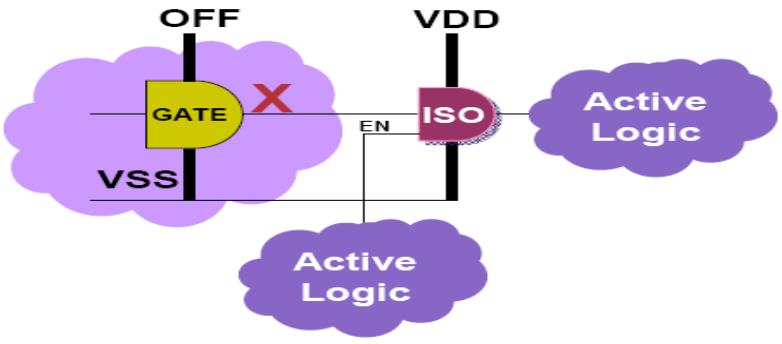


Figure 26: Level shifter

(Murali, 2009)

3.10.6 Retention registers

Retention registers are always powered up. Special low leakage flip-flops are used to hold the data of the main register of the power gated clock. A power gating controller controls the retention mechanism.

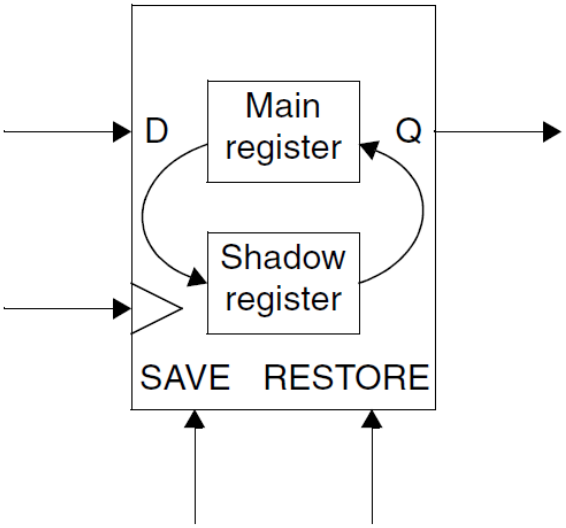


Figure 27: Retention register

When a block is powered down and then powered back up, it is often desirable for the block to be restored to the state it was in prior to the power-down event. A possible strategy is to use retention registers in the power-down block. A retention register can retain data during power-down by saving the data into a shadow register (also known as the bubble register) prior to power-down. Upon power-up, it restores the data from the shadow register to the main register. The shadow register has an always-on power supply, but it is constructed with high-Vt transistors to minimize leakage during the power-down period. The main register is built with fast but leaky low-Vt transistors.

One type of retention register implementation is shown in Figure 27. The SAVE signal saves the register data into the shadow register prior to power-down and the RESTORE signal restores the data after power-up. Instead of using separate, edge-sensitive SAVE and RESTORE signals, a retention register could use a single level-sensitive control signal.

A retention register occupies a larger area than an ordinary register, and it requires an always-on power supply connection for the shadow register in addition to the power-down supply used by the rest of the device. However, restoring the data to the registers after power-up is fast and simple compared with other strategies.

(Synopsys, 2010)

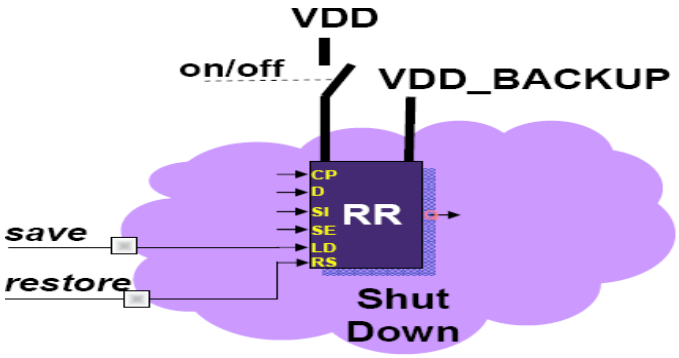


Figure 28: Connection of retention register signals

3.10.7 Always on logic

There's always some logic that needs to stay active during the shut-down period. The basic principle is shown on Figure 29. Examples of always-on-logic are the following:

- Internal enable pins (ISO/ELS)
- Power switches
- Retention registers
- User-specific cells

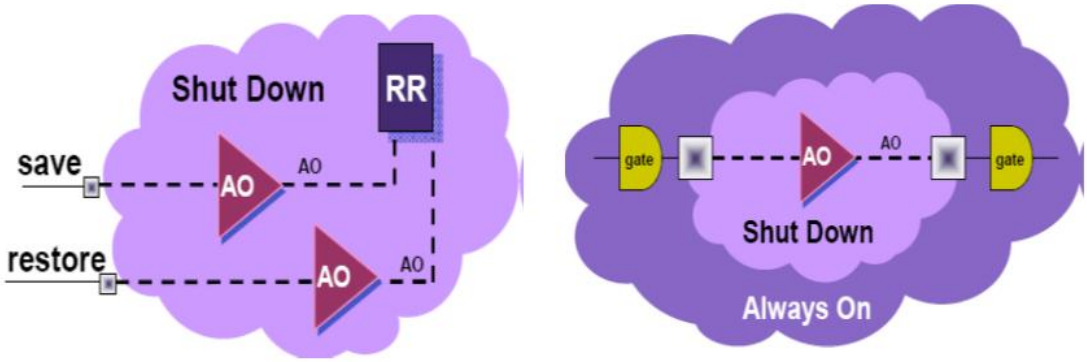


Figure 29: Always on logic

(Murali, 2009)

3.11 Conclusion of the listed low-power techniques

3.11.1 Clock gating and clock tree gating

Clock gating (automatic clock gating during synthesis) is a very easy but at the same time effective way how to implement a low-power technique in the design. The only thing that needs to be done is changing one command in the synthesis script. This method is often used.

Clock tree gating on a level by manually placing clock gating cells on RTL level is a way that can be used when the designed knows the power consumption modes of the device and approximately how much time the device spends in these modes.

These techniques show to be useful in the IP developed in this project.

3.11.2 Multi Vdd, SVS

These techniques are used as techniques in the physical design. This technique is used in SoC design to provide different voltages for different voltage islands.

3.11.3 DVS, MVS, DVFS, AVS

These techniques are an extension of Multi Vdd technique. Again, it's a matter of physical design and they're used in SoCs.

3.11.4 Power gating, Power Shut-Off

This is a technique used in physical design. Multiple-Vt transistors are usually used for this technique. It requires use of different extra blocks and the assignment would be too complicated.

3.11.5 Pipelining

Pipelining is an architectural technique used with advantage in processors. However it is not useful in this kind of design that my master's project is focused on.

3.11.6 Asynchronous design

Asynchronous design is a advanced and hard-to-design technique. It is not suitable for this kind of design.

3.11.7 Conclusion

Clock gating and Clock tree gating turns out to be the best implementable and useable technique in this design although it does effect only dynamic power consumption.

Techniques such as Multi Vdd, SVS, DVS, MVS, DVFS and AVS are used for SoCs mainly. This IP core is however not a SoC. Techniques like DVFS are also quite complicated, work with more consumption modes and are used in much bigger projects than this.

Power gating is focused on physical design and would not provide comparable results after synthesis.

Pipelining and asynchronous design are not suitable for this kind of architecture.

Therefore clock gating and clock tree gating will be used in the design. To be able to compare all the different clock gating methods and make the results more interesting, I decided to use the next four different clock gating methods:

- No use of clock gating
- Automatic clock gating (done during synthesis by synthesis tool)
- Manual clock gating (Clock tree gating)
- Manual + automatic clock gating

These four different kinds of the use of clock gating will be further used and their power consumption results compared in this document.

4 Design and Verification flow

4.1 Introduction

A design flow is a sequence of steps that had to be done during the design development of an IP. These steps are approximately similar for lots of projects; however there's usually something specific in each of them. For this project it meant to be able to get four different physical designs according to the type of clock gating that was used.

The design flow is a complex process of steps. The flow in Figure 30 shows how complicated this process is.

For the purpose of this project the typical S3 Group design flow was the start point, however it needed to be changed for this special purpose as some characteristics of this project are unique. The modified flow that was actually used is described in Figure 30. This flow was setup specifically for this project by creating four different run directories as four different variations of clock gating were used.

Chapters 4.3 to 4.12 describe the different steps of the design flow. The description contains what had to be developed, designed and done in those steps. Scripts has to be used for most of these steps to automate the development, however these scripts had to be changed and adjusted.

The possibility of being able to do my master's project at the S3 Group gave me the unique opportunity to go through these steps and learn how to work through them and learn the work in the tools that are used for each of the steps. I have never done most of those steps before as I only worked with FPGAs before.

4.2 Design and verification flow diagram

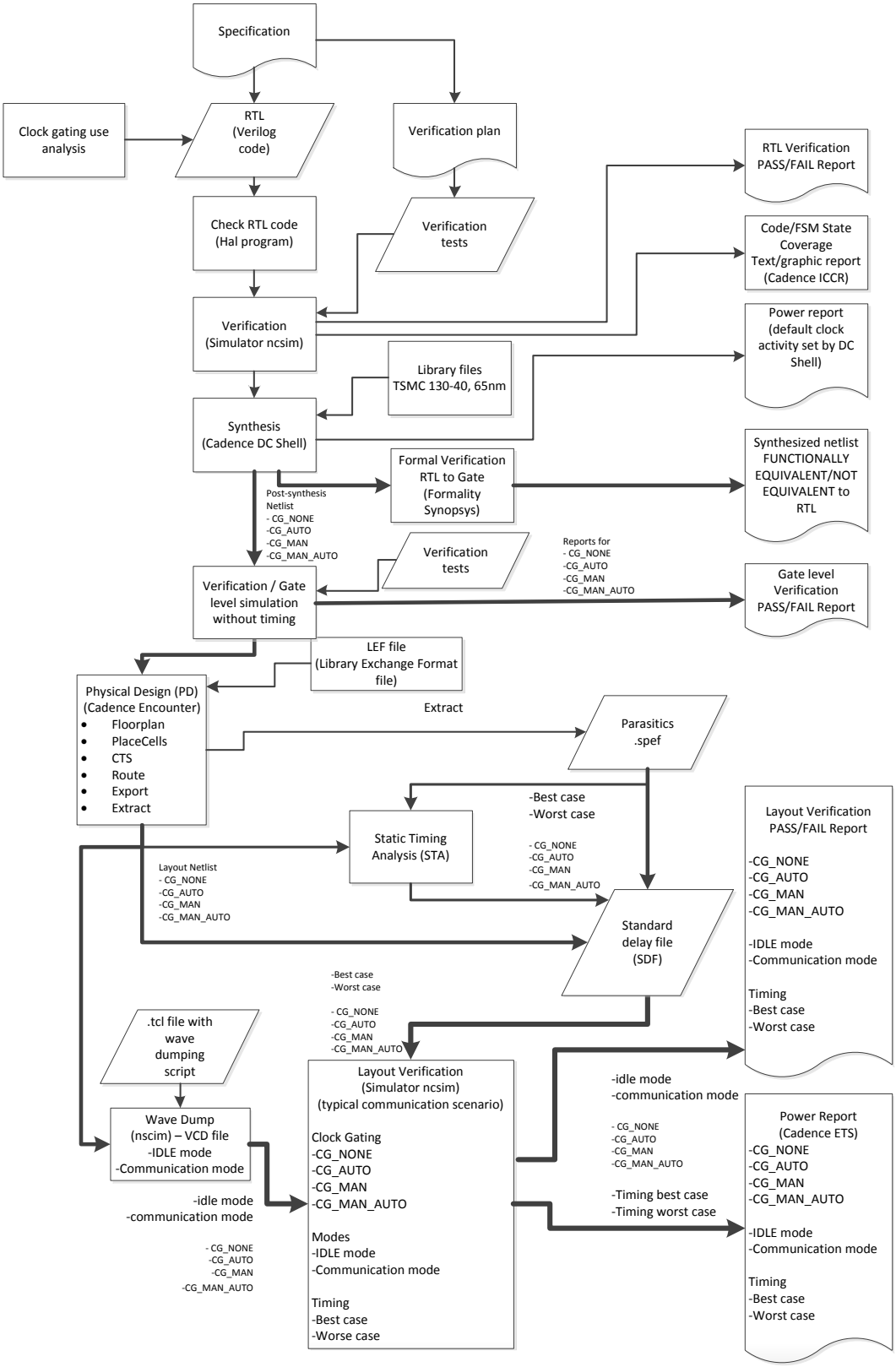


Figure 30: Design and Verification flow diagram

4.3 Specification

4.3.1 General description

This IP block is a device that enables the communication with I2C bus on one side and with AMBA 3 APB bus on the other.

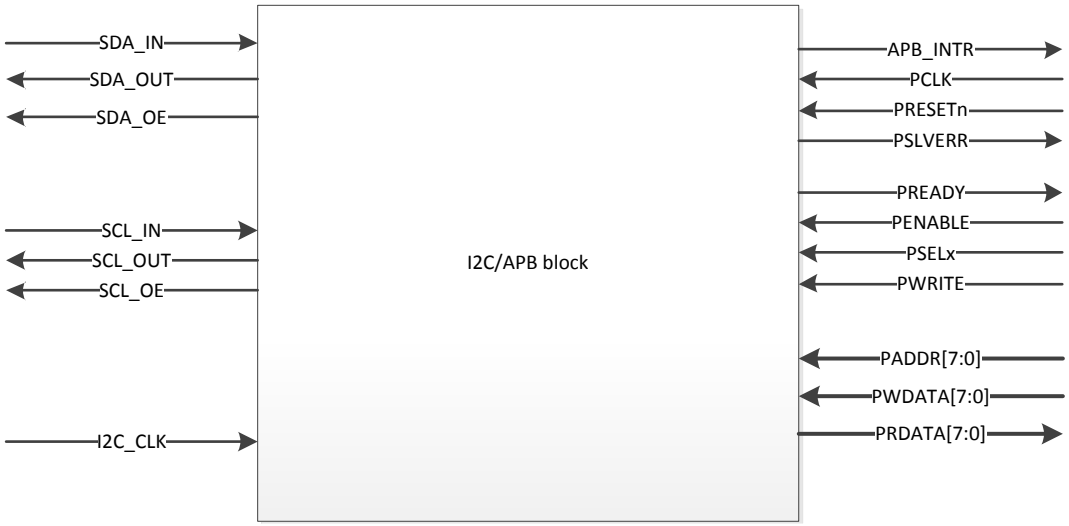


Figure 31: Top-level schema of the I2C/APB Block

The signals SDA_IN, SDA_OUT, SDA_OE are connected to a PAD before being connected to the I2C bus signal SDA. In the same sense are also signals SCL_IN, SCL_OUT, SCL_OE connected to another PAD to drive the SCL signal.

Table 5: Top-level I / O Port list

Port name	Direction	Function	Connected to
SDA_IN_i	Input	Serial Data Line Input	I2C
SDA_OUT_o	Output	Serial Data Line Output	I2C
SDA_OE_o	Output	Serial Data Line Output Enable	I2C
SCL_IN_i	Input	Serial Clock Line Input	I2C
SCL_OUT_o	Output	Serial Clock Line Output / Clock stretching	I2C
SCL_OE_o	Output	Serial Clock Line Output Enable/ Clock stretching enable	I2C
I2C_CLK_i	Input	I2C Block Clock	I2C
APB_INTR_o	Output	APB Interrupt	APB
PREADY_o	Output	APB Slave Ready for transfer	APB
PENABLE_i	Input	APB Enable	APB
PSELx_i	Input	APB Slave Device Selected	APB
PRESETn_i	Input	Global Reset	APB
PLCK_i	Input	APB Block Clock	APB
PWRITE_i	Input	APB read/write operation	APB
PADDR_i	Input	APB Address	APB
PWDATA_i	Input	APB Data Input	APB
PRDATA_o	Output	APB Data Output	APB

The I2C frequency needs to be in the following relationship with the APB frequency: $f_{I2C} \geq f_{APB}$ to ensure the correct function of the device.

4.3.2 Typical usage / Typical communication scenario

This device serves for the I2C Master to get information from an APB Bridge. Therefore the typical communication has the next several steps:

1. I2C Master writes data (that include request description) in I2C Slave
2. APB part of the DP device puts the interrupt signal on high according to the interrupt mask register
3. APB Bridge reads the interrupt register, recognizes a request (data in Fifo). APB Slave sends a signal to I2C Slave to reset the interrupt state signals (Start bit, Selected bit,…).
4. APB Bridge reads data from the DP device
5. APB Bridge sends an answer by writing data in DP device
6. I2C Master reads data by accessing I2C Slave DP device

4.3.3 Other functions of the DP device except the typical communication scenario

The DP device has also the following functions:

- Change of I2C Slave address by APB Bridge
- Read/Write mask in APB Interrupt Mask register by APB Bridge
- Read APB Interrupt register by APB Bridge

4.3.4 Register map

The access to the device from I2C Master is defined by the I2C standard, where the device needs to be first addressed, then the master chooses the operation (read/write) and afterwards the data is transferred. There are only two operations that the I2C Master can do – read and write data.

On the other hand, the access from the APB has a signal for read/write operation and also a bus for addressing an operation. Data can be written in the device and read from the device. The addresses with the operations of the device are fully adjustable in the dp_s_global_consts.v file. If no changes are made to this file, you can access the operations through the following addresses:

Table 6: Register map table

APB address	Register name	Width	Reset value	Bit functions	Note
000	FIFO_RX	8	00000000	[7:0] – read data from FIFO	RO
001	INTR_REG	8	00000000	[7] – selected_bit [6] – start_bit [5] – stop_bit [4:3] – error 00 – no error 01 – error during read op. 10 – error during write op. 11 – unspecified error [2] – fifo_rx_not_empty [1] – fifo_rx_full [0] – fifo_tx_ full	RO, COR RO, COR RO, COR RO, COR RO RO RO
010	FIFO_TX	8	00000000	[7:0] data_wr	WO
011	I2C SLAVE ADDR	8	00000000		WO
100	INTR_MASK_REG	8	11111111	[7] – selected_bit [6] – start_bit [5] – stop_bit [4] – not used/for future use [3] – error [2] – fifo_rx_not_empty [1] – fifo_rx_full [0] – fifo_tx_ full	

Since the start bit isn't very accurate when it comes to the fact that if the device is actually asked to communicate, there's also a selected bit. The selected bit serves for detecting that the I2C Slave has been successfully addressed and the address matches with its address.

4.3.5 Top level description

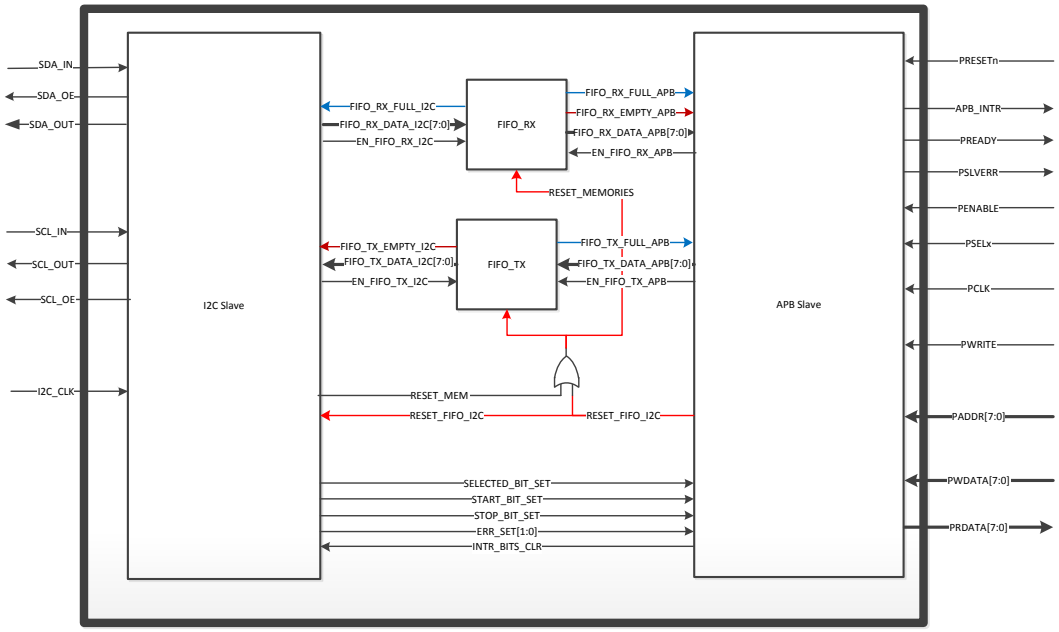


Figure 32: Top-level schema of I2C/APB Blocks

Figure 32 shows the connections between the I2C and APB blocks and the FIFOs that are used for transmitting data between these two blocks. The basics of this communication are pretty easy to understand – the data itself is transmitted only through the synchronous FIFOs which have different clocks for both read and write operations. Other than this there are signals for indicating start-bit, stop-bit, selected-bit, error bits and a signal for clearing these signals. These signals that are not transferred through a FIFO are synchronized to make sure the signals are transmitted correctly.

4.3.6 Functional descriptions

4.3.6.1 Design feature list

- Compatible with Philips I2C bus standard
  - Clock stretching generation
  - I2C communication error detection (interrupt on APB side)
- Compatible with ARM APB 3.0 bus standard
  - Interrupt poutput (Fifo TX full, Fifo RX full, Fifo RX not empty, I2C communication error, I2C Start bit, I2C Stop bit, I2C Slave Selected)
  - Interrupt masking on all interrupt bits
- 8bit data transfers
- Fifo Memories reset after I2C communication error detection
- $f_{I2C} \geq f_{APB}$

4.3.6.2 Reset description

The PRESETn_i signal coming from the APB bridge is used as a global reset for the whole device.

The APB block of the device generates the signal RESET_FIFO_I2C which is also used as a reset for both the FIFOs and the I2C block in case when the APB block receives a command to change the I2C Slave address. Then both of the FIFOs are emptied (by reset), I2C Slave set to reset and a new address is written to the I2C Slave block through TX fifo.

The reset signal RESET_MEM is generated from the I2C Slave block, which is used to empty both FIFOs in case an I2C communication error occurs. In that case an error bit is also set.

4.3.6.3 Setting I2C Slave default address

The I2C Slave device can have a default address. This address will be set every time after the PRESETn_i signal occurs, if the default address is not equal to Zero. The default address is defined as a parameter of the IP block instantiation. This means that if more than one instance of the DP device is instantiated in a design, each of these instances can have a different default I2C Slave address.

If the default address parameter is set to 0 (Zero), the default address is not used and the I2C Slave waits to get an address from APB.

The default address is always saved to the I2C block from the APB block through TX FIFO. This is because the I2C block is reset with every address change as well as the memories.

4.3.6.4 Setting of the I2C Slave address

Setting of the I2C Slave address (if the default I2C Slave address wasn't used) is done the same way as the change of the I2C Slave address. This is described in 4.3.6.5.

4.3.6.5 Change of the I2C Slave address

The address of the device can be changed by the APB command (APB address) PADDR_CHANGE_I2C_ADDR and writing the new address to PWDATA signals.

4.3.6.6 I2C Communication error detection

There's a certain chance that an error in the I2C communication can occur. This error is detected by the device if a start or stop condition comes in a time that it's not supposed to.

For example, that could mean that the device is transmitting data and it suddenly comes to a start/stop condition. The device then generates an error, the I2C block sets itself to the IDLE state where it expects new commands, resets the FIFOs and writes what kind of error occurred. The APB part of the device then signalizes an interrupt and it's up to the APB Bridge to read the APB Status register and do any further actions.

I2C Slave announces the following error alternatives:

- I2C_NO_ERROR
- I2C_READ_ERROR
- I2C_WRITE_ERROR
- I2C_UNSPECIFIED_ERROR

These constants are set in the dp_s_global_consts.v file.

4.3.7 I2C

The I2C Block of the device consists of a standard connection of two blocks - a Moore FSM and a Data Unit.

4.3.7.1 Functions

The I2C Slave device can only execute requests it receives from a master, which are receiving data from the master and sending data to the master. If we look at it from the master's side – read data from the I2C slave and write data in the I2C slave. It does not do any other actions. The way the I2C Slave address is set has been described in chapter 4.3.6.3.

4.3.7.2 I2C Slave block diagram

Figure 33 shows how the I2C Slave FSM and I2C Slave data unit are connected. It is a standard connection of a FSM and Data Unit. Data unit provides state signals for FSM and FSM sets control signals for the Data Unit. Since both FSM and Data Unit can send output to SDA, there's a multiplexor controlled by the FSM to determine which of these outputs goes to the SDA_OUT signal.

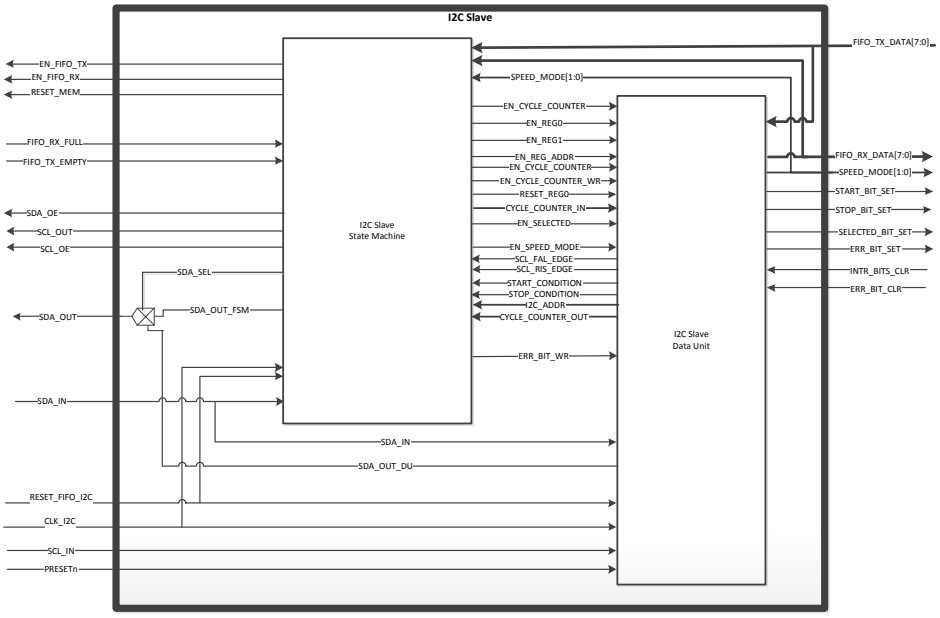


Figure 33: I2C Slave block diagram

4.3.7.3 FSM

The FSM Diagram for I2C Slave is displayed on Figure 34. Since a text description of this diagram could be confusing, I decided to put together Table 7 that describes what each state serves for and what the next states are and under what condition the transition is done.

The I2C communication is a serial bit communication and is therefore quite exact when each bit is set. This made it challenging to design the FSM. Values can be changed only in certain intervals when the SCL is low.

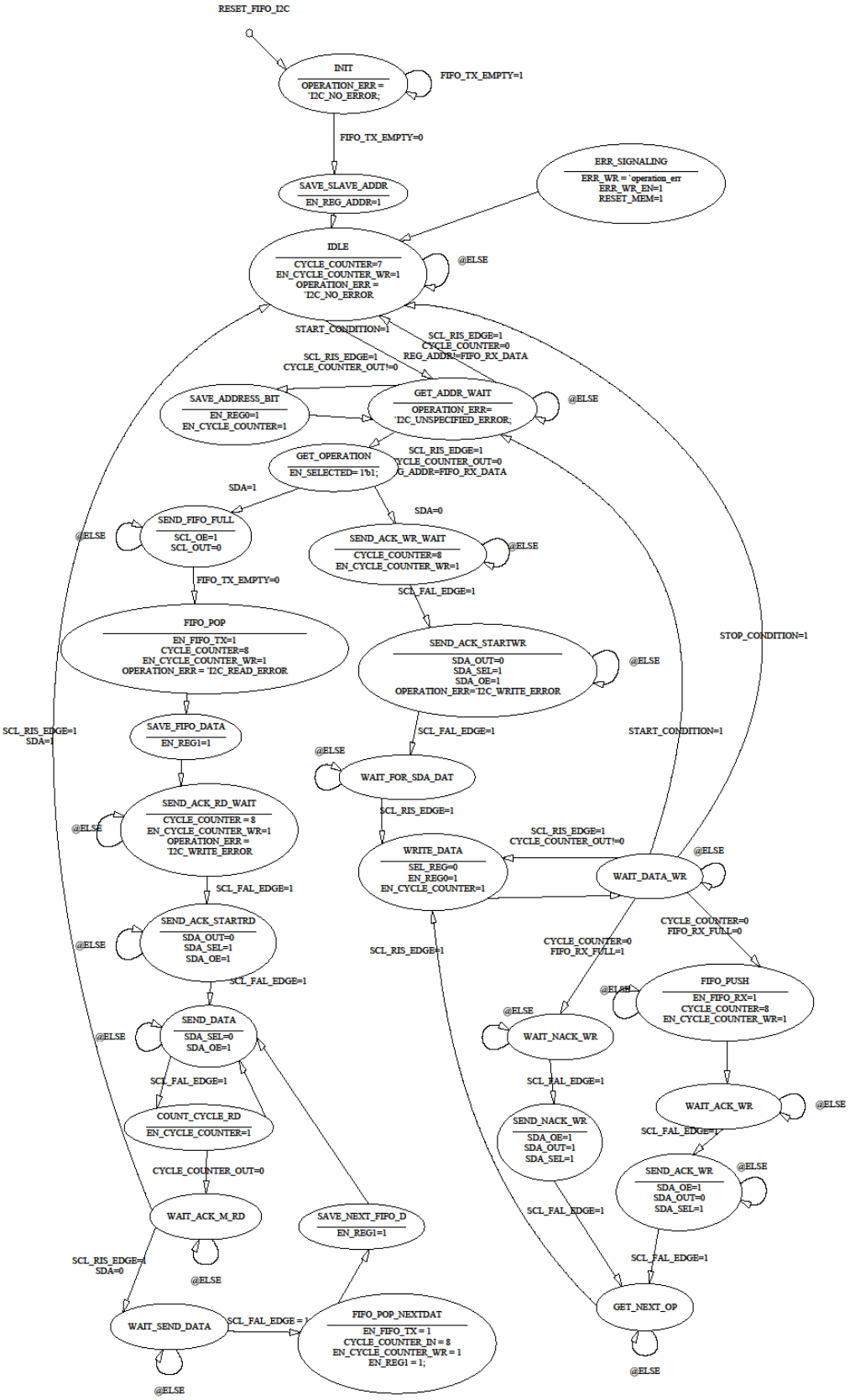


Figure 34: I2C FSM Diagram

Table 7: I2C FSM States

State name	Function	Next state
INIT	Initial state, waiting for I2C Slave address to be in TX Fifo	SAVE_SLAVE_ADDR
SAVE_SLAVE_ADDR	Save I2C Slave address	IDLE
IDLE	Idle state, waiting for the addressing by I2C Master	GET_ADDR_WAIT (if Start condition)
GET_ADDR_WAIT	Wait for SCL rising edge till all address bits are received	SAVE_ADDRESS_BIT – after SCL rising edge and not all 7bits of I2C Slave address received yet GET_OPERATION – after all 7bits of I2C Slave address are saved and match with the I2C Slave address that this device is using IDLE – after all 7bits of I2C Slave address are saved and they do not match with the I2C Slave address that this device is using GET_ADDR_WAIT - Otherwise
SAVE_ADDRESS_BIT	Save the I2C Slave bit that I2C Master is addressing the device with	GET_ADDR_WAIT
GET_OPERATION	Recognize the operation (read/write)	SEND_FIFO_FULL – if read operation SEND_ACK_WR_WAIT – if write operation
SEND_FIFO_FULL	Waits till TX Fifo in not empty (filled of some data)	FIFO_POP – if TX fifo filled with some data
FIFO_POP	Pops next data from TX fifo	SAVE_FIFO_DATA
SAVE_FIFO_DATA	Saves data from TX fifo to REG1 (see Figure 35 for more details)	SEND_ACK_RD_WAIT
SEND_ACK_RD_WAIT	Waits till SCL falling edge	SEND_ACK_START_RD
SEND_ACK_START_RD	Sends ACK to I2C Master	SEND_DATA
SEND_DATA	Sends one bit of data	COUNT_CYCLE_RD
COUNT_CYCLE_RD	Enables cycle counter to the next bit cycle	SEND_DATA – if not all bits sent yet to I2C Master WAIT_ACK_M_RD – if all bits sent to I2C Master
WAIT_ACK_M_RD	Decide if another Byte transaction follows	IDLE – if no other Byte transaction is followed WAIT_SEND_DATA – if another Byte transaction is followed
WAIT_SEND_DATA	Wait for falling edge of SCL to send the next Byte	FIFO_POP_NEXT_DATA
FIFO_POP_NEXT_DATA	Pops out next data from TX fifo	SEND_ACK_WR_WAIT
SAVE_NEXT_FIFO_DATA	Saves data from TX fifo to REG1	SEND_DATA
SEND_ACK_WR_WAIT	Wait for next SCL fall edge to send ACK	SEND_ACK_START_WR
SEND_ACK_START_WR	Sends ACK to write operation	WAIT_FOR_SDA_DATA

WAIT_FOR_SDA_DATA	Wait till next SCL rising edge to read the data after	WRITE_DATA
WRITE_DATA	Store data in Reg0	WAIT_DATA_WR
WAIT_DATA_WR	Decides if all bits are stored and according to that saving data to RX Fifo. Also after data is processed there's a transition to IDLE and GET_ADDR_WAIT state	WRITE_DATA – if not all data bits received yet FIFO_PUSH – all data bits received and RX fifo not full SEND_NACK_WR – if all data bits received, but fifo RX full IDLE – after stop condition (data processed) GET_ADDR_WAIT– after start condition (data processed)
FIFO_PUSH	Saves received data to RX Fifo	WAIT_ACK_WR
WAIT_ACK_WR	Waits till SCL falling edge	SEND_ACK_WR
SEND_ACK_WR	Sends ACK to I2C Master	GET_NEXT_OP
WAIT_NACK_WR	Waits till SCL falling edge	SEND_NACK_WR
SEND_NACK_WR	Send NACK to I2C Master	GET_NEXT_OP
ERR_SIGNALLING	Signals errors in I2C communication	IDLE
GET_NEXT_OP	Waits for SCL rising edge to get next operation (either write next data, repeated start or end of operation)	WRITE_DATA

4.3.7.4 Data Unit

The Data unit serves for storing the data and detecting different conditions. The list of all registers described in the Data Unit Diagram in Figure 35 with their functions is described in Table 8.

The following conditions are also detected by the Data unit:

- Start/stop condition detection – flip-flops (SDA_CURR, SDA_PREV, SCL_CURR, SCL_PREV) are used as a synchronizer. They compare the current and previous values of these signals and these signals detect the start or stop condition by an AND.
- SCL Rising edge detection - flip-flops (SCL_CURR, SCL_PREV) with and AND detect the rising edge of the SCL signal.
- SDA Rising edge detection - flip-flops (SDA_CURR, SDA_PREV) with and AND detect the rising edge of the SCL signal.

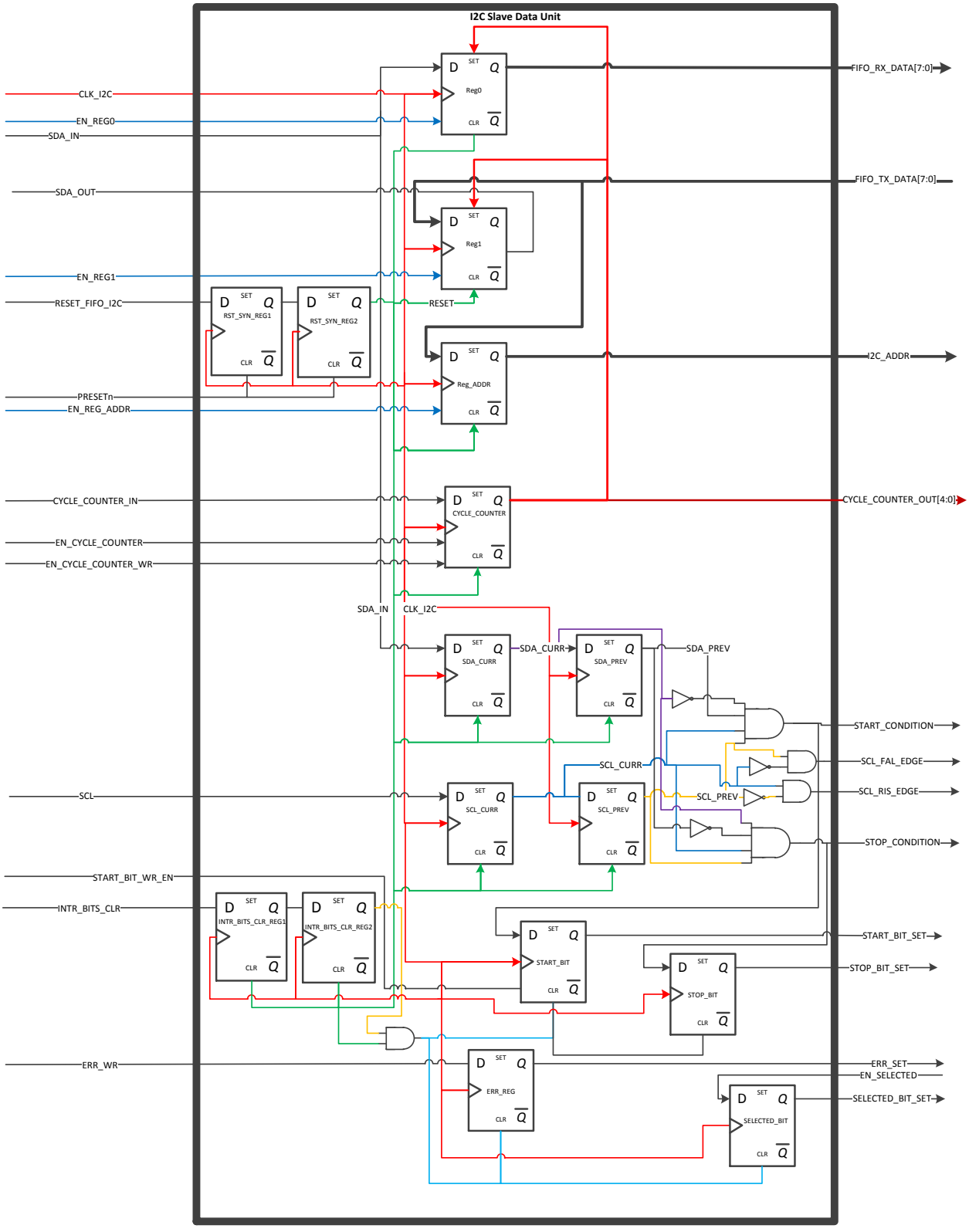


Figure 35: I2C Slave Data Unit

Table 8: I2C Registers list

Name	Function
CYCLE_COUNTER	Cycle counter for counting bit positions during I2C communication, addresses bits in Reg0 and Reg1 according to cycle number
ERR_REG	Storing type of error occurred in I2C communication
INTR_BITS_CLR_REG1	Resynchronization register for clearing interrupt bits
NTR_BITS_CLR_REG2	Resynchronization register for clearing interrupt bits
Reg_ADDR	Storing I2C Slave address
REG0	Storing bits coming from I2C Write command
REG1	Storing data from TX FIFO used for I2C Read command
RST_SYN_REG1	Resynchronization register for reset
RST_SYN_REG2	Resynchronization register for reset
SCL_CURR	Current SCL value
SCL_PREV	Previous SCL value
SDA_CURR	Current SDA value
SDA_PREV	Previous SDA value
SELECTED_BIT	I2C Slave selected bit (interrupt bit for APB)
START_BIT	Start condition bit (interrupt bit for APB)
STOP_BIT	Stop condition bit (interrupt bit for APB)

4.3.8 APB

The basics of this protocol were already described in chapter 2.2. The complete documentation that was used for the APB design can be found under (ARM, 2004). The APB device implemented in this design is a APB Slave.

4.3.8.1 Functions, modes

The APB bus is a parallel addressed as well as data bus. Address and data busses are each separated. The device can provide operations read/write data, read device status and change I2C Slave address. More concrete description of addressing these operations was described in chapter 4.3.4.

4.3.8.2 Block diagram

The structure of the APB block of the device shown in Figure 36 is traditional – there is a FSM and a data unit, which are connected together. Except the usual connection of FSM and standard unit, there's also a multiplexor used for determining whether the input of TX fifo is the I2C Slave default address or data from PWDATA.

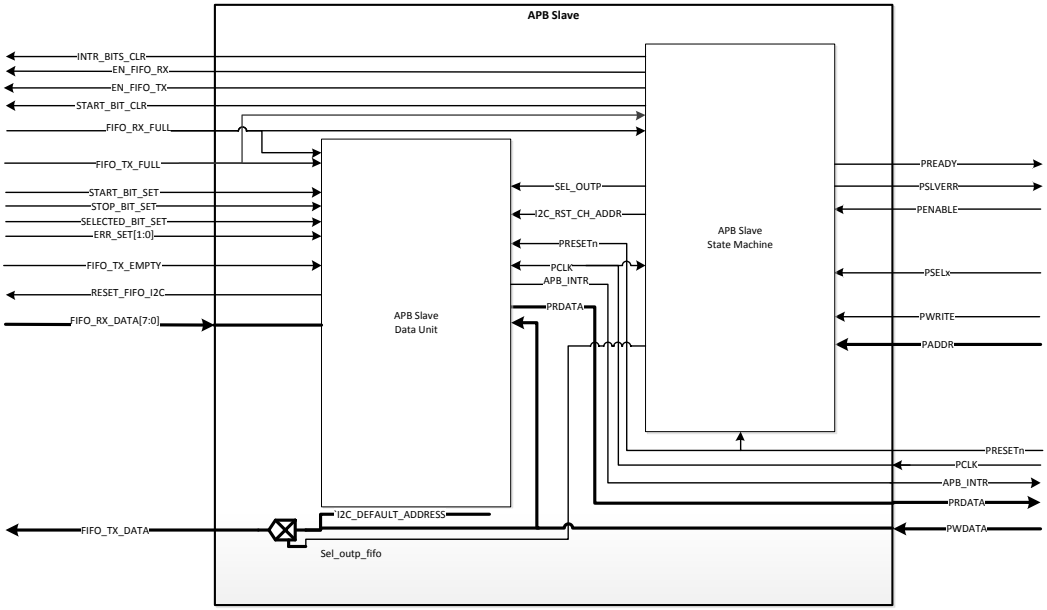


Figure 36: APB Block diagram

4.3.8.3 FSM

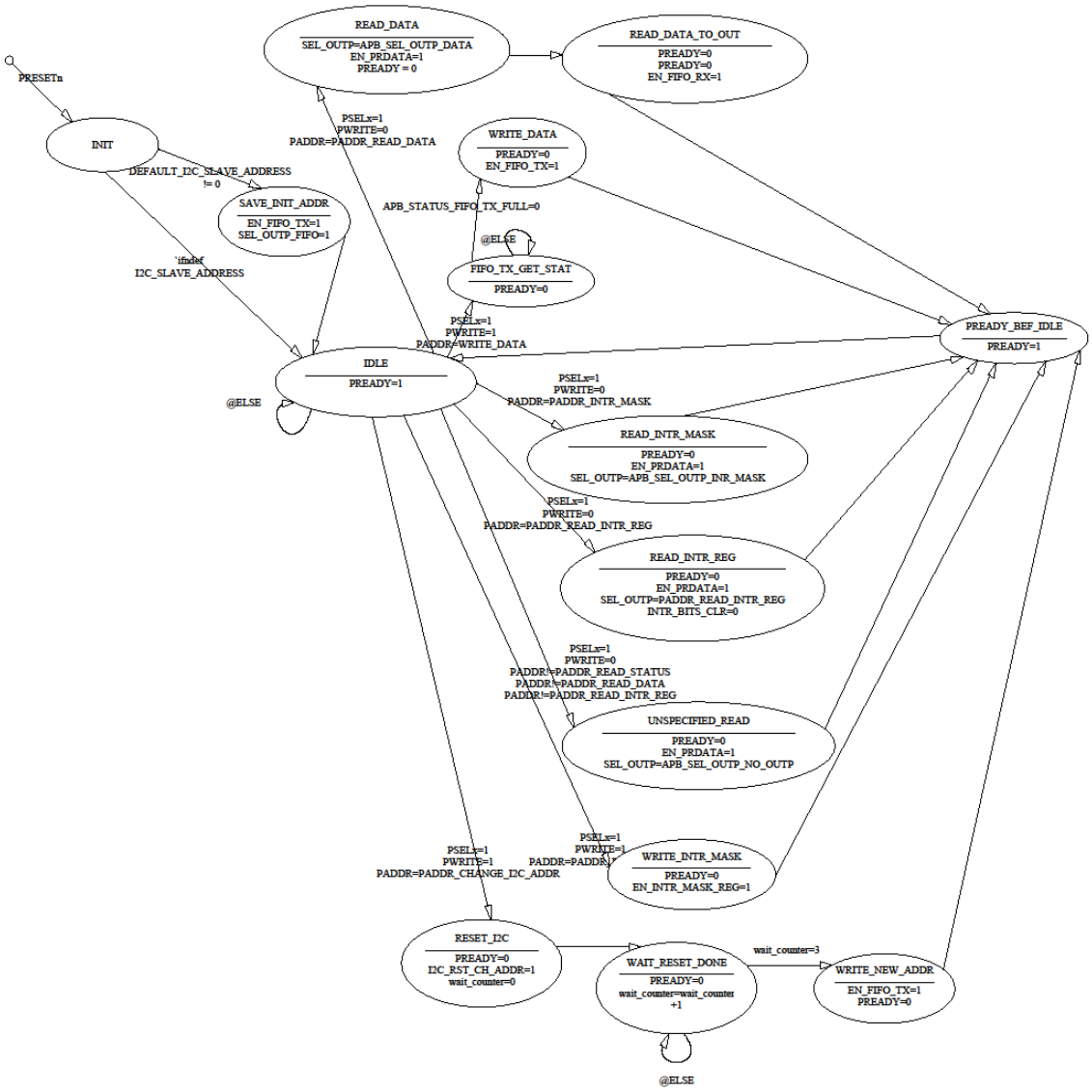


Figure 37: APB FSM Diagram

I2C FSM Diagram is described in Figure 37. A detailed description of the states is in Table 9. The most outstanding state is the IDLE state. The device stays in this state whenever it's waiting for a command from APB Bridge. All operations start from the IDLE State on request from the APB Bridge.

Table 9: APB FSM States

State name	Function	Next state
INIT	Init state after reset	SAVE_INIT_ADDR - used if default i2c address set IDLE - used if default i2c address not set
SAVE_INIT_ADDR	Saves the Default I2C Slave Address to the I2C Slave	IDLE
IDLE	Idle state	Various, see Figure 37
READ_DATA	Save data at TX fifo output to prdata_o register	READ_DATA_TO_OUTPUT
READ_DATA_TO_OUTPUT	Enables the next data in TX fifo to output	PREADY_BEFORE_IDLE
PREADY_BEFORE_IDLE	Pready on high, but APB FSM not in the IDLE state yet to prevent premature operation recognition	IDLE
RESET_I2C	Resets the I2C Slave + memories, sets wait counter to zero	WAIT_RESET_DONE
WAIT_RESET_DONE	Waits several cycles before saving the new I2C Slave address to TX Fifo to let the FIFOs get ready	WRITE_NEW_ADDR
WRITE_NEW_ADDR	Saves the new I2C Slave address to TX Fifo	PREADY_BEFORE_IDLE
FIFO_TX_GET_STATUS	Waits as long as TX Fifo is full	WRITE_DATA
WRITE_DATA	Saves data to TX Fifo	PREADY_BEFORE_IDLE
WRITE_INTR_MASK	Writes a new Mask to the Interrupt mask register	PREADY_BEFORE_IDLE
READ_INTR_REG	Saves the content of the interrupt register to prdata_o register, which means that the data from interrupt register gets to output. Deletes set interrupt bits in I2C Slave (intr_bits_clr_o <= 1'b0 because of inverted logic)	PREADY_BEFORE_IDLE
READ_INTR_MASK	Saves the Interrupt mask to prdata output	PREADY_BEFORE_IDLE
UNSPECIFIED_READ	Puts all Zeros to output	PREADY_BEFORE_IDLE

4.3.8.4 Data Unit

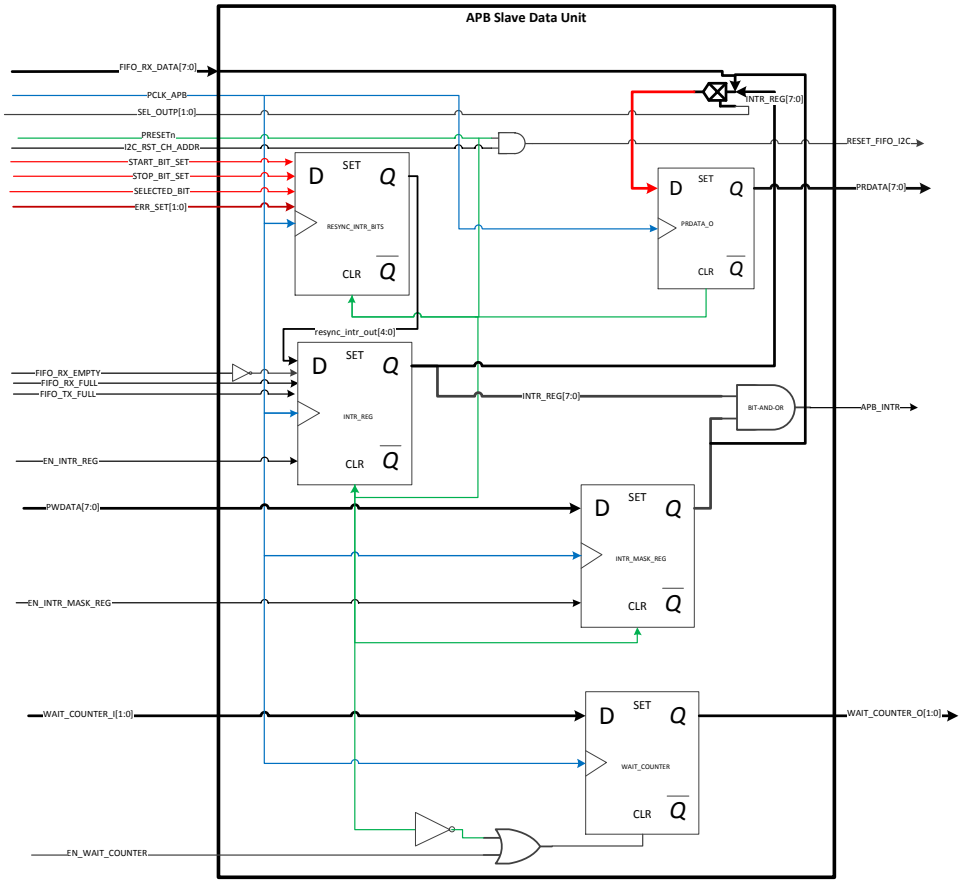


Figure 38: APB Block Data Unit

The APB Data unit presented in Figure 38 contains only a few registers, this is caused by the simplicity in which the APB transactions are done. The output PRDATA of the device is registered. Further detailed description of the registers is to be found in Table 10.

Table 10: APB Registers list

Name	**Function**
INTR_MASK_REG	Interrupt mask register
INTR_REG	Interrupt register
PRDATA_O	PRDATA registered output
RESYNC_INTR_BITS	Resynchronization cell for signals from I2C clock domain
WAIT_COUNTER	Waiting for memory + I2C reset to be done after change of I2C Slave address

4.3.9 FIFOs

Two FIFOs both of the size 16x8 bytes are used in the design.

4.3.10 Clock requirements

4.3.10.1 Minimum I2C Slave frequency

To be able to count the minimum I2C Slave frequency, the maximum amount of clock ticks which the I2C FSM needs during SCL high and SCL low needs to be know. The minimum length of the high and low signals is given by the I2C standard in (B.V., 2007). Knowing these facts, we divide the minimum high and low length of these signals by the amount of clocks that need to be done in the I2C FSM and we get two lengths of signals, from which we count the frequency. The higher frequency of these two frequencies is the minimum frequency that the I2C Slave can operate with.

The I2C FSM needs 4 cycles during SCL high (transitions between states GET_OPERATION, SEND_FIFO_FULL, FIFO_POP, SAVE_FIFO_DATA, SEND_ACK_RD_WAIT) and 2 cycles during SCL low (transitions between states FIFO_POP_NEXT_DATA, SAVE_NEXT_FIFO_DATA, SEND_DATA).

Table 11: I2C Slave minimum frequency

	I2C SCL frequency		
	100kbit/s	**400kbit/s**	**1Mbit/s**
Min. SCL high	4000ns	600ns	260ns
Rounded (Min. SCL high / cycles needed)	1000ns	150ns	66ns
Minimum frequency for SCL high	1MHz	6.67MHz	15.15MHz
Min. SCL low	4700ns	1300ns	500ns
Rounded (Min. SCL low / cycles needed)	2350ns	650ns	250ns
Minimum frequency for SCL low	430kHz	1,54MHz	4MHz
Minimum I2C Slave frequency	**1MHz**	**6.67MHz**	**15.15MHz**

The minimum I2C Slave frequencies mentioned in Table 11 were used during the verification.

4.3.10.2 Minimum APB Slave frequency

There is no minimum APB Slave frequency, because the I2C Slave uses clock stretching. However the following relationship should be fulfilled: $f_{apb} <= f_{i2c}$.

In case that the APB interrupt is generated based on RX fifo full/not empty signals, it is recommended to keep the APB frequency at least equal or higher as SCL frequency ($f_{APB} \geq f_{SCL}$) to be able to be able to correctly generate signals for APB interrupt. On the other hand this recommendation is often fulfilled automatically since APB frequencies are usually higher than SCL frequencies. In case that the interrupt based

on RX fifo_not_empty / fifo_full signals is not necessary and APB interrupt for bits related with RX FIFO are masked (interrupt generated based on selected bit), this recommendation does not apply.

4.4 Analysis of clock gating use in the design

4.4.1 Clock gating types

In order to achieve results that would be comparable, I chose the following four kinds of clock gating use.

- CG_NONE - No clock gating used at all.
- CG_AUTO - Automatic clock gating used in DC Shell during Synthesis as described in chapter 3.4.1.
- CG_MAN - Manual clock gating – manually added clock gating cells that were marked as dont_touch cells.
- CG_MAN_AUTO - This variant is a combination of automatic and manual clock gating.

4.4.2 Clock-gating analysis in I2C block

This following analysis was used for manual inserting of clock gating cells.

4.4.2.1 I2C FSM

The FSM controls when clock gating is used to enable registers. In addition, clock gating was used also clock gating inside the FSM. An extra signal was added to determine if next state is different from the current state. If so, the clock for the register that stores the current state is enabled.

4.4.2.2 I2C Data Unit

For the analysis of where to use clock gating, we have to decide which registers have to be part of the always-on logic and which can be used for clock gating. In this design it is important to keep the registers on that are used for generating interrupt signals for APB and those registers that are used for controlling I2C communication such as for determination of start, stop condition and SCL edges. These registers are listed in Table 12.

Table 12: I2C Always-on registers

Register	**Reason**
ERR_REG	Error register (interrupt signal for APB)
INTR_BITS_CLR_REG1, INTR_BITS_CLR_REG2	Synchronization registers for clearing interrupt bits
RST_SYN_REG1, RST_SYN_REG2	Synchronization registers for reset
SCL_CURR, SCL_PREV	Generating SCL rising edge, SCL Falling edge, start condition, stop condition
SDA_CURR, SDA_PREV	Generating start condition, stop condition
START_BIT	Start bit (interrupt signal for APB)
STOP_BIT	Stop bit (interrupt signal for APB)

This leaves us with registers that will not need to be clocked in some cases. FSM, however, generates enable signals for these registers anyway, so these signals will be used for enabling the clock cell. Registers in this design where clock gating is useful, are those that are used only during communication and the register for saving I2C Slave address, since this is used only at the beginning of the communication for saving the address. Table 13 provides a list of registers where clock gating was used. It also shows bit width of these registers. It is recommended to have at least 3-4 bits for an enable signal, and all these registers satisfy this condition. Therefore clock gating was used on them. Reg0 always changes only 1bit during a write operation in this register, but it is a 8bit register, therefore it is convenient to use clock gating for this register as well. Register Wait_Counter is a 2bit register. Therefore, clock gating wasn't used on this register.

All these registers have one thing in common – their enable signals are mostly on low. Therefore it is convenient to use clock gating on them.

Table 13: I2C Registers that can be clock gated

Register	Bits	Reason	Write enabled when	CG used
Reg0	8	Used only during communication. Change only 8x per transfer	Data received from I2C Master	Yes
Reg1	8	Used only during communication. Change only 8x per transfer	Data written from TX fifo (for transfer to I2C Master)	Yes
Reg_Addr	8	Used for saving I2C Slave address, address saved at beginning of communication, stays without change during most of the time of use	I2C Slave address stored from TX fifo	Yes
Cycle_Counter	4	Used only during communication. Change only 8x per transfer	Counting bit indexes when receiving / sending data bits	Yes
WAIT_COUNTER	2	Used when I2C Slave address changed	Reseting I2C Slave + memories after I2C Slave address change	No

4.4.2.3 FIFOs

Both TX and RX fifos are IP that have inconsiderable consumption. It is therefore important to take this into account. Clock signals for Fifos don't only serve for data push/pop, but also for generating state signals (full, empty,…). This makes it more complicated. For this reason there was an extra signal called i2c_active added to the I2C FSM that expresses when a transaction is being done. When this signal is on high, the Fifos I2C clock is enabled.

4.4.3 Clock-gating analysis in APB block

4.4.3.1 APB FSM

Clock gating was also used for APB FSM. The way it was done is similar to the way clock gating was applied to I2C FSM, the description is in chapter 4.4.2.1.

4.4.3.2 APB Data Unit

Table 14 lists the always-on registers. These are registers that are used for interrupt signals. They always have to be on for proper generation of the interrupt signal towards APB Bridge and therefore for correct function.

Table 14: APB Always-on registers

Register	Reason
INTR_REG	Interrupt register
RESYNC_INTR_BITS	Resynchronization of interrupt bits from I2C Slave

The registers list where clock gating is used is in Table 20. There are two registers and both of them are 8-bit registers which is wide enough to use clock gating on them. One of them is the register for storing interrupt mask, this value doesn't usually change very often, and therefore it is convenient to use clock gating with this register. The other register is for registering data output and its value changes only during communication.

Table 15: APB Registers with applied clock gating

Register	Bits	Reason	Clock enabled when
INTR_MASK_REG	8	Interrupt mask, change only on request from APB Bridge	Request from APB Bridge to write new interrupt mask
PRDATA	8	Registered data output	New data on output for APB Bridge

4.4.3.3 Fifos

As already mention in chapter 4.4.2.3, it is important for the fifos to have the clock active longer than just for data transfers to generate signals. For this reason, the signal i2c_active was synchronized on the top level to the APB clock domain and was used along with pselx and pready signals to enable clock for the FIFOs. The resynchronization cell for signal i2c_active becomes a part of the always-on logic.

4.4.4 Clock-gating code example

The following code describes an example of using a clock gating cell. It shows that the use of clock gating on RTL level doesn't do any major changes; however, it enlarges the code.

The first part of the code describes the case in which clock gating is used. First, an extra wire is instantiated for the gated clock and follows the instantiation of the gating cell. This gating cell is marked as a "dont_touch" cell for synthesis, so that the DC Shell

doesn’t change this cell in any way. The register then follows the description with the use of gated clock and without an enable signal.

The part of the code that follows after the `else command is the usual RTL description of a register without use of clock gating.

```
//-------------------------------------------------
//INTERRUPT MASKING REGISTER

`ifdef CLOCK_GATING_ENABLED
    wire clk_gate_1;

    gating_cell i_clk_gate_1(
      .clk_i(clk_pclk_i),
      .clk_en_i(en_intr_mask_reg_i),
      .clk_gate_o(clk_gate_1)
    );

   always @ (posedge clk_gate_1, negedge presetn_i)
     begin: intr_mask_reg_p
       if(presetn_i == 1'b0)
         intr_mask_o <= 8'b11111111;
       else
         intr_mask_o <= pwdata_i;
     end

`else


   always @ (posedge clk_pclk_i, negedge presetn_i)
     begin: intr_mask_reg_p
       if(presetn_i == 1'b0)
         intr_mask_o <= 8'b11111111;
       else if (en_intr_mask_reg_i == 1'b1)
         intr_mask_o <= pwdata_i;
     end

 `endif
```

Figure 39: Clock gating code example

4.5 RTL

4.5.1 Coding

The device was coded according to the specification in Verilog 2001. It is a fully synchronous, fully synthesis-able design. The code itself can be found on the enclosed CD.

4.5.2 Resynchronization between the clock domains

4.5.2.1 Resynchronization of data

The data are sent through asynchronous FIFOs between the two clock domains. Therefore all the resynchronization is done in the fifos. Further description of these FIFOs is in chapter 4.5.6.

4.5.2.2 Resynchronization of signals

The signal resynchronization is done by resynchronization units consisting of two flip flops.

I2C Slave sets state signals for the APB Slave. These signals are synchronized in the APB domain by a multiple-bit resynchronization unit (the unit is called RESYNC_INTR_BITS).

The INTR_BIT_CLR signal that goes from APB to I2C domain is implemented to reset the registers (SELECTED_BIT_SET, START_BIT_SET, STOP_BIT_SET, ERR_SET) in the I2C domain. This signal is also resynchronized by two flip flops in the I2C domain to ensure the right function. The INTR_BIT_CLR signal is set active when interrupt register is read by APB master to reset registers in I2C domain that set interrupt signalizing values of I2C communication. The sequence of this steps is described in chapter 4.3.2.

4.5.3 Signals for DFT

Signals for DFT are not used in this design. This device is either considered as a hard-macro or as a soft-macro where DFT is implemented on the top-level of the chip.

4.5.4 I2C Slave Default address

I2C Slave can have set a default address. This is done by instantiating the module in the design by setting an instantiation parameter.

4.5.5 Changing APB addresses for operations

If the user wishes to change the addresses for any APB operation, you can do so in the dp_s_global_consts.v file by changing the values of the constants. The names of constants that need to be changed of each operation are in Table 16.

Table 16: Names of constants and their APB functions

Default APB address	APB Constant name	Function
000	PADDR_READ_DATA	Read data from FIFO_RX
001	PADDR_READ_INTR_REG	Read interrupt register
010	PADDR_ I2C_ADDR	Changes the I2C Slave address
011	PADDR_WRITE_DATA	Write data to FIFO_TX
100	PADDR_WRITE_INTR_MASK	Write interrupt mask

4.5.6 Fifos

In the beginning I was using FIFO models generated by Xilinx Coregen, I was developing in Xiling ISE so that I would be able to work from home.

After migrating the files with RTL to S3 Group environment, I had to use new fifos that were synthesizable. Both TX and RX Fifo were generated by the DesignWare Synopsys tool. There were some challenges and changes with using these fifos, because they have the first data on output right after writing it in the FIFO and not after a request. These fifos also have inverted reset signals and separate signals for full and empty signaling. Changes had to be done to fix these problems and differences before continuing to the next steps.

4.6 RTL code check (Hal)

RTL code check is done by Cadence Hal program. This program checks for different conditions and mistakes in the code starting from white spaces that might be

causing problems for other programs later during the design to unconnected wires or latches.

The design was run through this program and all the errors were corrected as well as most warnings.

Most of the errors were caused by white spaces and wrong coding (codes were imported from MS Windows environment to Linux environment). Whitespaces (tabs) had to be replaced by simple spaces.

Hal also reported errors in resynchronization. This was solved by adding a resynchronization cell for several parallel signals instead of several resynchronization flip-flops that were each 1 bit width in the APB Slave. The hardware specification wasn't changed, but the description in Verilog was corrected.

4.7 Verification

4.7.1 Introduction to verification

Based on the specification of the design, a list of steps that need to be verified (called verification items) were written in a list and based on this list a Verification plan (see Table 18) was written. The verification tests were written afterwards based on the Verification plan.

A third-party I2C Master bock that was downloaded from (Herveille, 2006) was used for the verification. In order to cover all the useful possibilities of the design behavior, the verification contains the following steps:

- Direction APB to I2C
  - I2C Slave address change through APB command
- Direction I2C to APB
  - Sending data from I2C Master to APB Bridge

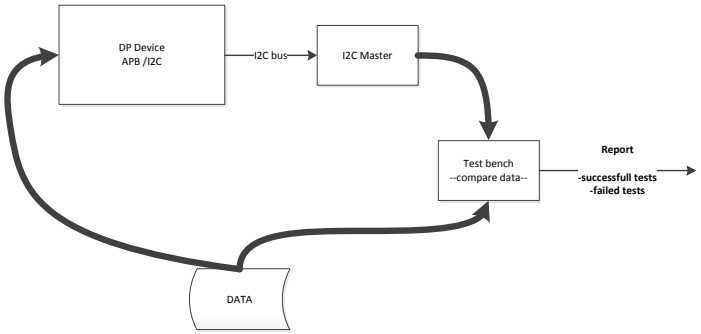


Figure 40: Testing sending data in the I2C to APB direction

- Direction I2C->APB ->I2C (typical communication scenario)
  - Sending a request from I2C Master, getting a response from APB Bridge to I2C Master

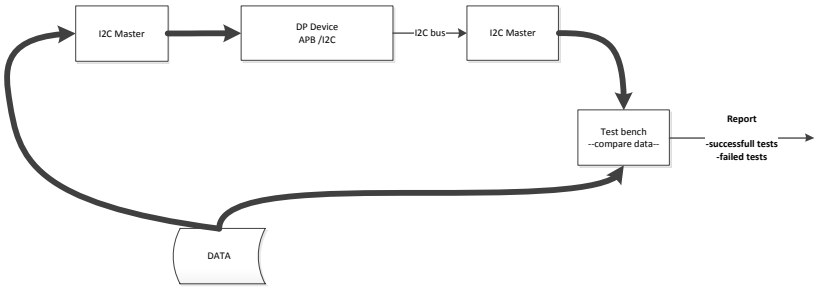


Figure 41: Typical communication test scenario – I2C->APB->I2C

- APB Interrupt
  - Fifo TX full
  - Fifo RX full
  - Fifo RX not empty
  - Unspecified error after START CONDITION (error caused by a start condition during data transfer)
  - Unspecified error after STOP CONDITION
  - Reading data error after START CONDITION
  - Reading data error after STOP CONDITION
  - Writing data error after START CONDITION
  - Writing data error after STOP CONDITION
  - Start bit
  - Stop bit
  - Selected bit
- Other
  - Verifying different I2C speeds - 10, 50, 100, 200, 400 kb/s and 1000kb/s

A script used to run all the tests at once. There are different tests and there were all run in different speeds – 10, 50, 100, 200, 400 kb/s and 1000kb/s. The speeds 100, 400 and 1000kb/s are given by the I2C standard, the other were used to verify compatibility with lower speeds.

4.7.2 Verification strategy

Assertions for I2C and APB protocols were not available during the design. A third party I2C Master was used to verify the correct communication of the I2C Slave. To model the APB Bridge, I wrote a model of this bridge for writing and reading data from the APB Slave. This decision was done based on the fact that APB is a quite easy protocol and in agreement with the submitter of this project.

All of the tests used for verification are self-checking, which means that after they run, a PASS/FAIL report is generated. They also generate logs during the simulation that include time of each log line, which help to determine and track the behavior of the device during the simulation. At the end of running the set of tests, a regression report is also generated that represents an overview of the tests passing/failing. Such a regression report can be found in Appendix B.

4.7.3 Frequencies used during verification

Frequencies for I2C Slave that were used are the minimum frequencies, which are mentioned in Table 11 and the reasons why these frequencies in chapter 4.3.10.1.

The frequency for APB Slave used during verification was set, so that $f_{apb}<f_{i2c}$ would be fulfilled. I chose a ratio $f_{apb} : f_{i2c}$ approximately 3.33:1. This means that for I2C speed 100kbit/s the frequency was 300 kHz (I2C Slave frequency 1MHz), 400kbit/s speed the frequency was 2MHz (I2C Slave frequency 6.67MHz) and for 1MBit/s I2C speed the frequency was 4.54MHz (I2C Slave frequency 15,15MHz).

Table 17: Frequencies used during verification

I2C Speed	**100kbit/s**	**400kbit/s**	**1Mbit/s**
I2C Slave frequency	1MHz	6.67MHz	15,15MHz
APB Slave frequency	300 kHz	2MHz	4.54MHz

The I2C frequencies were used the lowest possible to ensure that the device works with these frequencies. This was done, because for low power reasons, it is convenient to use the lowest frequencies possible.

4.7.4 Verification Plan

Note: In several places in the Verification plan, "send several data" is stated – these data were sent in a cycle which was controlled by a variable and usually 5 or 6B of data were transferred during these operations.

Table 18: Verification Plan

Abbreviation	Description	How to achieve
TC_TX000	Changing the I2C Slave address through a APB command (after default address)	Use default I2C address, generate reset (presetn). Then write to APB the command with the address PADDR=`PADDR_WRITE_I2C_ADDR and set a new I2C Slave address to PWDATA (different from default address), set PSELx=1 and in the next PCLK clock set PENABLE=1. Hold these values as long as PREADY=0. To verify that the device responds to this address, write data to RX fifo and read them through I2C Master.
TC_RX000	Writing data several bytes data through I2C Master to APB using burst mode at I2C. Using I2C default address	Reset the device (by PRESETn). Use default I2C address for I2C Slave. Send data from I2C Master to I2C Slave. Read the data through APB Master and compare the data. The data received by APB Master has to be the same as sent by I2C Master.
TC_RX001	Writing new I2C Slave address without using a default address first. Writing data several bytes data through I2C Master to APB using burst mode at I2C for varification that the I2C Slave actually communicated at the new address.	Comment the constant I2C_SLAVE_ADDRESS (in dp_s_global_consts.v file). Then reset the device (by PRESETn), after reset set PADDR=`PADDR_WRITE_I2C_ADDR and set a new I2C Slave address to PWDATA, set PSELx=1 and in the next PCLK clock set PENABLE=1. Hold these values as long as PREADY=0. Then Send data from I2C Master to I2C Slave
TC_RX002	Verifying APB device is returning Zeros for unspecified read operation	Generate reset (presetn), send a not-specified address as a read-request to APB device.
TC_RXTX000	Change of direction during I2C communication	Use default I2C address, generate reset (presetn). Write data to RX fifo, then I2C Master generates repeated start, changes the direction. After data is in RX fifo, read the data from RX fifo and write the same data to TX fifo. If data is not in TX fifo yet when required from I2C Master, the I2C Slave has to pull SCL to low. Then I2C Master reads data from TX fifo
TC_INTR000	Verifying APB Interrupt - fifo TX full	Use default I2C address, generate reset (presetn). Set the tx_fifo_full bit in the mask register to 1 and all other bits of the mask register to 0. Fill up the whole TX Fifo. Then set all bits of the mask register to zeros.
TC_INTR001	Verifying APB Interrupt - fifo RX full. Verifying NACK to I2C Master after sending more data to I2C Slave	Use default I2C address, generate reset (presetn). Set the rx_fifo_full bit in the mask register to 1 and all other bits of the mask register to 0. Fill up the whole RX fifo. Then try to write one more byte. Then set all bits of the mask register to zeros.

TC_INTR002	Verifying APB Interrupt - fifo RX not empty	Use default I2C address, generate reset (presetn), write interrupt mask with rx_not_empty bit on 1. Write data in RX memory through I2C. Then set all bits of the mask register to zeros.
TC_INTR003	Verifying APB Interrupt - unspecified error after START CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Send a START CONDITION to I2C Slave and in the middle of sending the address bits send a new START CONDITION to the I2C Slave. Then set all bits of the mask register to zeros.
TC_INTR004	Verifying APB Interrupt - reading data error after START CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Write data to TX FIFO. Start reading data from I2C Slave and then in the middle of the transfer start a new START CONDITION. Then set all bits of the mask register to zeros.
TC_INTR005	Verifying APB Interrupt - writing data error after START CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Start writing data to I2C Slave and then in the middle of the transfer start a new START CONDITION. Then set all bits of the mask register to zeros.
TC_INTR006	Verifying APB Interrupt - unspecified error after STOP CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Send a START CONDITION to I2C Slave and in the middle of sending the address bits send a new STOP CONDITION to the I2C Slave. Then set all bits of the mask register to zeros. Then set all bits of the mask register to zeros.
TC_INTR007	Verifying APB Interrupt - reading data error after STOP CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Write data to TX FIFO. Start reading data from I2C Slave and then in the middle of the transfer start a new STOP CONDITION. Then set all bits of the mask register to zeros. Then set all bits of the mask register to zeros.
TC_INTR008	Verifying APB Interrupt - writing data error after STOP CONDITION	Use default I2C address, generate reset (presetn). Write data to both RX and TX Fifo. Write interrupt mask with error bit on 1 and all other bits zeros. Start writing data to I2C Slave and then in the middle of the transfer start a new STOP CONDITION. Then set all bits of the mask register to zeros.
TC_INTR009	Verifying APB Interrupt - stop bit	Use default I2C address, generate reset (presetn), write interrupt mask with stop bit on 1 and all other bits zeros. Write data (1 byte) to I2C Slave through I2C Master. Then set all bits of the mask register to zeros.

TC_INTR010	Verifying APB Interrupt - start bit	Use default I2C address, generate reset (presetn), write interrupt mask with start bit on 1 and all other bits zeros. Write data (1 byte) to I2C Slave through I2C Master. Then set all bits of the mask register to zeros.
TC_INTR011	Verifying APB Interrupt - selected bit	Use default I2C address, generate reset (presetn), write interrupt mask with selected bit on 1 and all other bits zeros. Write data (1 byte) to I2C Slave through I2C Master. Then set all bits of the mask register to zeros.
TC_OTHR_000	Verify the reset values of all registers	Use default I2C address, generate reset (presetn), Verify the reset values of all registers

4.7.5 Code coverage

Code coverage describes how much the code is covered by the verification tests. Cadence NCSim simulator was used for running the tests. Another tool by Cadence ICCR is also able to view the code coverage and parts of the code that are not covered as well as visualize final state machines and show which states are covered. Fifos were excluded from the code coverage, because they were generated Synopsys Design Ware and are not a part of the master's project development.

The test tc_rx001 doesn't use the default I2C address and a whole new different run of make file had to be done for this test, which means that this test can't be merged with the other tests (not available by the development tools) in order to view the code coverage merged for all the tests together. Therefore there are two different sections, the section 4.7.5.1 contains the main tests and section 4.7.5.2 contains only the tc_rx001 test that verifies the case when default I2C address isn't used and so the only differences between using and not using the I2C default address will be mentioned there.

4.7.5.1 Verification tests using I2C Slave address

Figure 42 shows percentage coverage of the merged tests. The coverage isn't 100% which is given by two different facts. The first fact was described above (the use of default I2C Slave address). The other fact is that the ICCR tool expects to cover every "else" branch of any "if" command. The FSM was written by a "case" command where at the very beginning the current state is assigned as the next state and then possibly the next state is changed, but doesn't have to be changed. Therefore the "else branch" is written in the code, although the ICCR tool doesn't understand this.

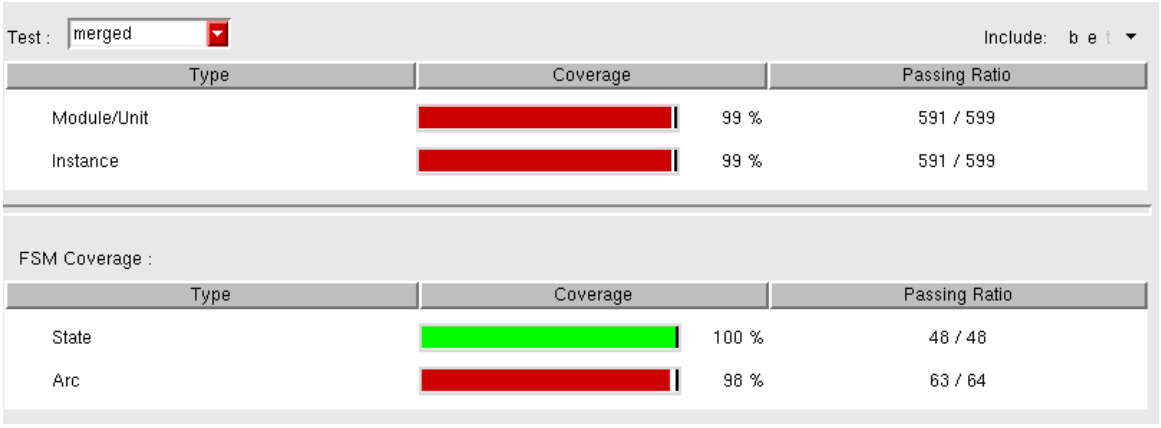


Figure 42: Code coverage summary

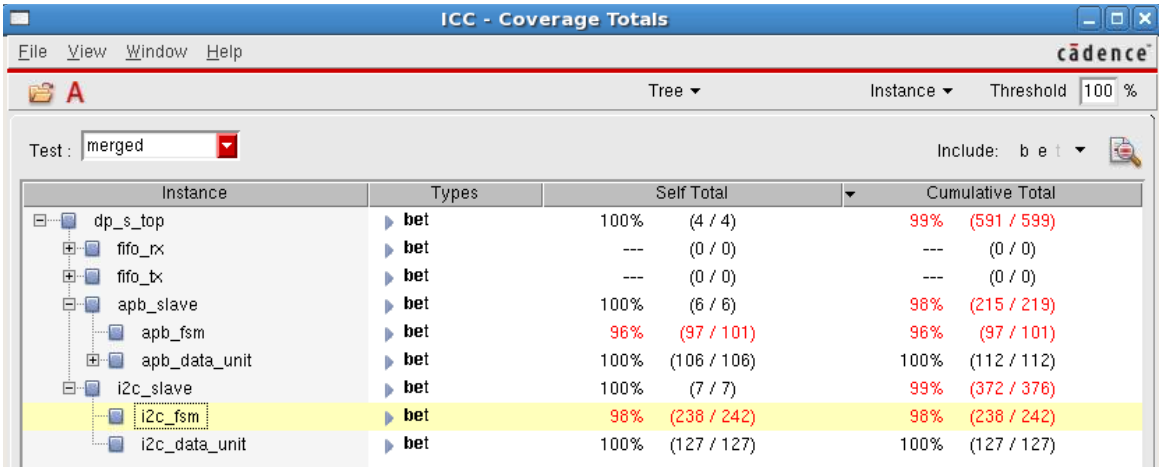


Figure 43: Code coverage code/data overview

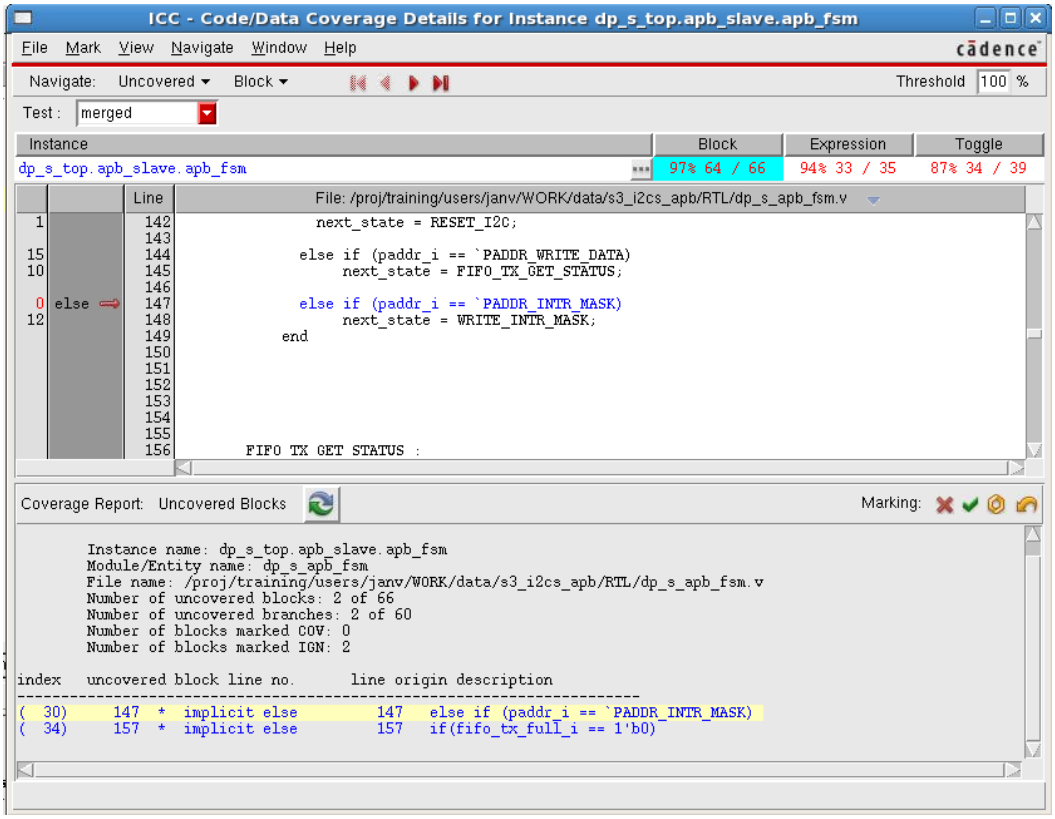


Figure 44: Implicit else example

Figure 45 shows the state and transition coverage for APB part of the device. The transition between INIT and IDLE state isn't covered, because that is the transition that is used in cases when default I2C Slave address isn't used. Therefore the INIT state is colored purple.

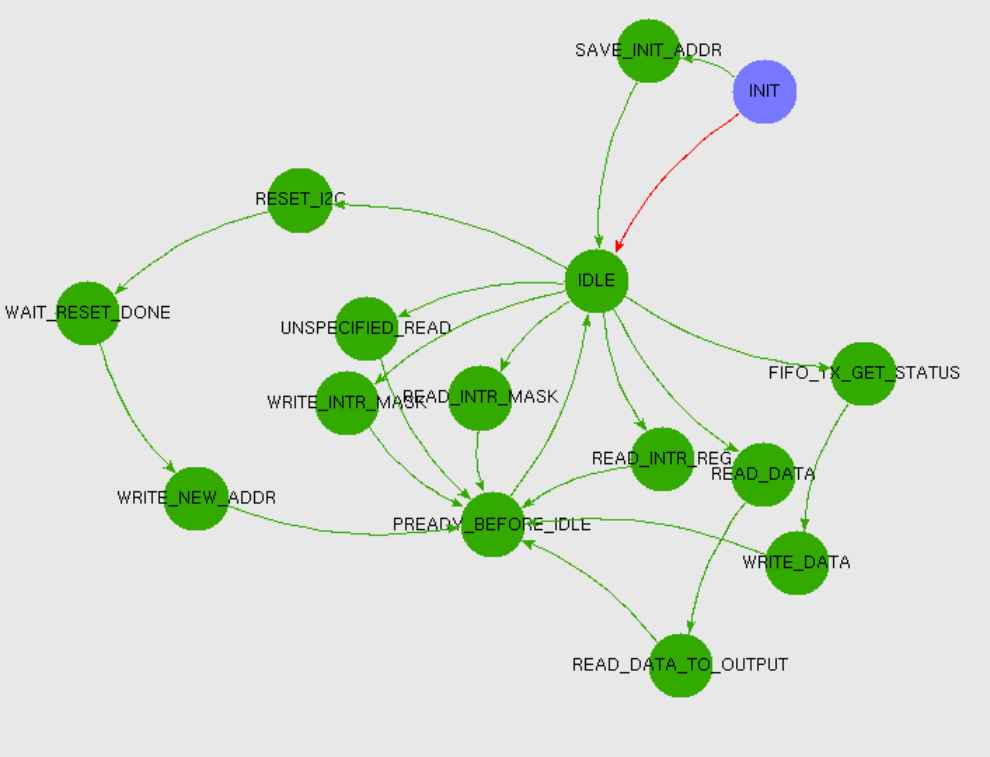


Figure 45: APB FSM state coverage (not using default I2C Slave address)

Figure 46 shows the state and transition coverage for the I2C FSM. The diagram shows that all states and transitions are covered.

State ERR_SIGNALING is assigned from all other states (except those states and conditions when it is not useful) whenever an error in the I2C communication occurs. Therefore this condition is coded as an "if" command after the "case" statements in the FSM process for selecting the next state. This is also the reason why this state is colored in a purple color.

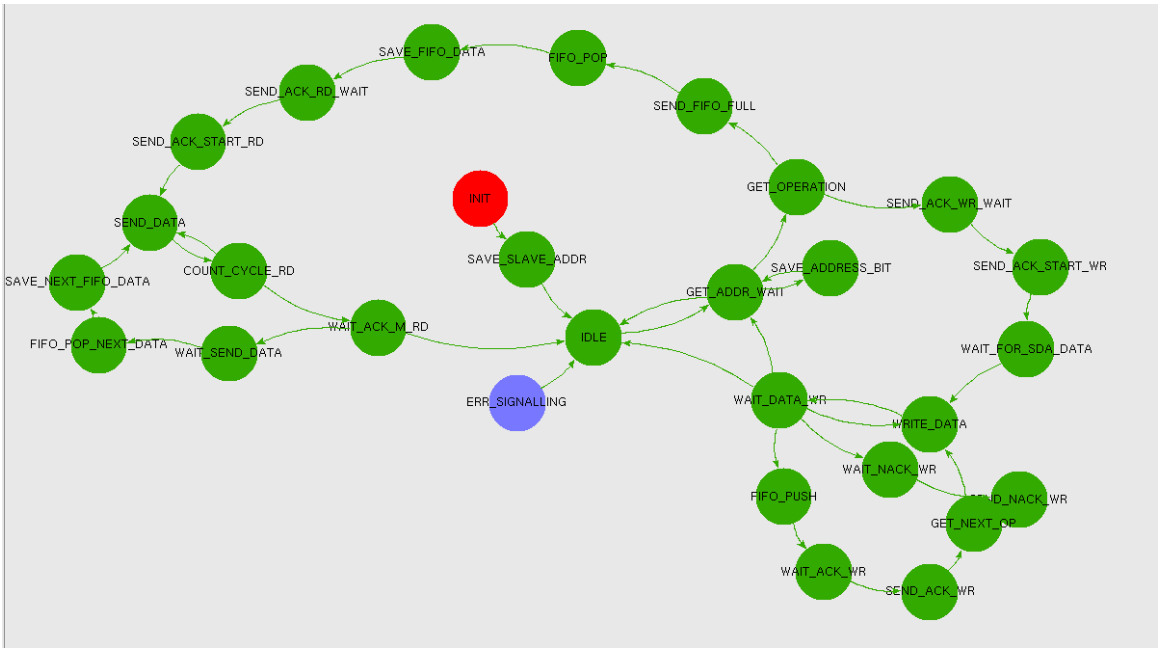


Figure 46: I2C FSM state coverage

4.7.5.2 Verification tests without using default I2C Slave address

Figure 47 displays state coverage of test tv_rx001. The only purpose of this diagram and these tests is to prove that the transition from state INIT to state IDLE which isn't covered in Figure 45 is also covered by the verification tests.

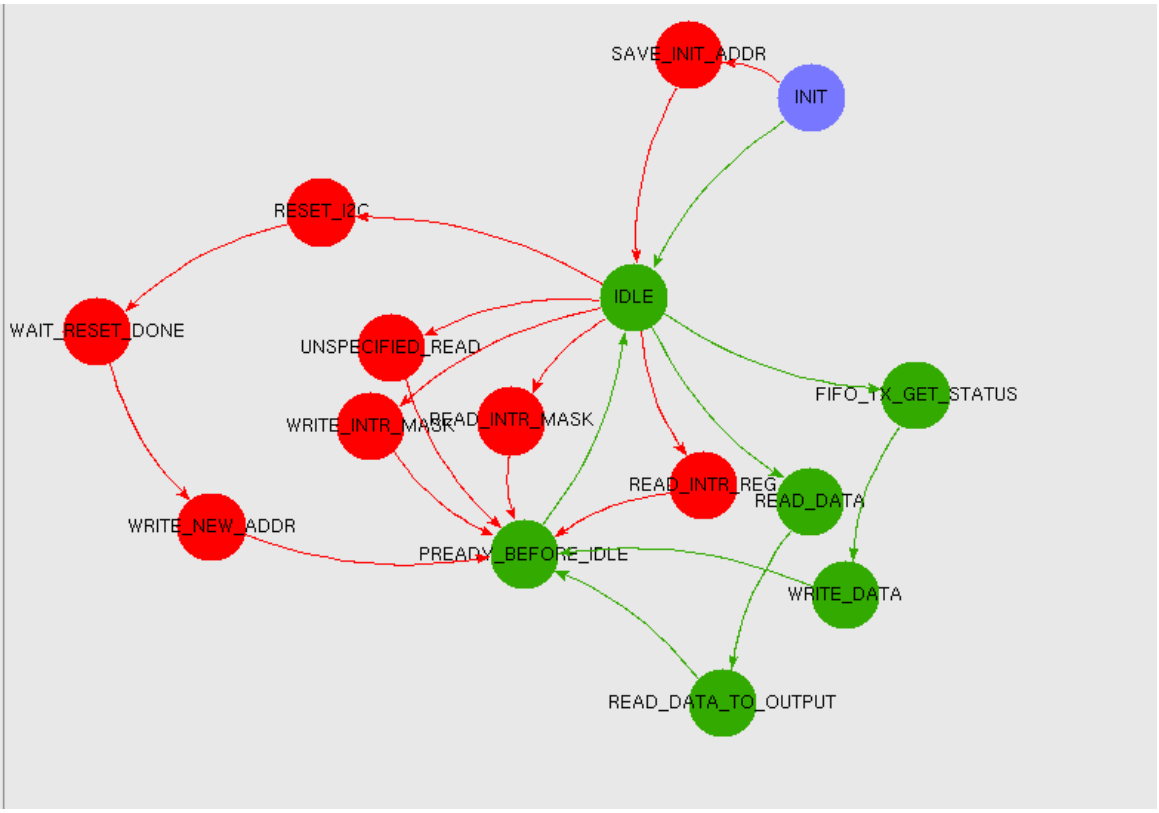


Figure 47: APB FSM state coverage (using default I2C Slave address)

Code coverage is useful to make sure all the important parts of code are covered. By being able to view the FSM, I found out some redundancies that I removed after

realizing them. I even found one state that was never reached and didn't even have any transition going out to another state.

I also found that some parts of the code were not covered although the tests were supposed to cover them. This signalized a mistake in the particular tests, which I corrected thanks to being able to know that the test is a wrong-pass.

4.8 Synthesis

4.8.1 What happens during synthesis

Synthesis is a step where RTL code (written in Verilog in this case) is translated into standard logical cells connected by nets – so called netlist.. The input for synthesis is the RTL code and Library files. The library files were used for the technology TSMC 65nm (tcbn65lp – low power).

Synthesis also generates warning (or error) reports concerning the design. This can be e.g. warnings about latches in the design, nets without a type, driver, fanout etc. etc.

DC Shell also generates consumption estimation during synthesis, which is further described in chapter 4.8.2 and 5.1.

Synthesis was run 4 times in this design according to the kind of clock gating that was used in the design. This is a nonstandard solution and was done in order to be able to compare different consumption results by the end of the project. Automatic clock gating described in chapter 3.4.1 can be added during synthesis just by changing one command in the synthesis command script.

4.8.2 Synthesis power consumption

A power consumption estimation report is generated by the Synopsys DC Shell tool during synthesis. This report is based on an approximate expected signal and clock activity. The consumptions are in stated mW.

4.8.2.1 Synthesis power consumption – without Clock gating

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
dp_s_top	9.99e-04	3.67e-02	1.26e+03	**3.90e-02**	100.0
apb_slave (dp_s_apb_slave)	1.87e-04	2.01e-03	152.869	2.35e-03	6.0
apb_data_unit (dp_s_apb_data_unit)	2.38e-05	1.26e-03	84.136	1.37e-03	3.5
resync_intr_bits (dp_s_resync)	3.46e-07	3.49e-04	14.218	3.64e-04	0.9
apb_fsm (dp_s_apb_fsm_10)	6.48e-05	7.08e-04	63.168	8.36e-04	2.1
i2c_slave (dp_s_i2c_slave)	1.80e-04	7.65e-03	213.076	8.04e-03	20.6
i2c_fsm (dp_s_i2c_fsm)	1.06e-04	3.35e-03	128.057	3.58e-03	9.2
i2c_data_unit (dp_s_i2c_data_unit)	7.34e-05	4.30e-03	83.935	4.46e-03	11.4
fifo_tx (dp_s_top_dp_s_fifo_1)	3.04e-04	8.39e-03	442.418	9.14e-03	23.5
fifo_rx (dp_s_top_dp_s_fifo_0)	2.73e-04	1.86e-02	451.683	1.94e-02	49.7

4.8.2.2 Synthesis power consumption – with automatic Clock gating

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
dp_s_top	1.12e-03	1.73e-02	1.23e+03	**1.97e-02**	100.0
apb_slave (dp_s_apb_slave)	2.34e-04	1.59e-03	153.907	1.98e-03	10.1
apb_data_unit (dp_s_apb_data_unit)	2.33e-05	8.06e-04	81.414	9.11e-04	4.6
resync_intr_bits (dp_s_resync)	3.46e-07	3.49e-04	14.218	3.64e-04	1.9
apb_fsm (dp_s_apb_fsm_10)	8.41e-05	7.37e-04	66.329	8.87e-04	4.5
i2c_slave (dp_s_i2c_slave)	2.59e-04	5.37e-03	213.367	5.84e-03	29.7
i2c_fsm (dp_s_i2c_fsm)	1.81e-04	3.53e-03	123.225	3.83e-03	19.5
i2c_data_unit (dp_s_i2c_data_unit)	7.87e-05	1.84e-03	89.058	2.01e-03	10.2
fifo_tx (dp_s_top_dp_s_fifo_1)	3.00e-04	4.97e-03	439.221	5.71e-03	29.1
fifo_rx (dp_s_top_dp_s_fifo_0)	2.71e-04	5.36e-03	426.900	6.06e-03	30.8

4.8.2.3 Synthesis power consumption – with manual Clock gating

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
dp_s_top	3.33e-03	3.05e-02	1.32e+03	**3.51e-02**	100.0
apb_slave (dp_s_apb_slave)	2.17e-04	1.67e-03	169.495	2.05e-03	5.8
apb_data_unit (dp_s_apb_data_unit)	2.29e-05	8.10e-04	91.296	9.24e-04	2.6
i_clk_gate_1 (dp_s_top_gating_cell_1)	0.000	2.77e-05	4.880	3.25e-05	0.1
i_clk_gate_2 (dp_s_top_gating_cell_2)	7.52e-07	3.30e-05	4.873	3.86e-05	0.1
resync_intr_bits (dp_s_resync)	3.46e-07	3.49e-04	14.219	3.64e-04	1.0
apb_fsm (dp_s_apb_fsm_10)	9.59e-05	8.13e-04	72.631	9.81e-04	2.8
i_clk_gate_11 (dp_s_top_gating_cell_3)	1.51e-05	1.23e-04	4.547	1.43e-04	0.4
i2c_slave (dp_s_i2c_slave)	1.44e-04	4.78e-03	239.313	5.17e-03	14.7
i2c_fsm (dp_s_i2c_fsm)	6.16e-05	2.87e-03	132.755	3.07e-03	8.7
i_clk_gate_10 (dp_s_top_gating_cell_4)	2.81e-06	1.20e-04	4.867	1.28e-04	0.4
i2c_data_unit (dp_s_i2c_data_unit)	8.25e-05	1.91e-03	105.473	2.10e-03	6.0
i_clk_gate_6 (dp_s_top_gating_cell_5)	4.34e-08	9.25e-05	4.880	9.74e-05	0.3
i_clk_gate_5 (dp_s_top_gating_cell_6)	0.000	9.22e-05	4.880	9.71e-05	0.3
i_clk_gate_3 (dp_s_top_gating_cell_7)	6.51e-06	1.67e-04	4.841	1.78e-04	0.5
i_clk_gate_4 (dp_s_top_gating_cell_8)	9.31e-07	9.98e-05	4.877	1.06e-04	0.3
fifo_tx (dp_s_top_dp_s_fifo_1)	3.02e-04	7.53e-03	444.298	8.27e-03	23.5
fifo_rx (dp_s_top_dp_s_fifo_0)	2.72e-04	1.59e-02	454.980	1.66e-02	47.2
resync_active (dp_s_resync_BIT_WIDTH1)	1.17e-06	8.42e-05	3.221	8.86e-05	0.3
i_clk_gate_9 (dp_s_top_gating_cell_9)	5.77e-04	1.36e-04	4.485	7.17e-04	2.0
i_clk_gate_8 (dp_s_top_gating_cell_0)	1.76e-03	4.39e-04	4.533	2.21e-03	6.3

4.8.2.4 Synthesis power consumption – with manual + automatic Clock gating

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
dp_s_top	2.12e-03	1.62e-02	1.29e+03	**1.96e-02**	100.0
apb_slave (dp_s_apb_slave)	2.49e-04	1.71e-03	166.879	2.12e-03	10.8
apb_data_unit (dp_s_apb_data_unit)	2.33e-05	8.15e-04	88.852	9.27e-04	4.7
i_clk_gate_1 (dp_s_top_gating_cell_1)	1.84e-07	2.88e-05	4.878	3.38e-05	0.2
i_clk_gate_2 (dp_s_top_gating_cell_2)	2.21e-07	3.37e-05	4.872	3.88e-05	0.2
resync_intr_bits (dp_s_resync)	3.46e-07	3.49e-04	14.219	3.64e-04	1.9
apb_fsm (dp_s_apb_fsm_10)	9.90e-05	8.49e-04	71.861	1.02e-03	5.2
i_clk_gate_11 (dp_s_top_gating_cell_3)	5.62e-06	1.23e-04	4.553	1.33e-04	0.7
i2c_slave (dp_s_i2c_slave)	1.46e-04	4.83e-03	239.315	5.21e-03	26.6
i2c_fsm (dp_s_i2c_fsm)	6.20e-05	2.87e-03	132.730	3.07e-03	15.6
i_clk_gate_10 (dp_s_top_gating_cell_4)	2.85e-06	1.20e-04	4.867	1.28e-04	0.7
i2c_data_unit (dp_s_i2c_data_unit)	8.35e-05	1.95e-03	105.500	2.14e-03	10.9
i_clk_gate_6 (dp_s_top_gating_cell_5)	9.95e-09	9.25e-05	4.880	9.74e-05	0.5
i_clk_gate_5 (dp_s_top_gating_cell_6)	0.000	9.22e-05	4.880	9.71e-05	0.5
i_clk_gate_3 (dp_s_top_gating_cell_7)	3.08e-06	1.67e-04	4.841	1.75e-04	0.9
i_clk_gate_4 (dp_s_top_gating_cell_8)	9.46e-07	9.99e-05	4.877	1.06e-04	0.5
fifo_tx (dp_s_top_dp_s_fifo_1)	3.01e-04	4.41e-03	439.986	5.15e-03	26.3
fifo_rx (dp_s_top_dp_s_fifo_0)	2.71e-04	4.60e-03	427.782	5.30e-03	27.0
resync_active (dp_s_resync_BIT_WIDTH1)	1.17e-06	8.42e-05	3.220	8.86e-05	0.5
i_clk_gate_9 (dp_s_top_gating_cell_9)	2.71e-04	1.34e-04	4.487	4.09e-04	2.1
i_clk_gate_8 (dp_s_top_gating_cell_0)	8.25e-04	4.32e-04	4.534	1.26e-03	6.4

4.8.3 Synthesis power consumption summary

Chyba! Chybný odkaz na záložku. shows the consumption estimations after synthesis. Automatic clock gating has quite a big effect here, it saves approximately 50%. Manual clock gating has obviously less impact with the signal and clock activity the synthesis tool uses. This is caused because the consumption modes are basically not used.

Table 19: Power consumption results – after synthesis

Netlist type	Clock gating type				Units
	NONE	AUTO	MAN	MAN_AUTO	
After synthesis, no timing, estimated switching activities	39.00	19.70	35.10	19.60	uW/1s

4.9 Formal verification RTL to Gate

Formal verification that compares the equivalence of the RTL and Gate level netlist was also run in the Synopsys Formality tool. This tool compares there two netlists and as a result gives a report whether the two are equivalent or not. This has been used to make sure that the synthesis was run successfully without any changes in the design in any of the synthesis steps.

4.10 Verification – Gate level simulation without timing

After having the netlist generated through synthesis, I also did a gate level simulation by running the verification test on the netlist. This resulted in some failed tests which I had to fix. Minor changes had to be done in the RTL code and also some data was one clock cycle late on the output. I fixed these problems and continued towards the physical design.

4.11 Physical design

4.11.1 Introduction

For the Physical design of the device, the following steps were used, which will be further described:

- Floorplan
- PlaceCells
- CTS (Clock Tree Synthesis)
- Route
- Export
- Extract

In addition to these basic steps, several optimization scripts were also run that are usually connected with one of the steps.

Four different rundirs had to be made for physical design and the physical design was run under them to be able to make four different designs to be able to measure four different consumptions. This is a step that's very unusual for development and had to be done for the purpose of being able to get several different consumption estimation values.

4.11.2 Floorplan

Area allocation is done during the Floorplan step. This means that measures of the chip are defined. Power supply and ground is defined by placing a ring around the chip. Port placement is also set. Macro cells are also placed in this step, but they were not used in this design. All these steps are defined by the designer.

Four metallization layers were used for the design. Density of cells is 70%. These numbers were recommended by the S3 Group designers.

The proportions of the measurements of the chip were chosen in approximate ration 1:2. The sizes are 157um and 82um, which gives 12874 um^2 of area.

4.11.3 Place cells

Standard logical cells are placed in the area and time optimization is done.

4.11.4 Clock tree synthesis

Clock tree synthesis serves for defining the clock tree in the chip. This is one of the most important steps. It is an interesting point of how different the clock trees are in the different uses of clock gating, which will be described in the next following chapters.

4.11.4.1 Logic clock tree

Figure 48 and Figure 49 show the logic clock tree for I2C / APB of DP device. There is no clock gating used, therefore the clock signal leads to all registers.

Figure 48: I2C clock tree – no clock gating

Figure 49: APB Clock tree – no clock gating

Figure 50 shows the clock tree of automatic clock gating. It is very obvious and visible how DC Shell implements clock gating by using functional clock gating. Since most registers in the design are 8bit, so are usually 8 registers connected to each gating cell.

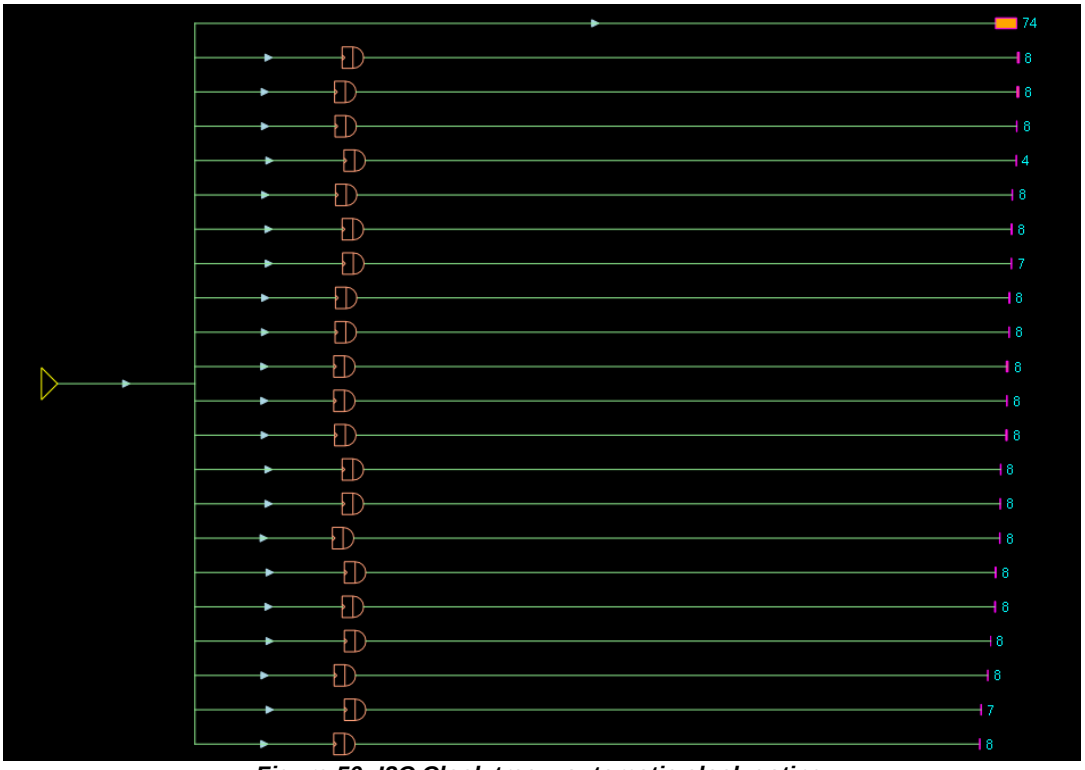


Figure 50: I2C Clock tree – automatic clock gating

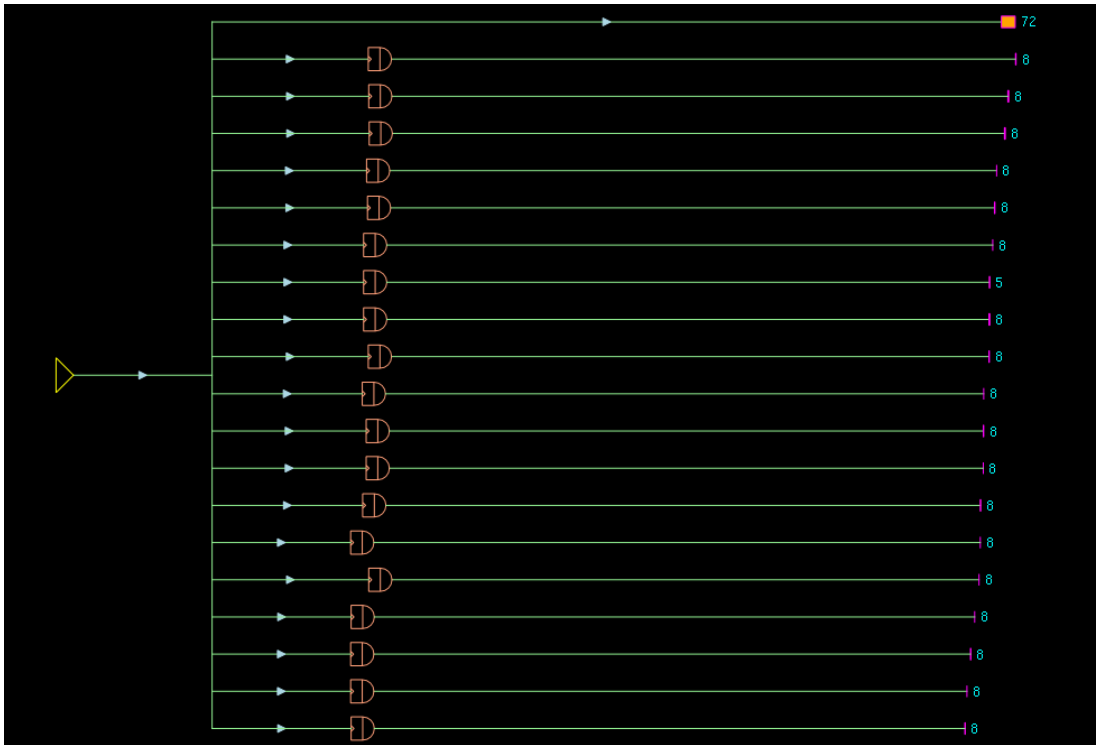


Figure 51: APB Clock tree – automatic clock gating

Clock tree with manual clock gating is shown on Figure 52 and Figure 53. It is very obvious that there are only those gating cells that were placed manually since there are only a few.

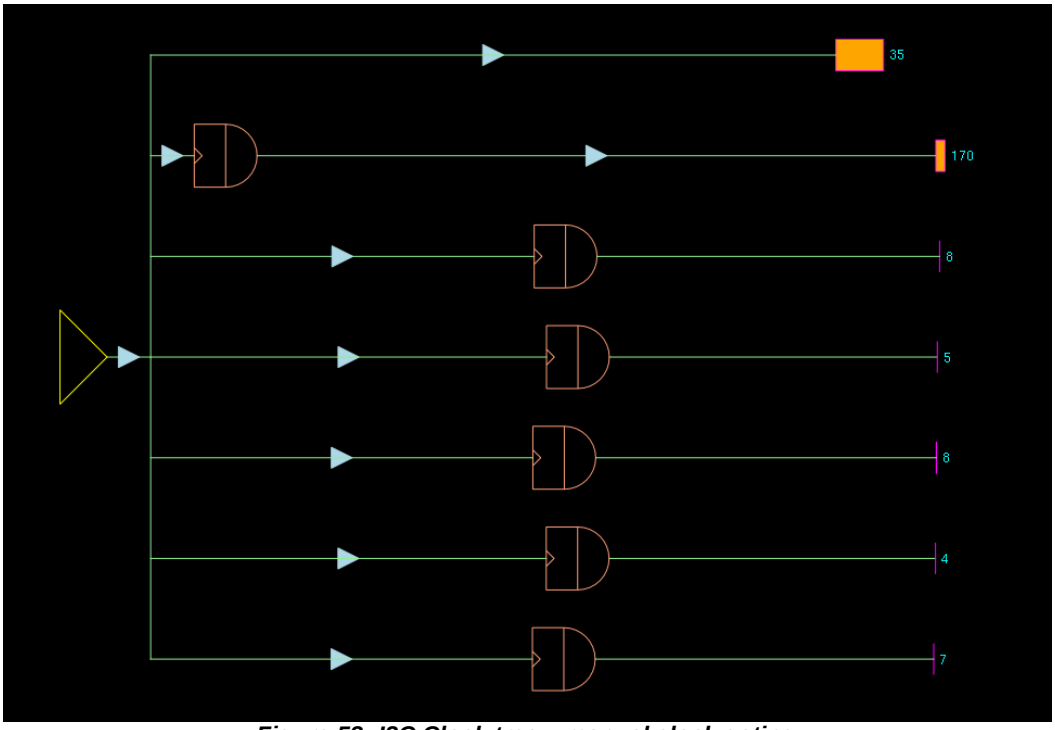


Figure 52: I2C Clock tree – manual clock gating

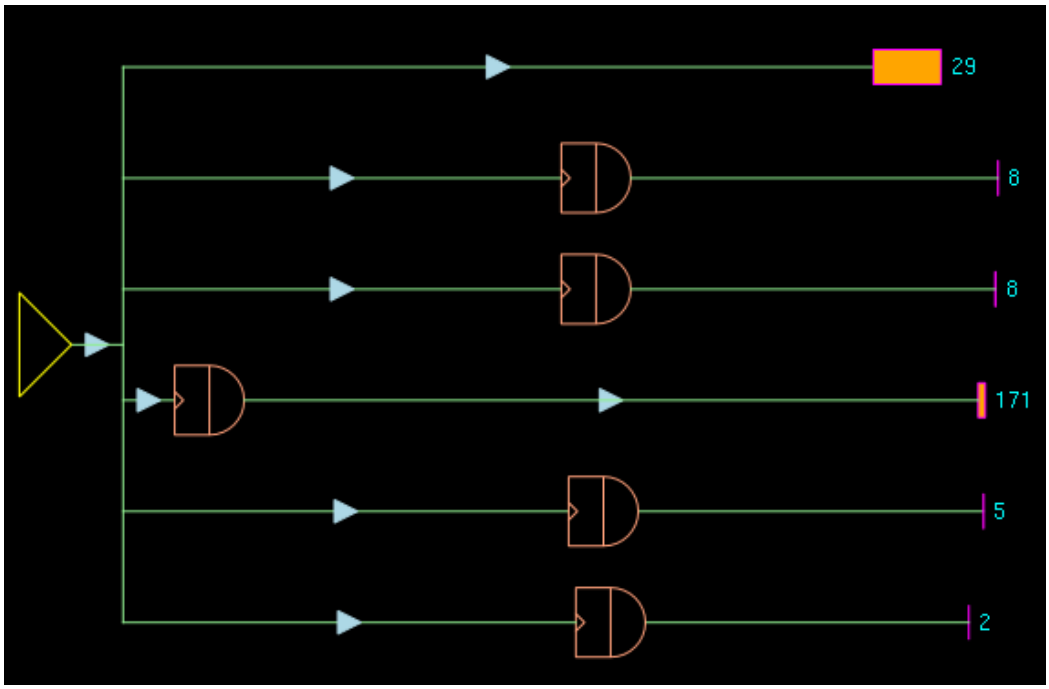


Figure 53 APB Clock tree – manual clock gating

Figure 54 and Figure 55 show I2C clock tree for combined clock gating. Here we can see how first the manual clock gating divides the tree in several branches and then in these branches automatic clock gating was used.

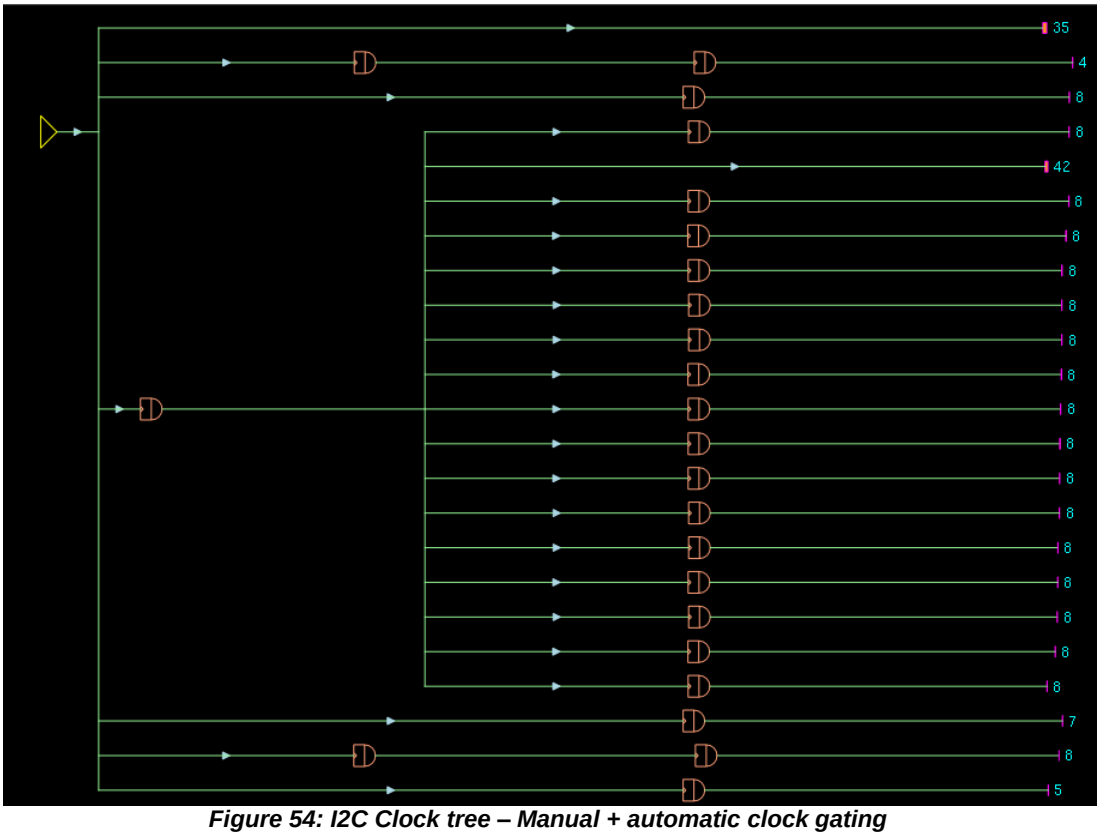


Figure 54: I2C Clock tree – Manual + automatic clock gating

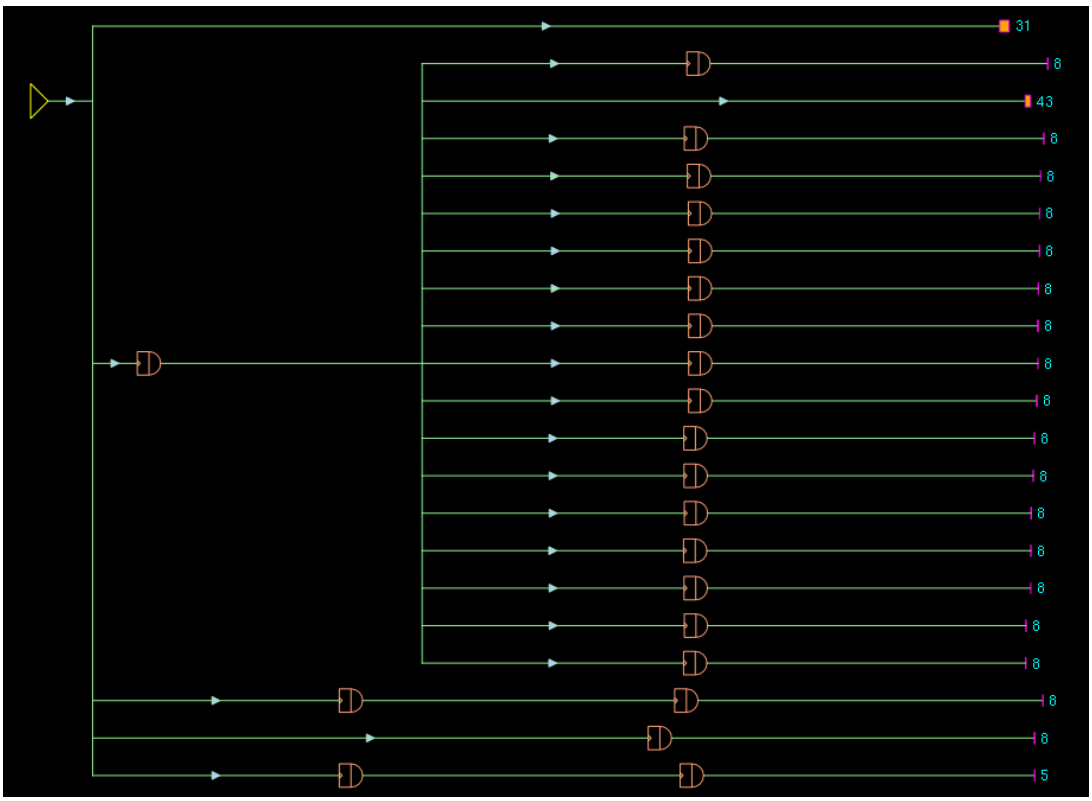


Figure 55: APB Clock tree – Manual + automatic clock gating

4.11.4.2 Physical clock tree

The following pictures show the physical clock tree of the chips. The clock pins are purposefully placed close to the middle of the sides of the chip, because the Cadence tool does the routing of the clock tree from the center of the chip to make possibly short ways to all registers.

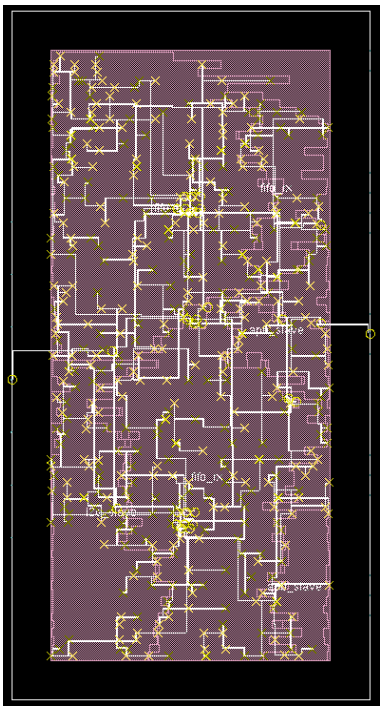


Figure 56: Clock tree – no clock gating

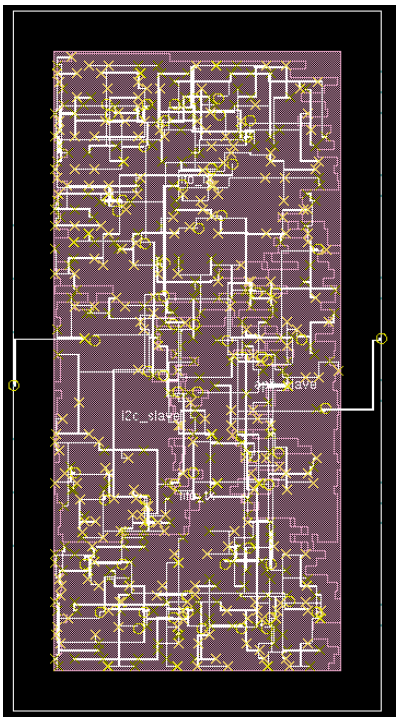


Figure 57: Clock tree – automatic clock gating

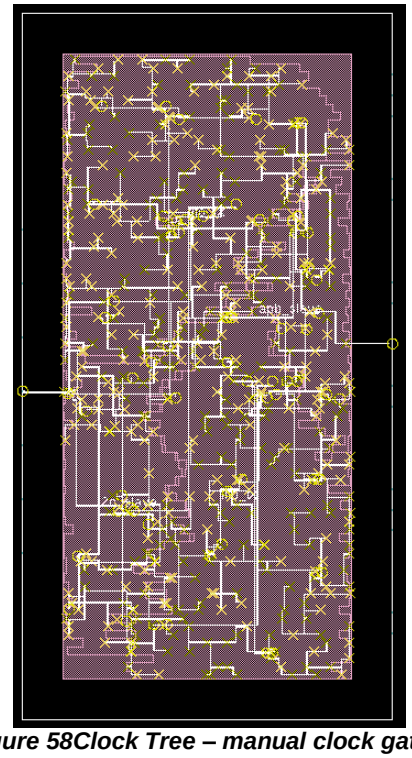


Figure 58Clock Tree – manual clock gating

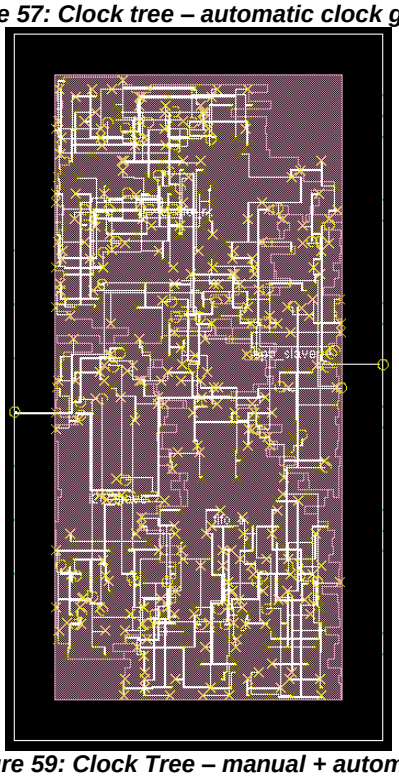


Figure 59: Clock Tree – manual + automatic clock gating

4.11.5 Root

In this step all cells and gates are connected.

4.11.6 Export

The netlist of the layout is exported after the physical design steps.

4.11.7 Extract

Extract serves for extracting a .spef (Standad Parasitic Extraction File) file with parasitics (resistances and capacitances) of the design under the best and worst conditions. This file will serve for generating a SDF (Standard Delay File).

4.11.8 Final Floorplan

The following pictures show the final floorplan after all the steps of the physical design of the chip (according to using the clock gating). As the pictures show, Cadence tool always used a different placement for different parts of the design. We can see that it always placed the I2C Slave close the left side, because the I2C pins are places on the left and APB Slave is placed towards the right side since the APB pins are on the right side.

4.11.8.1 Floorplan – no clock gating

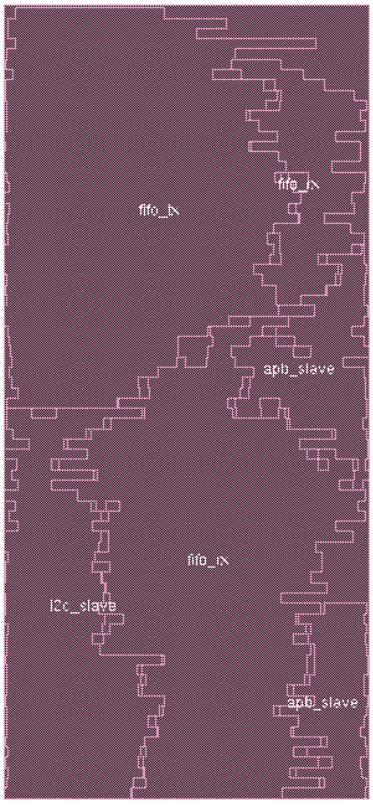


Figure 60: Floorplan – no clock gating

Figure 61: Floorplan no clock gating with nets

4.11.8.2 Floorplan – automatic clock gating

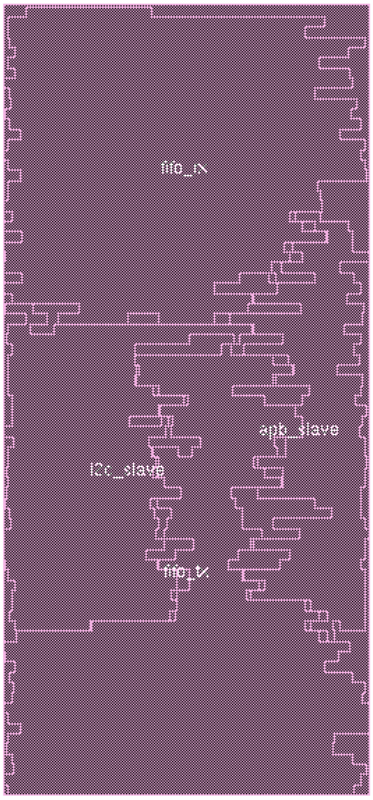


Figure 62: Floorplan – automatic clock gating

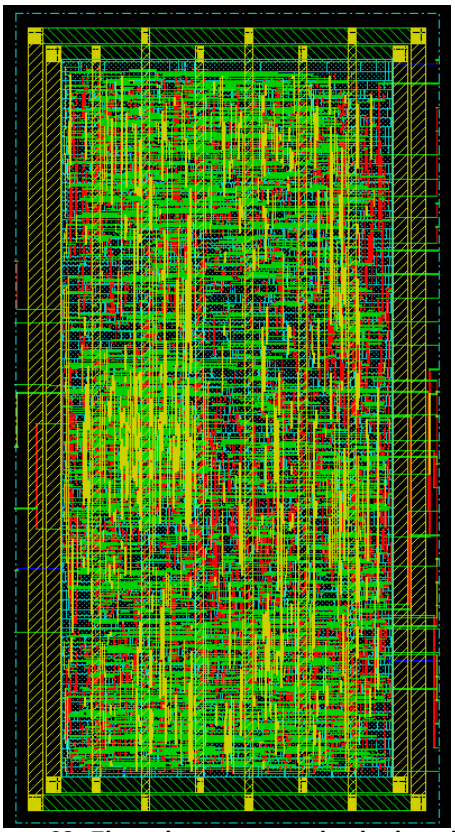


Figure 63: Floorplan – automatic clock gating with nets

4.11.8.3 Floorplan – manual clock gating

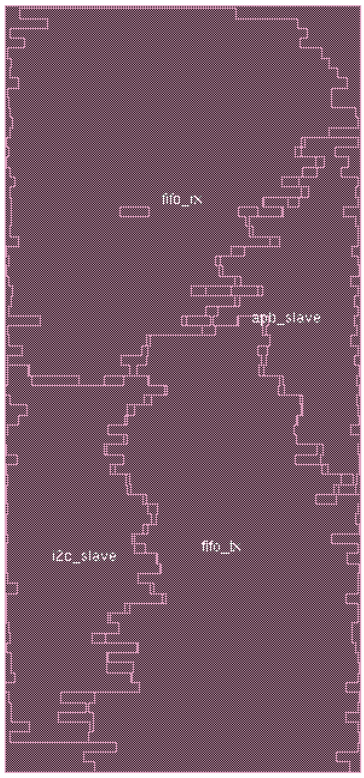


Figure 64: Floorplan – manual clock gating

Figure 65: Floorplan – manual clock gating with nets

4.11.8.4 Floorplan – manual + automatic clock gating

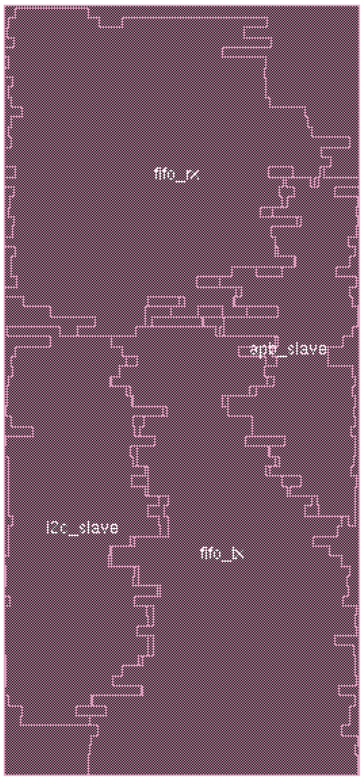


Figure 66: Floorplan – manual + automatic clock gating

Figure 67: Floorplan – manual + automatic clock gating with nets

4.12 Layout Verification with timing

4.12.1 Description

The layout verification serves as the final verification in this design and it serves especially for measuring the power consumption. Therefore, there was only one verification test used and this was the tc_rxtx000, which is the standard behavior test.

The inputs of this verification are a wave dump file (VCD file) and standard delay file (SDF). All of these are for four different variants according to the kind of clock gating that was used (CG_NONE, CG_AUTO, CG_MAN, CG_MAN_AUTO). VCD files are generated for IDLE mode and COMMUNICATION mode. SDF files are also generated for best and worst cases, which mean there are 8 VCD files and 8 SDF files.

The output of Layout verification is a PASS/FAIL report (specifying if the test passed or failed) and a Power Report. Timing reports for worst case of timing are in chapter 4.12.2. The following numbers and results in this document are only for timing worst case, because worst case is obviously more important to pass than best case.

The power estimation results were measured for 1Mbit/s speed transfers. The lowest possible frequency (15.15MHz) was used for the I2C Slave as the goal was to reach lowest power consumption possible and frequency influences dynamic power consumption. The reasons for using the frequency of 15.15MHz are mentioned in chapter 4.3.10.1.

4.12.2 Layout Verification Power reports for timing worst case

The following values are mentioned in mW.

4.12.2.1 Layout Verification Power report – no clock gating, Idle mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.03493	3.572e-06	0.0007587	0.0357	72.91
Macro	0	0	0	0	0
IO	0	0	4.57e-10	4.57e-10	9.335e-07
Combinational	7.971e-09	0	0.0005272	0.0005272	1.077
Clock (Combinational)	0.002272	0.01043	3.913e-05	0.01274	26.01
Total	0.03721	0.01043	0.001325	0.04896	100

4.12.2.2 Layout Verification Power report – no clock gating, Communication mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.03315	0.0001125	0.0007459	0.03401	71.7
Macro	0	0	0	0	0
IO	0	0	4.57e-10	4.57e-10	9.634e-07
Combinational	0.0001592	0.0002734	0.0004715	0.000904	1.906
Clock (Combinational)	0.002236	0.01025	3.891e-05	0.01252	26.4
Total	0.03555	0.01064	0.001256	0.04744	100

4.12.2.3 Layout Verification Power report – automatic clock gating, Idle mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.01501	0.0003347	0.0009836	0.01633	54.82
Macro	0	0	0	0	0
IO	0	0	7.901e-08	7.901e-08	0.0002652
Combinational	1.523e-06	2.374e-06	0.0003836	0.0003875	1.301
Clock (Combinational)	0.004542	0.008446	8.705e-05	0.01307	43.88
Total	0.01956	0.008783	0.001454	0.02979	100

4.12.2.4 Layout Verification Power report – automatic clock gating, Communication mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.01346	0.0003558	0.0009657	0.01478	52.04
Macro	0	0	0	0	0
IO	0	0	7.901e-08	7.901e-08	0.0002782
Combinational	0.0001444	0.0002665	0.0003324	0.0007434	2.618
Clock (Combinational)	0.004475	0.008316	8.68e-05	0.01288	45.35
Total	0.01808	0.008939	0.001385	0.0284	100

4.12.2.5 Layout Verification Power report – manual clock gating, Idle mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.006285	0.0001088	0.0008658	0.007259	40.95
Macro	0	0	0	0	0
IO	0	0	2.01e-08	2.01e-08	0.0001134
Combinational	4.57e-08	5.802e-08	0.0005284	0.0005285	2.981
Clock (Combinational)	0.004933	0.004831	0.0001765	0.00994	56.07
Total	0.01122	0.004939	0.001571	0.01773	100

4.12.2.6 Layout Verification Power report – manual clock gating, Communication mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.02736	0.0004649	0.0008172	0.02865	56.9
Macro	0	0	0	0	0
IO	0	0	2.01e-08	2.01e-08	3.992e-05
Combinational	0.0001974	0.000337	0.0004422	0.0009766	1.94
Clock (Combinational)	0.008208	0.01239	0.0001197	0.02072	41.16
Total	0.03577	0.01319	0.001379	0.05034	100

4.12.2.7 Layout Verification Power report – manual + automatic clock gating, Idle mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.006596	0.0001538	0.001017	0.007767	45.09
Macro	0	0	0	0	0
IO	0	0	9.079e-08	9.079e-08	0.0005271
Combinational	4.57e-08	6.305e-08	0.0003927	0.0003928	2.28
Clock (Combinational)	0.004382	0.004526	0.0001561	0.009064	52.63
Total	0.01098	0.00468	0.001566	0.01722	100

4.12.2.8 Layout Verification Power report – manual + automatic clock gating, Communication mode

Group	Internal Power	Switching Power	Leakage Power	Total Power	Percentage (%)
Sequential	0.01252	0.0004771	0.0009923	0.01399	47.2
Macro	0	0	0	0	0
IO	0	0	9.079e-08	9.079e-08	0.0003064
Combinational	0.0001918	0.0003663	0.0003115	0.0008696	2.935
Clock (Combinational)	0.006229	0.008437	0.0001089	0.01477	49.86
Total	0.01894	0.00928	0.001413	0.02963	100

5 Power consumption results

5.1 Power consumption results

There are two main consumption modes for this device – the Idle mode and Communication mode. Both of these modes were measured since the device stays in Idle mode part of time of its use and the consumption is lower during this period. A transfer of 6bytes both ways (I2C-> APB, APB -> I2C) was run during the communication mode to avoid inaccuracies which might be caused by not transferring enough data.

Note: Transferring 6bytes using the typical communication test took 155us.

The results of this consumption estimation are in Table 20. This table also shows the consumption estimation generated during synthesis bases on an expected clock activity by the synthesis tool. This information is only approximate, but can be quite useful, because it is available right after synthesis before any steps of physical design. Compared with the Communication mode, this value is between 60-80% of the consumption in Communication mode. Because synthesis estimations are not as accurate as estimations after physical design, the result evaluations in chapter 5.2 is written for estimations run after the physical design.

Table 20: Power consumption results

Netlist type	Consumption mode	Clock gating type				Units
		NONE	AUTO	MAN	MAN_AUTO	
After synthesis, no timing, estimated switching activities	-	39.00	19.70	35.10	19.60	uW/1s
After layout, with timing switching activity dumped from gate level simulations	**IDLE**	48.19	29.32	17.73	17.22	
	COMMUNICATION, transfer of 6B	47.09	28.08	50.34	29.63	

Percentage consumption of different modes compared to the consumption without use of clock gating is described in Table 21. This is done for better and more concrete results evaluation. Description and evaluation of Table 21 is in chapter 5.2.

Table 21: Power consumption energy savings

Consumption mode	Clock gating type		
	AUTO	MAN	MAN_AUTO
IDLE	39.16%	63.21%	64.24%
COMMUNICATION, transfer of 6B	40.37%	-6.90%	37.07%

Table 22 shows the amount of instances in each design. It is expected that clock gating will have more logic than the case without any clock gating; this can be seen with manual clock gating. On the other hand it is interesting that automatic clock gating and combined clock gating has fewer instances than the case without clock. Obviously DC Compiler uses some kind of optimalization for registers with automatic clock gating done during synthesis than for registers without this kind of clock gating.

Table 22: Number of instances in the design

Clock gating type				
NONE	**AUTO**	**MAN**	**MAN_AUTO**	
1538	1285	1588	1342	**instances**

5.2 Power consumptions results evaluation

5.2.1 Automatic clock gating

5.2.1.1 General

Just by using automatic clock gating the consumption drops to about 60% compared to not using clock gating. This means about 40% of power consumption is saved just by adding one command during the synthesis. So basically, it is very low effort for the designer.

5.2.1.2 Idle and Communication mode compare

Both Idle and Communication mode have approximately the same consumption. This is based on the fact of how the clock gating is done – it is functional clock gating (described in chapter 3.4.1), so basically the same logic is still on most of the time. The interesting thing is that since there are many gating cells that need to be supplied, the consumption in IDLE mode is slightly higher than in communication mode.

5.2.1.3 Summary:

By basically no designer effort 40% of consumption can be saved.

5.2.2 Manual clock gating

5.2.2.1 Idle mode

Here is a significant power saving compared to automatic clock gating done during synthesis. There is 63.21% of saved consumption during IDLE mode, compared to automatic clock gating there was only 39.16% of saved consumption. This result is more than satisfactory and shows how power consumption can be saved with reasonable placement of clock gating cells based on activity modes.

5.2.2.2 Communication mode

Consumption during communication mode is higher by 6.90% than when clock gating wasn't used. One of the usual characteristics of manual clock gating is that maximum momentary consumption is higher than when clock gating is not used, because more cells are in use at one time.

5.2.2.3 Summary:

This mode has high communication consumption, which is higher than without clock gating (6.9% higher); however the consumption in idle mode is lower than in the automatic clock gating. In idle mode 24% more was saved with manual clock gating than in idle mode with automatic clock gating.

5.2.3 Manual + automatic clock gating combination

5.2.3.1 Idle mode

In this mode the consumption saving was 64.24%. This is slightly higher than how much was saved in idle mode with manual clock gating and is caused by the fact that the use of combined clock gating gated some registers that were not gated during manual clock gating.

5.2.3.2 Communication mode

In this mode the consumption saving is 37.07% compared with the consumption without clock gating. This is 3.3% lower than with only automatic clock gating. It is the highest consumption saving in communication mode of all clock gating variants.

5.2.3.3 Summary:

This combination seems like a good compromise between communication mode (30.07% of consumption saved) and idle mode consumption (64.24% of consumption saved).

5.3 Practical examples of use

I prepared the following examples to show how this IP block could be used and how useful for saving consumption it could be with using clock gating. These following examples were chosen on purpose to show an example when the access through DP device would be used often and an example when it would be accessed only in certain intervals (this is closer to the actual use scenario than accessing constantly).

5.3.1 DP IP block as a device assessing a memory

Let's expect that the I2C Master is accessing a memory connected to the APB Master. Expect 70% of time in communication mode.

Consumption = (70% * average communication consumption + 30% * average idle consumption) * time of communication

Table 23: Consumption for use to access a memory

	Clock gating type				
	NONE	**AUTO**	**MAN**	**MAN_AUTO**	
Consumption	47.420	28.452	40.557	25.907	uW/1s
Power consumption savings	-	40.00	14.47	45.36	%

Table 23 compares the different clock gating technique types in an example of accessing memory. The consumption values are in uW per 1 second activity. The power consumption saving values is compared with the case without clock gating use.

The device is able to save 40% of power consumption with automatic clock gating. This was already seen from Table 21.

Manual clock gating in this case is convenient to use when the device stays in idle mode a lot. Here it is expected that it will be in communication mode 70% of time, therefore the manual mode gives the worst results with only 14.47% saved consumption.

Manual clock gating combined with automatic clock gating thanks to the combination of reasonable gating cell placing dependent on the operation as well as the use of logic clock gating round registers gives the best result – 45.36% saved consumption. I would describe this as a very good result.

5.3.2 DP IP block as a device accessing temperature measure unit

Let's expect that the I2C Master is accessing a unit for temperature measuring once every 30 seconds. It sends a 6B command and receives data of 6B. This whole transfer takes approximately 155us.

This means that the device spends 155us in communication mode and 29845us in idle mode.

Table 24: Consumption for use to access a temperature measure unit

	Clock gating type				
	NONE	**AUTO**	**MAN**	**MAN_AUTO**	
Consumption	1438.24	875.06	529.16	513.94	uW/30s
Power consumption savings	-	39.16	63.21	64.27	%

This example better shows the effectiveness of manual clock gating in idle mode. It also demonstrates a use case much closer to the actual use of this IP block than the use case described in 5.3.1.

While automatic clock gating provides the same value of about 40% of saved power consumption, with manual clock gating I achieved 63.21% of saved power consumption. This is a very good result and shows how effective clock gating can be.

Combined clock gating gives a result of 64.27% saved consumption, which is just slightly higher than manual clock gating. These values are close to the values only in idle mode, because the device spends most of its time in idle mode. It also takes fewer instances in the physical design (see Table 22) by about 15%, which can be useful and is

one of the reasons why combined clock gating gives better results. Gating cells are also placed to convenient registers besides that.

5.3.3 Summary

The effectiveness of saved power consumption directly depends on the amount of time the device spends in each mode – in this case the Idle and Communication mode. Each mode has different consumption and it is necessary to take the actual use of the device in account. This is expressed by Ahmdal's law and taking this in account is usually more effective than just trying to lower the consumption in all modes. Focusing on the modes where the device spends most of its use is very important.

6 Summary

6.1 Goals

The goal of this thesis was to design and verify a Slave IP core for transmitting data between I2C and APB buses using low-power consumption techniques and comparing the results of power consumption.

6.2 Low-power techniques

The thesis describes the use of low-power techniques in IP design and compares different techniques and their characteristics that can be used to achieve low-power consumption. The result of the comparison was the selection of clock gating for use in the design.

To be able to compare more results, four different clock gating modes were used – no use of clock gating, automatic clock gating (cells placed during synthesis), manual clock gating (clock tree gating/ cells placed manually) and combination of manual clock gating and automatic clock gating.

6.3 Workflow and power estimations

The workflow starts from specification and goes to physical design. It includes verification at different points of the workflow. Power estimations are run after synthesis, as well as after the physical design.

The power estimations after synthesis are done for a typical clock activity; therefore they're not very accurate. The power estimations after the physical design are accurate, because they count with all the delays in connections. The power estimations after the physical design run in two different modes – idle mode and communication mode. Because of this, the results after physical design are very accurate.

It was necessary to plan what tools to use for the design, since there was usually a limited amount of licenses.

6.4 Verification

A third-party I2C Master was used for the verification to communicate with the I2C Slave designed in the Master's thesis. A behavior model of the APB Master (bridge) was written as a part of the thesis to verify the right transfer of data. The verification was run for all different speeds, including: I2C speed modes 10, 50, 100, 200, 400 kb/s and 1Mb/s to verify compatibility. Self-checking verification tests were used for the verification. Code coverage was also run as well as FSM state coverage and graphical examples of the FSM coverage are a part of the thesis.

6.5 IP core

This IP can be used as a hard as well as a soft macro in the designs. The size of the design was determined by the amount of cells and the technology (65nm). The size is 157x82um, which equals 12874 um^2.

6.6 Results

The saved power consumption estimation results were run for I2C data transfer speed of 1Mbit/s and the results were more than satisfactory.

6.6.1 Automatic placing of the clock gating cells

Automatic placing of the clock gating cells during synthesis generally saves about 40% of power consumption, which is a very interesting and good result. What is even more interesting is that the use of automatic clock gating results in the use of fewer cells in the design – the tools are able to make good use of the logic. The synthesis tool is able to put gating cells even inside the FIFOs (because the FIFIOs are from the same vendor as the synthesis tool), which leads to achieving these results. When using automatic clock gating the clock is disabled for those registers that don't change their value (input is the same as output of the register.

6.6.2 Manual placing of clock gating cells

Manual placing of clock gating cells gave better results in idle mode compared with automatic placing – 63% of power consumption was saved. We can see that reasonable gating cell placement gives good results. On the other hand, in communication mode, the power consumption was 6.9% higher than in the case without the use of clock gating. This is because there is more logic that needs to be driven during communication mode than in the case with no clock gating. This is the typical behavior of clock gating – average power consumption is lower, but maximum consumption is higher.

6.6.3 The combination of manual and automatic clock gating

The combination of manual and automatic clock gating provided the best results. 64% of power consumption was saved in idle mode and 37% in communication mode. The higher power consumption saving in idle mode was achieved thanks to the reasonable manual placement of gating cells that disable clocks for larger blocks (FIFOs). In communication mode, the power consumption saving was achieved thanks to disabling the clock to those registers that don't change their value.

6.7 Conclusion

The results imply that it is convenient to use automatic clock gating along with reasonable manual placement of clock gating cells. Automatic clock gating ensures that the register clock is not enabled unless the value on the input is changed. Manual clock gating makes sure that the clock is disabled for registers that are not needed according to the device mode. The device mode expresses the function of the device in the mode and only the designer knows best what parts of the device are used in which mode.

The outputs of the thesis show power consumption savings results that are more than satisfying. All the requirements of the assignment were fulfilled. In addition to that, I did not finish the project with synthesis, but continued in the workflow to the physical design to obtain more accurate power consumption results for idle and communication mode as post-synthesis power consumption estimations are not very accurate (often 30-70% inaccurate) and they only provide results for a typical clock activity. The power consumption results obtained after the physical design (after the layout) provided very accurate and impressive results.

7 References

7.1 References cited

ARM. 2004. AMBA 3 APB Protocol Specification. *ARM The Architecture for the Digital World.* [Online] August 17, 2004. [Cited: Semtember 8, 2011.] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0024b/index.html.

B.V., NXP. 2007. UM10204. *I2C-bus specification and user manual.* [Online] June 19, 2007. [Cited: September 7, 2011.] http://www.nxp.com/documents/user_manual/UM10204.pdf.

Bečvář, Miloš. 2011. Techniky Návrhu pro Nízkou Spotřebu (Low Power). *Edux FIT ČVUT.* [Online] November 24, 2011. [Cited: March 12, 2012.] https://edux.fit.cvut.cz/courses/MI-SOC/_media/lectures/10/low_power.pdf.

Frank Emnett, Mark Biegel. 2000. Power Reduction Through RTL Clock Gating. *AIEC - Automotive Integrated Electronics Corporation.* [Online] 2000. [Cited: March 12, 2012.] http://www.aiec.com/Publications/snug2000.pdf.

Goering, Richard. 2008. Low Power Design. *Lee Public Relations.* [Online] September 3, 2008. [Cited: December 3, 2011.] http://www.leepr.com/PDF/SCDsource_STR_LowPower.pdf.

Herveille, Richard. 2006. I2C Controller's verilog,VHDL Source code,Testdench. *ASIC.CO.IN , ASIC and VLSI Job Seekers Paradise.* [Online] April 9, 2006. [Cited: October 1, 2011.] http://asic.co.in/projects/i2c_files/i2c.htm.

Murali, Keshava. 2009. Low Power Techniques. *SlideShare.* [Online] July 14, 2009. [Cited: March 12, 2012.] http://www.slideshare.net/shavakmm/lowpowerseminar810.

Synopsys. 2010. Synopsys Low Power Flow User Guide. *Academic Computing & Media Services.* [Online] March 2010. [Cited: March 26, 2012.] http://acms.ucsd.edu/_files/slpfug.pdf.

Yang, Ruixing. 2008. Frequency and Voltage Scaling Design. *Tampere University of Technology.* [Online] December 4, 2008. [Cited: March 12, 2012.] http://www.tkt.cs.tut.fi/kurssit/9626/S08/Chapters_9_10.pdf.

7.2 Other used literature

DAHAN, Nir. The Principle Behind Multi-Vdd Designs. *The Principle Behind Multi-Vdd Designs* [Online]. April 2, 2008. [Cited: April 28, 2012]. http://asicdigitaldesign.wordpress.com/2008/04/02/the-principle-behind-multi-vdd-designs/

APTE, Charwak. Power Gating Implementation in SoCs. *University of California Los Angeles.* [Online]. February 1, 2011 [Cited: April 28 2012]. http://nanocad.ee.ucla.edu/pub/Main/SnippetTutorial/PG.pdf.

JOHNSON, R. Colin. How best to reduce power on future ICs. *EE Times* [Online]. February 21, 2011. [Cited: April 28, 2012]. http://www.eetimes.com/electronics-news/4236645/How-to-reduce-power-on-future-ICs?cid=NL_EETimesDaily

YANG, Ruixing. Frequency and Voltage Scaling Design. *Tampere University of Technology.* [Online]. Tampere, 2008 [Cited: April 28, 2012]. http://nanocad.ee.ucla.edu/pub/Main/SnippetTutorial/PG.pdf. Lecture.

JAKOVENKO, Jiří. Digitální návrh I. *Moodle KME FEL ČVUT.* [Online]. May 5, 2010. [Cited: April 28, 2012]. http://moodle.kme.fel.cvut.cz/moodle/file.php/117/prednasky/07_AMS-Digital-I.pdf

JAKOVENKO, Jiří. Digitální návrh II. *Moodle KME FEL ČVUT.* [Online]. May 5, 2010. [Cited: April 28, 2012]. http://moodle.kme.fel.cvut.cz/moodle/file.php/117/prednasky/08_AMS-Digital-II.pdf

DURGA PRASAD, B.C. ; KRISHNA, N.V.R. Synthesis of a TI MSP430 microcontroller core using Multi-Voltage methodology. *Communication Control and Computing Technologies (ICCCCT).* [Online]. 2010, vol. 93-97 [Cited: April 29, 2012]. DOI: 10.1109/ICCCCT.2010.5670534. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5670534&isnumber=5670438

A. Appendix – Regression report

Below is the regression report of the verification tests. This report was passed for all the different speeds as well as types of clock gating use.

```
==========================================================================
  Regression date: 2012-Mar-27
    Start time   : 2012-Mar-27 10:22 CEST
    End time     : 2012-Mar-27 10:25 CEST
 ==========================================================================
   tc_tx000.v             SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_rx000.v             SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_rx002.v             SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr000.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr001.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr002.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr003.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr004.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr005.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr006.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr007.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr008.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr009.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr010.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_intr011.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_othr000.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
   tc_rxtx000.v           SIMULATION STATUS: PASSED        (CPU:0.1s, mem:41.1M)
 --------------------------------------------------------------------------
 ==========================================================================
  Total of 17 tests, 0 failing.
 ==========================================================================
```

```
==========================================================================
  Regression date: 2012-Apr-18
    Start time   : 2012-Apr-18 15:07 CEST
    End time     : 2012-Apr-18 15:07 CEST
 ==========================================================================
   tc_rx001.v             SIMULATION STATUS: PASSED        (CPU:0.1s, mem:39.0M)
 --------------------------------------------------------------------------
 ==========================================================================
  Total of 1 tests, 0 failing.
 ==========================================================================
```

B. Appendix – Schematics from Novas Verdi

Verdi is a tool developed by Novas to view RTL schematics from Verilog code. The code was also run through this program to avoid some of the look-and-see mistakes and also to prove that the design is actually written according to the description above in this text.

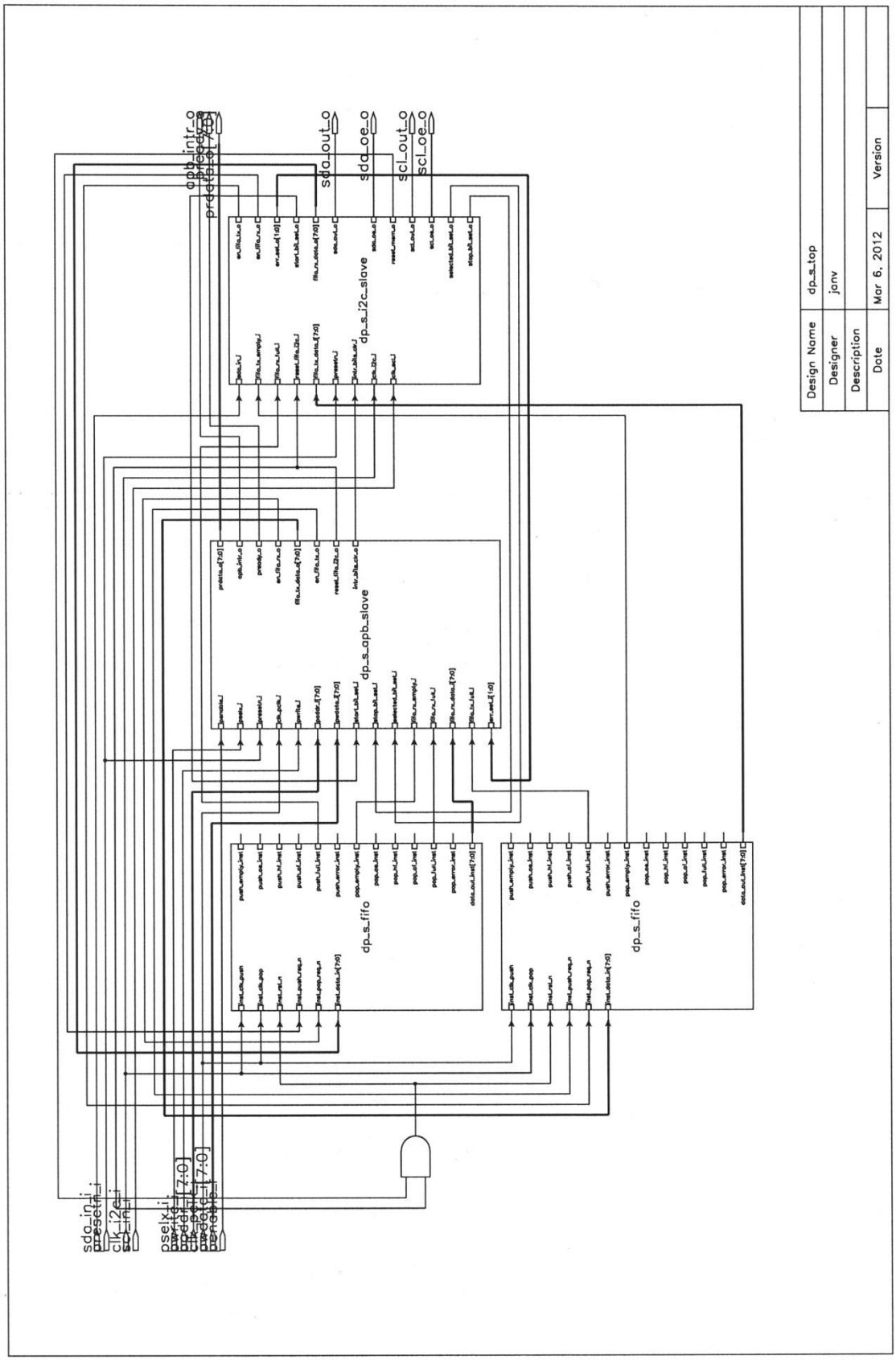


Figure 68: Schematic from Verdi: dp_s_top

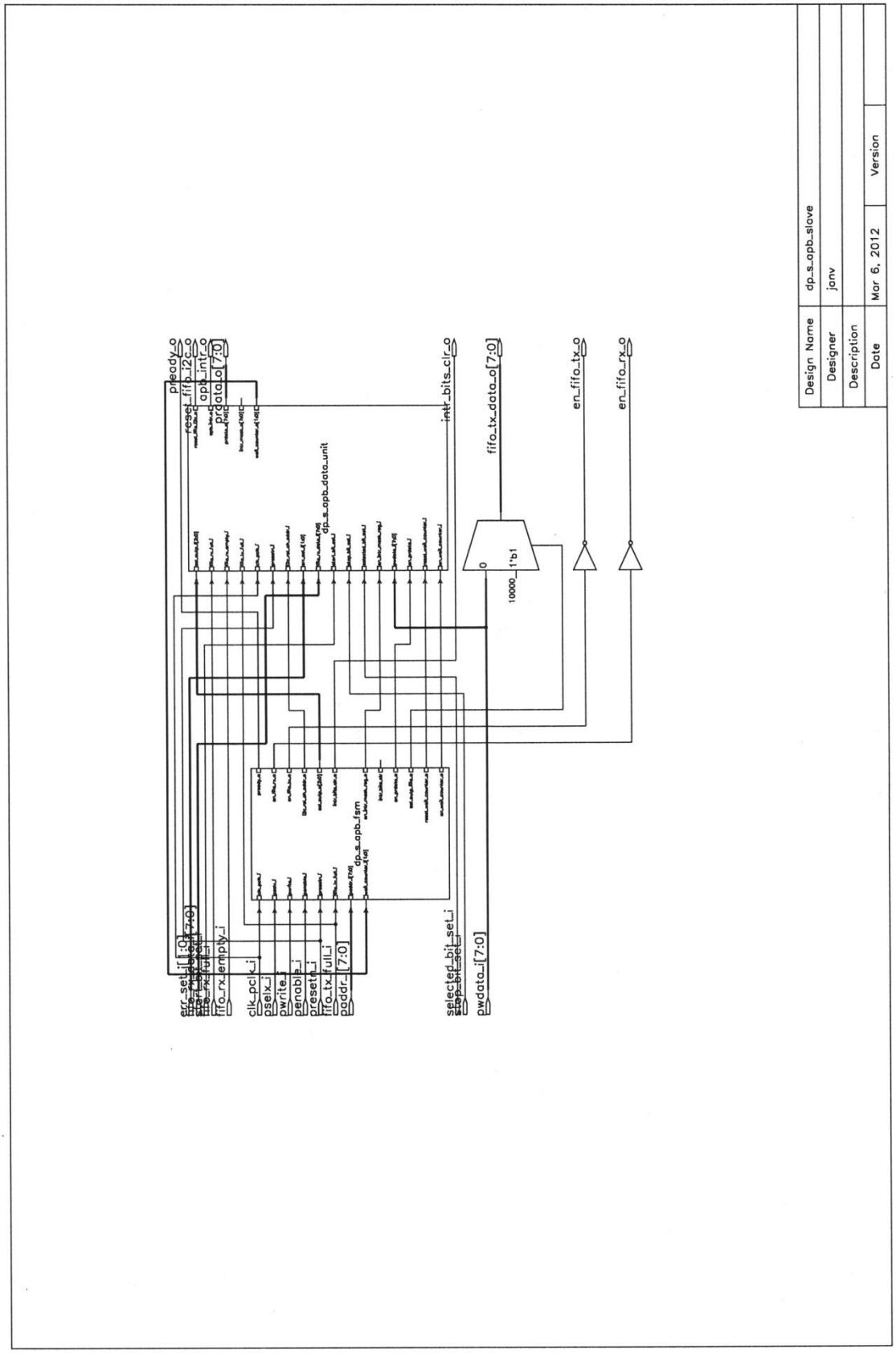


Figure 69: Schematic from Verdi: dp_s_slave

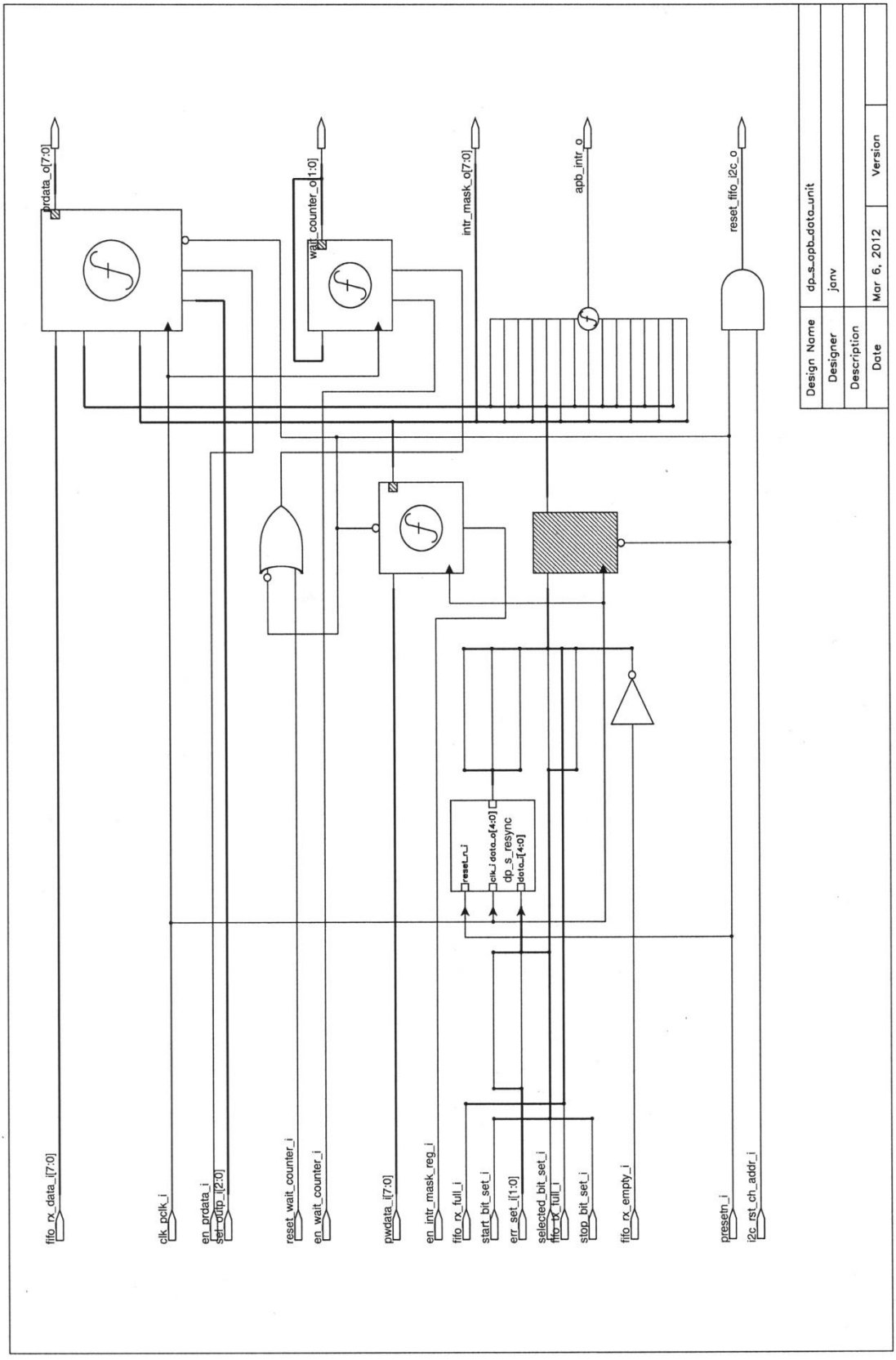


Figure 70: Schematic from Verdi: dp_s_apb_data_unit

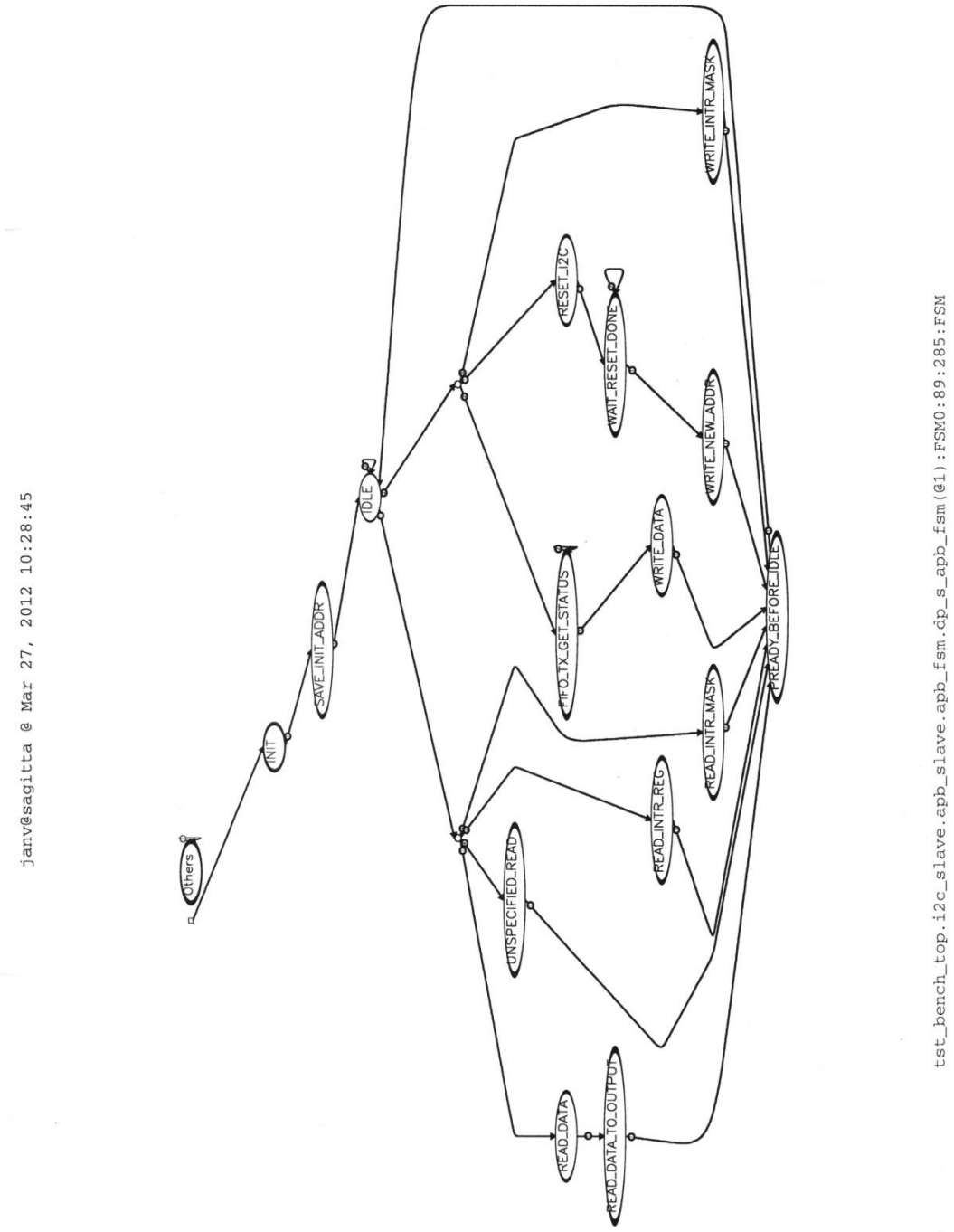


Figure 71: Schematic from Verdi: dp_s_apb_fsm

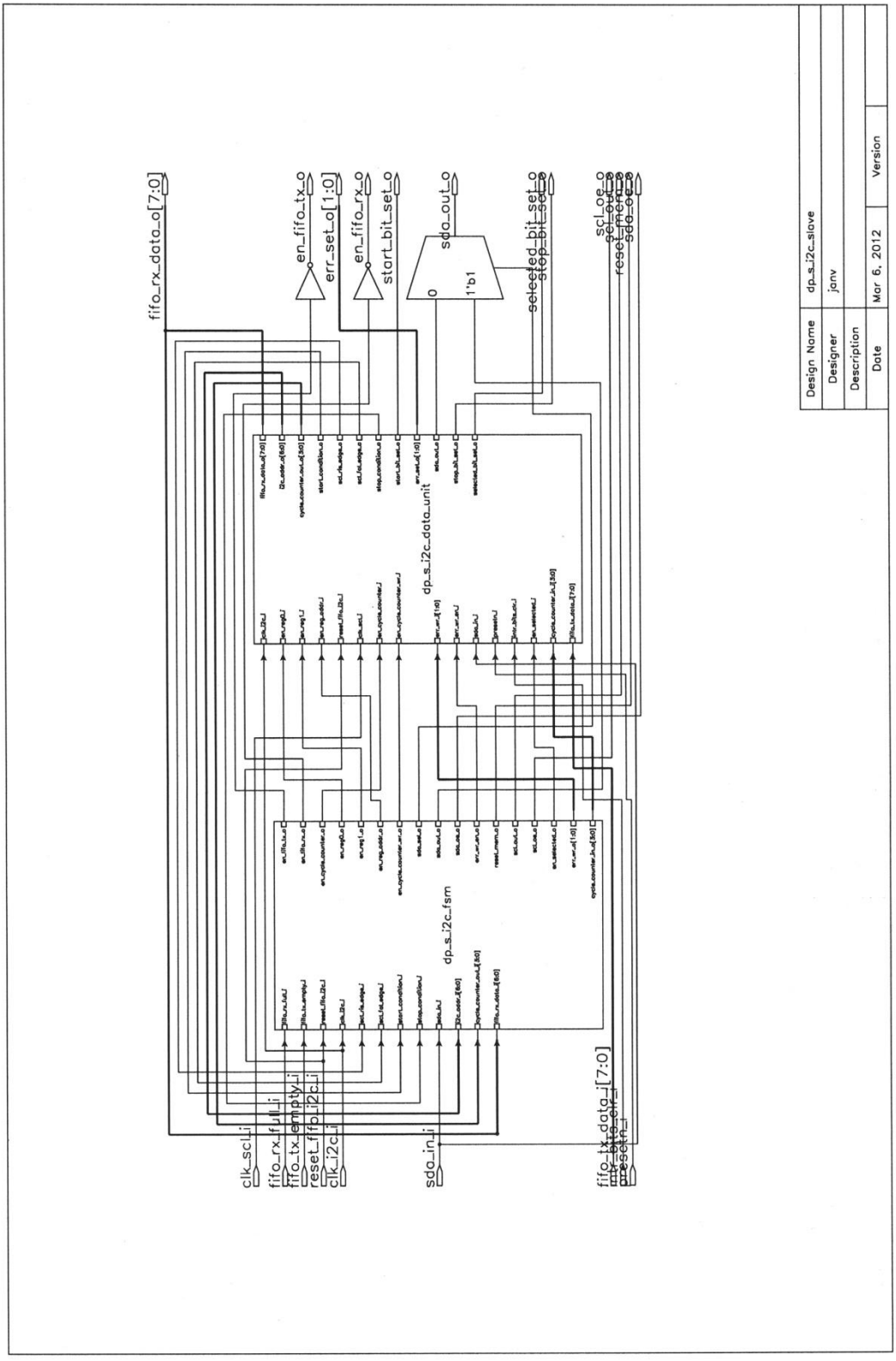


Figure 72: Schematic from Verdi: dp_s_i2c_slave

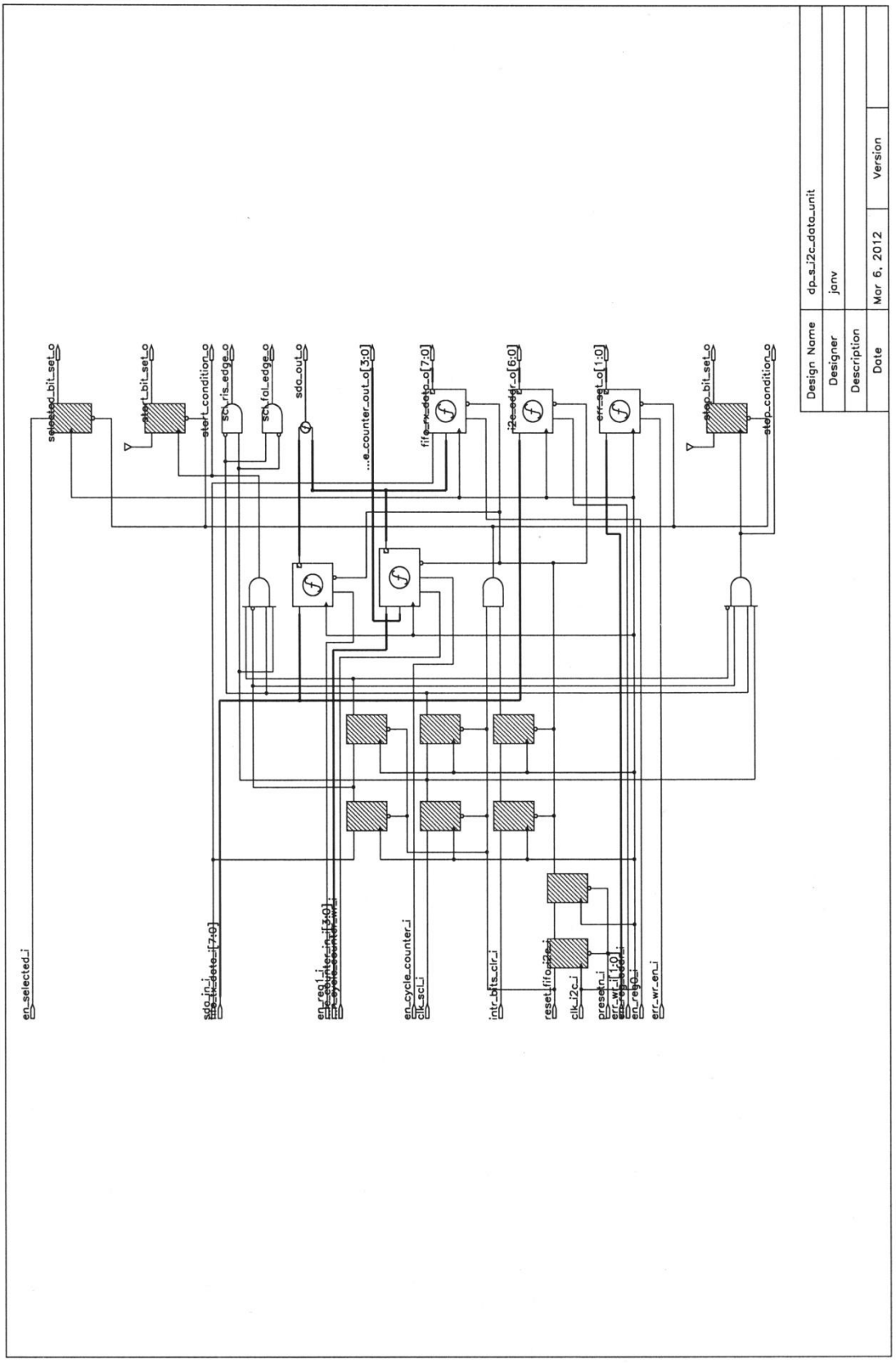


Figure 73: Schematic from Verdi: dp_s_i2c_data_unit

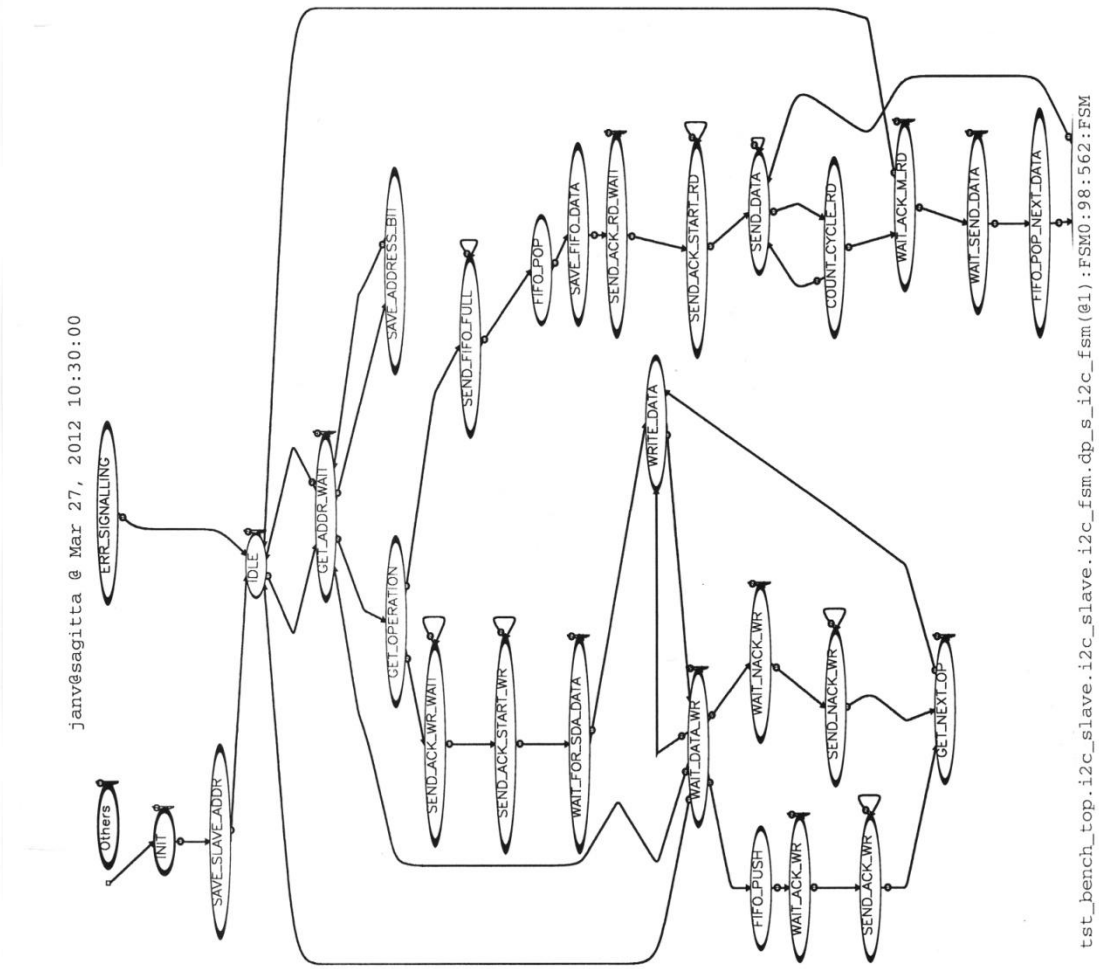


Figure 74: Schematic from Verdi: dp_s_i2c_fsm

C. Structure of the enclosed CD

```
/src
     /RTL
          dp_s_top.v - DP device top level module
          dp_s_i2c_slave.v – I2C Slave top level module
          dp_s_i2c_fsm.v – I2C Slave FSM
          dp_s_i2c_data_unit.v – I2C Slave data unit
          dp_s_apb_slave.v – APB Slave top level module
          dp_s_apb_fsm.v – APB Slave FSM
          dp_s_apb_data_unit.v – APB Slave data unit
          dp_s_global_consts.v  - defines and constants
          dp_s_gating_cell_wrapper.v – wrapper for manually
               placed gating cell
          dp_s_fifo.v – instantiation of asynchronous FIFO
          dp_s_resync.v – resynchronization unit


     /TESTBENCH
          tst_bench_top.v – test bench top file
          wb_master_model.v – third party I2C Master file
          i2c_master_top.v – third party I2C Master file
          i2c_master_defines.v – third party I2C Master file
          i2c_master_byte_ctrl.v – third party I2C Master file
          i2c_master_bit_ctrl.v – third party I2C Master file
          dp_s_pad.v – pad model
          tc.v – verification tests
          tc_tx000.v – code for running test case tc_tx000
          tc_rx000.v – code for running test case tc_rx000
          tc_rx001.v – code for running test case tc_rx001
          tc_rx002.v – code for running test case tc_rx002
          tc_rxtx000.v – code for running test case tc_rxtx000
          tc_intr001.v – code for running test case tc_intr001
          tc_intr002.v – code for running test case tc_intr002
          tc_intr003.v – code for running test case tc_intr003
          tc_intr004.v – code for running test case tc_intr004
          tc_intr005.v – code for running test case tc_intr005
          tc_intr006.v – code for running test case tc_intr006
          tc_intr007.v – code for running test case tc_intr007
          tc_intr008.v – code for running test case tc_intr008
          tc_intr009.v – code for running test case tc_intr009
          tc_intr010.v – code for running test case tc_intr010
          tc_intr011.v – code for running test case tc_intr011
          tc_othr000.v – code for running test case tc_othr000

/text
     dp.pdf          - Master’s thesis in PDF format
     dp.docx         - Master’s thesis in MS Word format
```