FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

BACHELOR'S THESIS

CDC LOGIC VERIFICATION IN RTL DESIGN

Author Elmeri Hekkala

Supervisor Jukka Lahti

April 2023

Hekkala E. (2023) CDC Logic Verification in RTL Design. University of Oulu, Degree Programme in Electronics and Communications Engineering. Bachelor's Thesis, 21 p.

ABSTRACT

This bachelor's thesis provides overview of the clock domain crossing verification in RTL design. Metastability problem and different synchronization methods are explained, and the CDC verification flow is introduced. The work is focused to discuss about different challenges in the CDC verification.

Key words: Clock domain crossing, metastability, synchronization, verification.

Hekkala E. (2023) CDC logiikan tarkastaminen RTL suunnittelussa. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 21 s.

TIIVISTELMÄ

Tämä kandidaatintyö antaa yleiskatsauksen kelloalueylitysten verifiointiin RTL suunnittelussa. Metastabiilisuus ja erilaiset synkronointitavat selitetään sekä CDC verifiointiprosessi esitetään. Työ painottuu tarkastelemaan erilaisia haasteita CDC verifioinnissa.

Avainsanat: CDC, metastabiilisuus, synkronointi, verifiointi.

TABLE OF CONTENTS

FOREWORD

I want to thank Jukka Lahti for supervising this bachelor’s thesis.

Oulu 11.04.2023

Elmeri Hekkala

LIST OF ABBREVIATIONS AND SYMBOLS

CDC	Clock Domain Crossing
IC	Integrated Circuit
IP	Intellectual Property
EDA	Electronic Design Automation
SoC	System on Chip
RTL	Register Transfer Level
CLK	Clock
FF	Flip-flop
FSM	Finite State Machine
FIFO	First-In-First-Out
RAM	Random Access Memory
BB	Black Box
MUX	Multiplexer

FC	Clock frequency
FD	Rate of data changing
TW	Clock's setup-and-hold time
GHz	Gigahertz
ps	Picosecond
MHz	Megahertz

1 INTRODUCTION

Modern large integrated circuits (ICs) contain multiple IPs and blocks, coming from various design teams. Each IP can have several different clocks and designer selects clocks according to common guidelines and to be lowest frequency possible to guarantee the lower power consumption.[1] These blocks create different clock domains to design and when implementing together, signals pass from different clock domain to another. The IP designer needs to synchronize these clock domain crossings (CDC) and confirm with CDC checkers paths to be correctly synchronized. Otherwise, those CDC errors can cause incorrect functionality to microchip.[2] CDC verification is also required to be done at top level to verify correctness of synchronization and communication between different IPs.[3]

The verification can be done with CDC checkers provided by EDA tools.[1] The IP- and top-level CDC verification can be quite challenging due to custom synchronizers and complexity of today's SoCs. CDC verification requires lots of time and effort by engineers to manually check different errors and develop CDC environment and constraints.

Since most designers develops their own synchronizers that gives better performance to their design, CDC tools have difficulties to recognize those different synchronization methods and CDC verification needs more manual effort and time. A study proposes a meta-model synchronization, which is recognized better by CDC tools. The meta-model provided faster verification results in some cases.[1]

The purpose of this bachelor's thesis is to study about clock domain crossing verification, why it is needed and what difficulties it has. At first, metastability, purpose of synchronization and different synchronization methods are explained. The experimental part of the work is based on the use of a commercial software.

Last summer I worked at certain technology company, doing clock and reset domain crossing environment enhancement and maintenance work and CDC analysis and RTL (register transfer level) fixes. This bachelor's thesis is implemented as a literature review and based on my experiences with CDC verification.

2 SYNCHRONIZATION IN DIGITAL CIRCUITS

This chapter goes through fundamentals of RTL sequential logic design, metastability and solutions to deal with metastability. Different synchronization methods are introduced.

2.1 Flip-flops in RTL design

RTL design consists of combinational and sequential logic. Where combinational logic output depends on only the current input, sequential logic depends on prior and current inputs.[4] Sequential logic is very important part of the RTL design. Sequential logic often uses clocks and CDC errors occur in these parts of the design.

Flip-flops are most common building block of sequential logic. To understand the structure of most common D flip-flop, we go through basic SR latch and D latch. Simple SR latch has two cross-coupled NOR-gates, as seen in figure 1.[4] SR latch can also be made with two NAND-gates.[5] When (S=1 and R=0) or (S=0 and R=1), latch is transparent, and it propagates input values Q=S and ¬Q=R. When both S=0 and R=0, outputs remain at their prior values, which means SR latch has memory. S stands for set and R for reset and when set changes to one, it sets Q value to 1 and when reset is asserted it resets Q to zero.[4] When both are asserted simultaneously output is unpredictable and in worst case it can start oscillating.[5]

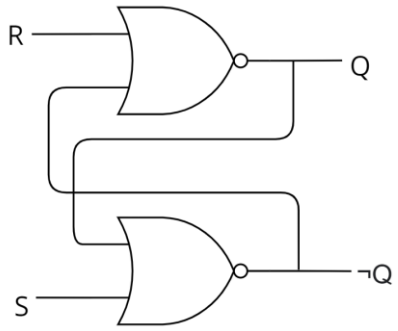

Figure 1. SR latch schematic.[4]

Since SR latch behaves awkwardly when both inputs are asserted simultaneously, D latch fixes this problem by implementing clock to circuit to control when the state should change. D latch implements clock as in the figure 2(a) so that it propagates data signal D to Q when clock is 1. As soon as CLK=1 Q can change as D changes. [4]

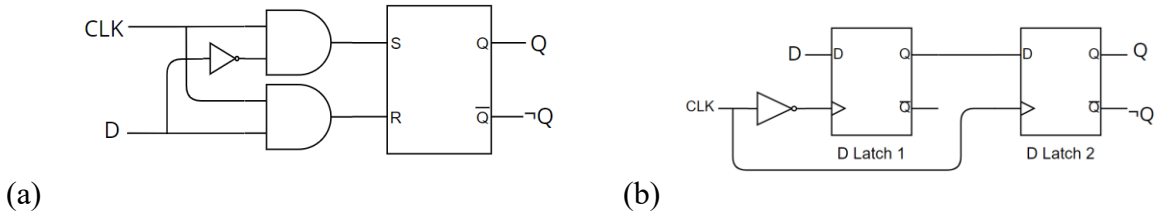

Figure 2. (a) D latch, (b) D flip-flop.[4]

When two D latches are implemented sequentially as in figure 2(b), we get a D flip-flop. The master latch is transparent while CLK=0, and slave holds the same value as before. This way while clock rises master latch is opaque and slave latch reads input value N1 to output Q. This way D flip-flop operates such way that it propagates input signal D value to output Q at the rising clock edge and holds it until the clock rises again. These D flip-flops are most important component of sequential RTL design. With additional AND-gates reset and enable functionality can be implemented to the basic D flip-flop. Registers are bank of flip-flops with same clock, so the flip-flop can be expanded to multibit memory.[4]

2.2 Metastability

Major CDC issue is metastability. Due to unstable behaviour in SR latches when set and reset are asserted simultaneously, it can create fatal consequences in circuit. It can happen if flip-flop's clock and input data signal are asserted simultaneously. That is why input signal must be stable during flip-flop's clock setup and hold times. These times comes from flip-flop's internal structures and when violated the output is unpredictable and causes metastability.[1] Clock setup and hold times are shown in figure 3. The unstable output signal causes erroneous functionality because at the next flip-flop it's value can be estimated to be whichever 0 or 1.

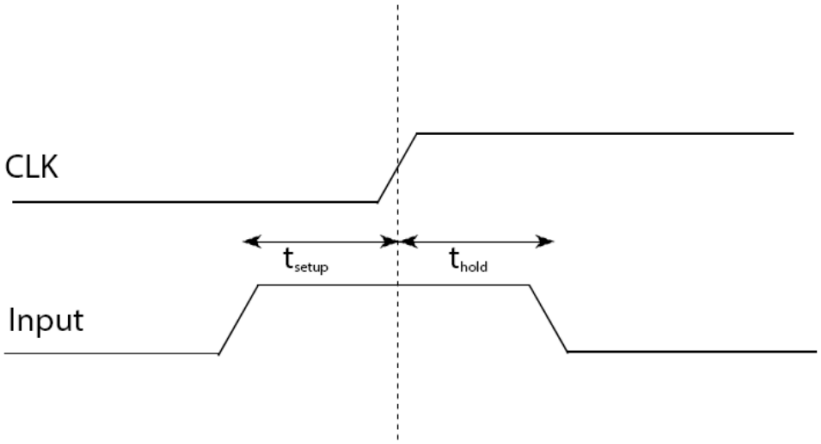

Figure 3. Setup and hold times. [5]

In multi-clock designs metastability is a common problem. When signal crosses from one clock domain to destination domain, it can create metastability at the destination flip-flop.[1] Today's SoCs have so big number of flip-flops and several clock domains with high speeds, so metastability events are common without correct synchronization. A study shows that with simple calculation, assuming the data can change at any time, we can calculate the probability of metastability. Let's assume the clock frequency FC = 1 GHz, clock's setup-and-hold time is TW = 20 ps and rate of data changing FD= 100 MHz. Then rate of metastability becomes Rate = FDFCTW = 2000000 times/sec, which is once in every 500 clock cycles.[2]

In addition to metastability, other serious CDC problems are data loss, glitches capture and incoherent data propagation.[3]

2.3 Synchronizers

Metastability cannot be eliminated so synchronizers must be implemented in design to minimize the effect.[1] Synchronizer is placed between clock domain crossing to add latency, so the unstable signal does not propagate to rest of the design.

2.3.1 Two-flip-flop synchronizer

Two-flip-flop synchronization is basic building block of synchronizers. Asynchronous input from different clock domain goes structure with two flip-flops connected consequently and has same clock. Synchronization structure can be seen in figure 4.

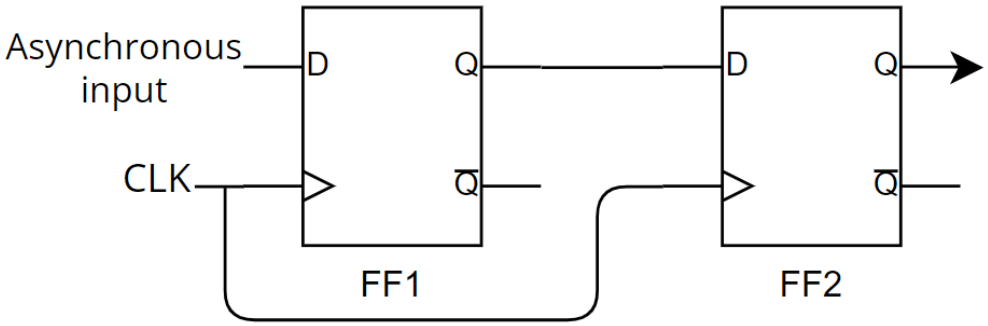

Figure 4. Two-FF synchronization.[2]

If input signal and clock toggles simultaneously, first flip-flop can go metastable. In this case FF1 output Q1 is metastable, but second flip-flop assumes from signal either 0 or 1 and propagates that to output one clock cycle later. This way output cannot be metastable, but it can have false data for one clock cycle. After one clock cycle the output goes to correct value, assuming the input data remains the same. If input signal asserts only for one clock cycle or source clock is much faster than destination one, data can be lost completely in 2-FF synchronizer due to metastability.[2]

2.3.2 Handshake synchronizer

Due to unpredictable behaviour of two-FF synchronizer, it cannot be used to synchronize large multibit data buses crossing to different clock domain. The handshake synchronizer is a common “qualifier-based” synchronizer.[3] Two-flip-flop synchronization is only used to synchronize request and acknowledgement signals between clock domains. The handshake synchronizer is shown in figure 5.

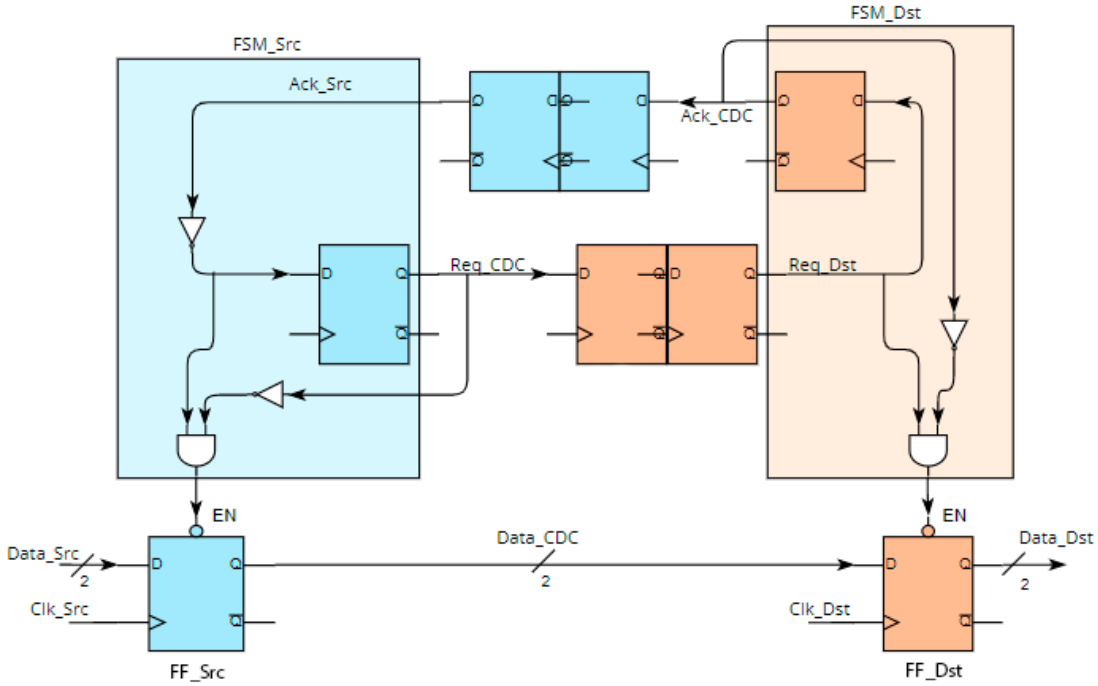

Figure 5. Handshake synchronizer. [1]

Source and destination finite state machines controls the request and acknowledgement signals and enable signals to data bus flip-flops. When new data arrives to source flip-flop, FSM sets enable signal low and request signal high. As soon as destination FSM sends the acknowledgement back it enables the destination data FF and data is propagated forward. New request cannot be sent if no acknowledgement is arrived back to source. [1]

The handshake synchronization structure deals with metastability events and other CDC problems. It is robust method, but it is not very efficient because it has lot of latency due to synchronization of request and acknowledgement signals. [1]

2.3.3 Control-based synchronizer

Control-based synchronizer is like handshake synchronizer, but it has enable-signal only at the destination end. The enable signal is not synchronized either, but it bases on a counter that calculates the ratio between source and destination clock frequencies.[1]

2.3.4 FIFO synchronizer

The faster solution than handshake-based synchronizer is FIFO synchronizer. It is structurally reliable and usually it can be found in predesigned libraries. It consists of dual-ported RAM, where data is stored temporarily. It functions with read and write pointer.[2] Figure 6 shows structure of FIFO synchronizer.

When source is writing data, it sets write enable signal high and if comparator does not give signal that RAM is full, it forwards the data to memory at the location pointed by write pointer. At the destination reader sets read enable signal high and if RAM is not empty it reads data from memory at the location of read pointer. Comparators are located at both clock domains, which compares read and write pointer and gives warning if RAM is full or empty. These are only parts that need to be synchronized with two-FF synchronization. [2]

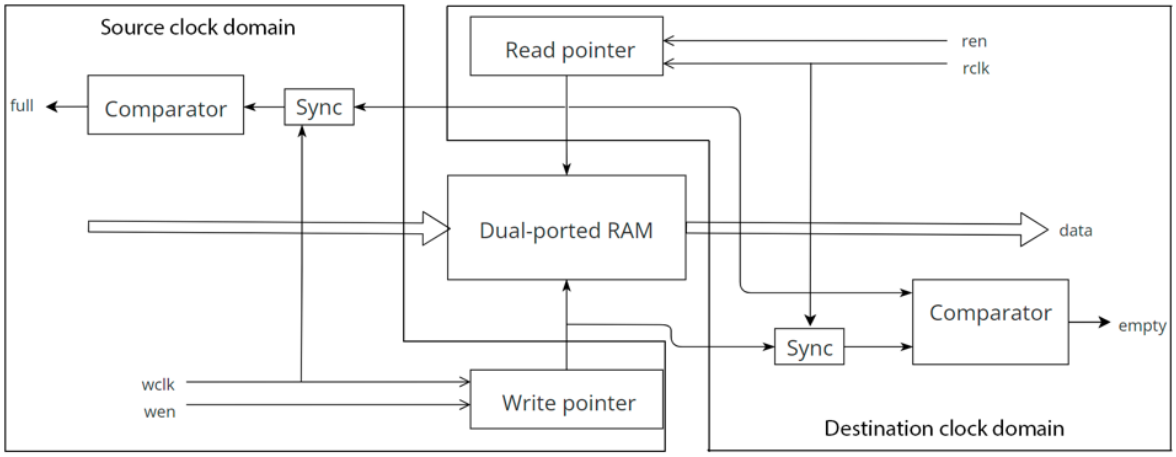

Figure 6. FIFO Synchronizer. [2]

The FIFO synchronizer is very common structure and often designer does not have to design it from the beginning. Designer needs only to choose how big RAM is needed for the design. FIFO synchronizer is quite fast, but to have more efficient and quicker synchronization designer needs to put more effort designing synchronization that is more suitable to specific design. [2]

3 CDC LOGIC VERIFICATION

This chapter goes through CDC logic verification tools and verification flow. The examples of the verification is based on the use of a commercial software.

3.1 CDC Static Checkers

Since CDC errors can be dangerous to design, those need to be verified properly. Verification is done with CDC checkers. There are three main CDC checkers commercially available: Spyglass CDC tools from Synopsys, Questa CDC checker from Siemens and conformal CDC checker from Cadence.[6]

CDC tools can perform structural and functional verification. In structural verification, tool compares the hardware description, which can be RTL or gate-level, and CDC structure pattern library. Tool searches for synchronizers and signals crossing from different clock domains. Based on these it analyses are the synchronization between CDC structures done correctly and gives error reports. The tool analyses only the interconnections between blocks and not functionality.[3]

In the functional CDC verification, the model-checking engine is also used. It checks design's functionality and formal properties on an architectural model created in structural verification. It gives the result proven or failed. [3]

The CDC verification and structure matching is highly dependent on pattern libraries that used tool has. For example, custom synchronizers are easily not recognized by CDC tools, since they have finite structure libraries.

3.2 CDC Verification process

CDC verification requires a big effort from verification engineer by manual error checking and writing constraints. CDC verification is ranked as one of the most difficult parts of RTL verification.[7] Verification engineer needs to have good understanding of the design or needs to consult a lot from designers to correctly write constraints.

In CDC verification, verification engineer needs to identify all clocks and resets in constraint file. It represents the features of the design that the tool can not identify itself. The software reads this file when analysing the RTL code. Tool can have different goals to go through. For example, setup goals checks just if the clock and reset constraints are correctly set. Further goals check for different rules for metastability and other CDC problems, and for synchronization structures.

One type of a block-level verification flow is shown in Figure 7.

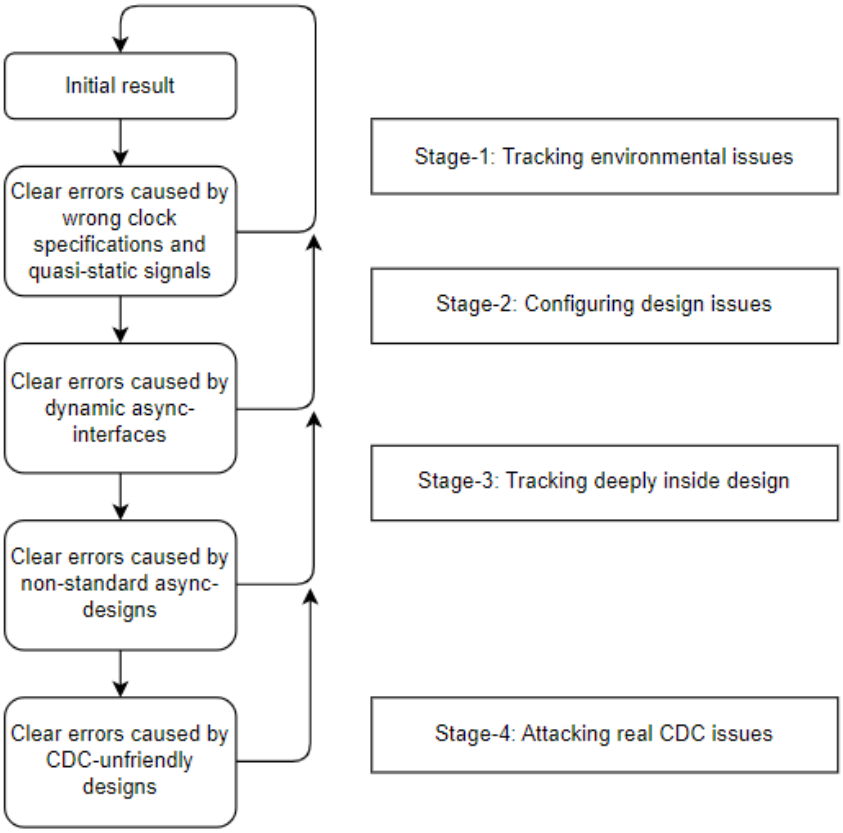

Figure 7. CDC Verification workflow. [8]

CDC verification workflow begins with writing basic constraints and running setup goals. The clock and reset constraints can be autogenerated with tool. In addition, possible black box clocks need to be checked if they are valid clocks or are they propagated from known domain. After setup checks, verification engineer starts running the verification goals and analysing the results.

The number of errors can be very big in the beginning, and it can be difficult to begin looking through those. Most of errors are probably not real errors, but they are caused by incomplete constraints. Design is full of stable signals, which need to be defined as quasi-static, and custom synchronization structures may noy be recognized. This part of the verification can be challenging and time consuming. These are explained more in detail in chapter 4. The workflow presented in Figure 7 is optimized to solve first different false errors in order.

According to the metastability and other CDC errors, verification engineer needs to add constraints to constraint file or fix RTL if errors are real problems in design. False errors can also be waived away with waivers. Waivers simply tells the software to not care about certain error message.

4 CHALLENGES IN CDC VERIFICATION

CDC verification has lot of different issues and error noise is difficult to manage especially at the beginning of the verification process. This chapter explains these problems.

4.1 Issues with constraints and waivers

Waivers and especially constraints are a big part of CDC verification workflow. In large SoC designs constraint and waiver files can grow big due to complexity of design and lack of automatic recognition in CDC tools. Constraints can be difficult to manage and write correctly.

4.1.1 Wrong clock specifications

The setup of CDC verification environment is crucial in the beginning. Block's inputs need to be correctly introduced in constraints. Wrongly specified inputs give huge amount of error messages.

Especially clocks need to be specified well in the constraints. Possible mistakes in constraining are wrong or missing specification of clocks, domain specifications in primary inputs and outputs, and wrong or missing constant values on primary inputs.[8] Those will give many error messages. For example, if input clock is wrongly specified to different clock domain, and it interacts with multiple blocks in domain, the tool will create error message due to clock domain crossings without proper synchronization in all paths to flip-flops and registers that uses different clock domain.[8]

Also, the operational mode (normal or test) needs to be specified correctly.[8] Many designs have different clock inputs and other inputs specified for test purposes. Those clocks usually are selected via MUX with regular clock and set_case_analysis constraints need to be set properly to choose correct clock for CDC analysis. Otherwise, tool gets confused with several different clocks and domains and reports these as domain crossing errors.

It is recommended to become familiar with clocks and domains in the design and try to specify clocks properly in the beginning. Error numbers can rise to overwhelming amounts in the beginning so it can be hard to separate if the errors are originated from just wrong clock or domain specification.

4.1.2 Quasi-static signals

Quasi-static signals also cause lot of unwanted noise in error reports from the CDC verification runs. Quasi-static signals are stable signals that does not change its state in normal operation. Signal changes its state usually in the initial set or when changing modes. These signals usually set clocks, enables/disables portions of design or set the operation modes.[8] These signals do not need synchronization since they are stable.

In example, if static signal passes to different clock domain as in Picture 8, the software would create false error in every register it connects. In this case as signal from Static_reg register is quasi-static and it does not change its state, all errors from this signal are false and unintended. These types of signals can be specified with quasi_static constraint to reduce the noise.

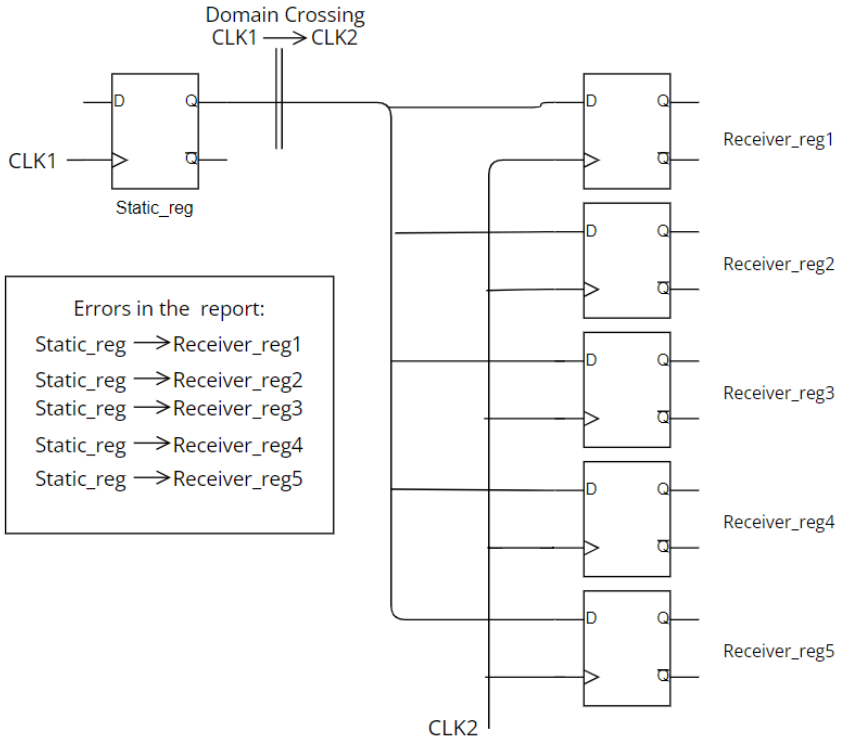

Figure 8. Quasi-static signal.[8]

Quasi-static signals are used in control logic, and it can create false errors in cases where two clock domains cross and are correctly synchronized but quasi-static signal from third clock domain is connected to logic. These quasi-static signals can be used in control logic in data transfer between two clock domains. The synchronization would be correctly recognized without the control signal from third domain, but the signal disturbs the recognition. These issues generate lots of false error reports and verification engineer needs to waste big amount of time to solve what is a problem in the synchronization. [8] Quasi-static signals can be hard to detect from RTL, especially if CDC verification engineer does not have a good understanding of the design.

4.1.3 Name-specific constraints and waivers

Constraints and waivers are written referring to signal and module names. Waivers can also be written referring to error messages. When RTL code is changed, it can have effects in CDC constraints and generate errors in CDC checks that has existed before. Even small changes in parameters and signal names in RTL can set the existing constraints ineffective.

This is a problem especially with waivers. Many times, those are specified such a way that the tool would ignore certain error messages. When signal name or parameters are changed, waivers become ineffective, and number of errors rises.

This problem can be reduced by using wild card characters. These meta-characters can be used to not write complete signals or messages, but just the part and not mentioning the rest. For example, if asterisk (*) matches any number of characters and question mark (?) just one, hierarchical block description “top.\block*” expands to all the different blocks in the design that starts with literal “block”. For example, “top.\block1”, “top.\block2” and “top.\block3” are mentioned with “top.\block*”. Using wild card characters, the size of constraint- and waiver file can be reduced, because constraints and waivers can be written such a way, they have an

effect in multiple parts of the design. Wild card operators need to be used carefully so that they do not affect wrong parts of the design and real errors are waived away.

4.1.4 Verification tool's constraint library

As we have learned, working with constraints is a big part of CDC verification. RTL design has so many correctly synchronized structures that CDC checkers recognized as an erroneous. Writing the constraint file however can require lot of time and effort.

Tool's constraints can be difficult to specify so that is has a correct effect. Verification engineer needs a good understanding of the design but also knowledge of constraint rules. Tools has a large library of different constraints that can be used in different cases. Many constraints need to be specified with different parameters, such as source and destination clock or clock domain names and crossing signal names. When working in the technology company doing block level CDC verification, I needed to use a lot of trial-and-error type of work when using new constraints. The new constraints were difficult to parametrize so that it has the required effect.

Tools has a library of constraints and database how to use constraints. However, this is very time consuming from the verification engineer to figure out how to write constraints correctly.

4.2 Dynamically enabled and disabled interfaces

Many asynchronous interfaces can be enabled or disabled with control signals. Those designs are configured by control registers, primary inputs, or macro definitions.[8] Because CDC verification tools are static, they cannot analyse multiple different cases at once. When modules are disabled, they can create a lot of errors in CDC analysis. When enabled with correctly specified quasi-static signals, checker should not give errors. These structures need to be manually recognized and specified with verification engineer. [8]

Modern SoCs has a lot of this kind of structures to minimize power consumption. Different modules and clocks can be disabled during functional operation. Since CDC tools does not have many capabilities to recognize these structures, the job needs to be done by human.

4.3 Custom synchronizers

As explained earlier, often designer creates modified or completely new synchronizers for power consumption reasons, and to make synchronization to specific design. CDC checkers searches for synchronization methods comparing design to library of synchronization structures. Because this library is finite, often custom synchronization methods are not recognized in CDC verification.[8] Verification engineer needs to manually specify the synchronization blocks which can be tricky. This is a common problem with non-standard asynchronous designs.

4.4 CDC false-error amounts

A study tests with three different project designs the number of CDC errors in different categories. Youngchan Lee, Namdo Kim, Jay B. Kim and Byeong Min separates CDC errors in article *Millions to thousands issues through knowledge based SoC CDC verification* to five different categories: [8]

- Type 1: Wrong clock specifications

- Type 2: Quasi-static signals
- Type 3: Dynamic asynchronous interfaces
- Type 4: Non-standard asynchronous designs
- Type 5: CDC-unfriendly designs

In the study type 1 errors took about 7-28 % of the errors, type 2 40-67% and type 3 16-20%. Type 4 had 6-7% of errors in three projects.[8] This means that about 95% of errors are false and unwanted noise. Especially quasi-static signals and dynamically enabled/disabled asynchronous structures creates a lot of noise. In one project the number of errors only caused by quasi-static signals was 125168.

A study proposes a new workflow to CDC verification where errors are cleared first in order of different error types.[8] This can be seen in Figure 7. This way error count reduces and in the end verification engineer can start to find real CDC problems easier and begin to solve those.

5 DISCUSSION

The purpose of this thesis was to introduce metastability, synchronization methods and clock domain crossing verification. At first, fundamentals of flip-flops, metastability problem and different synchronization methods were explained. After that CDC verification flow was introduced. Finally different issues with CDC verification were discussed.

In this thesis I wanted to explain the problem with clock domain crossings, introduce CDC verification and focus on issues in verification flow. As explained in chapter 4, CDC verification is a lot of verification environment enhancement and maintenance work rather than solving real CDC problems. Most time in CDC verification goes to writing constraint file, reducing false errors, and finding real CDC errors from the noise. Verification engineer needs to have a large knowledge about CDC issues and tools, but also the actual design. Real CDC errors can also be forwarded to designer to solve and fix the RTL.

CDC verification is quite challenging, and I think a lot of work needs to be done to optimize it. Different methods are introduced as synchronization meta-model [1] and specified workflow [8]. Every design is still very different and has new ways to do things and those create new difficulties in CDC verification.

Making this bachelor's thesis has given me a deeper understanding of clock domain crossing problems and verification. I believe that this topic has a lot of possibilities for further research, for example researching most optimal workflow and CDC tool or doing optimization with scripts that automates writing constraint-file.

6 SUMMARY

This bachelor's thesis goes through metastability, synchronization and CDC verification. It is implemented as a literature review and based on my work experience with CDC verification at technology company.

The core problem with clock domain crossings is metastability. When flip-flops clock and input signals are toggled simultaneously, it can create metastability event when output signal is unstable and not 0 or 1. This can happen especially when flip-flop's input signal comes from different clock domain than the clock signal itself. Metastability affects to design's functionality and can have fatal effects. To minimize the effect of metastability, synchronizers are implemented in design between different clock domains. Synchronizers bases on two-flip-flop synchronizer. Different synchronization methods are for example handshake, control-based and FIFO synchronizers.

Since metastability events are dangerous to functionality, clock domain crossings need to be verified correctly. CDC verification is done with CDC checkers. Verification flow contains writing constraint file where all clocks and resets must be specified. A lot of verification flow is to modify the constraint file, for example for quasi static signals. When real errors are found with CDC tool, it needs to be fixed properly in RTL.

CDC verification has a lot of noise in the beginning. CDC checker can have a lot of false error messages due to wrong clock specifications, quasi-static signals, dynamically enabled interfaces, and custom synchronizers. A lot of time goes to writing correct constraints to reduce unwanted false errors.

CDC verification is challenging and time-consuming and would need lot of optimizations. The topic would have a lot of further research possibilities.

7 REFERENCES

[1] M. Kebaili, J. C. Brignone, and K. Morin-Allory, "Clock domain crossing formal verification: A meta-model," *2016 IEEE International High Level Design Validation and Test Workshop, HLDVT 2016*, pp. 136–141, Nov. 2016, doi: 10.1109/HLDVT.2016.7748267.

[2] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design and Test of Computers*, vol. 28, no. 5, pp. 23–35, Sep. 2011, doi: 10.1109/MDT.2011.113.

[3] G. Plassan and M. Kebaïli, "Conclusive Formal Verification of Clock Domain Crossings using SpyGlass-CDC," 2016, Accessed: Feb. 16, 2023. [Online]. Available: http://www.st.com/

[4] D. M. Harris and S. L. Harris, "Chapter 3 - Sequential logic design," in *Digital Design and Computer Architecture*, D. M. Harris and S. L. Harris, Eds., Burlington: Morgan Kaufmann, 2007, pp. 103–165. doi: https://doi.org/10.1016/B978-012370497-9/50004-4.

[5] R. Mehler, "Chapter 7 - Synchronization," in *Digital Integrated Circuit Design Using Verilog and Systemverilog*, R. Mehler, Ed., Oxford: Newnes, 2015, pp. 201–247. doi: https://doi.org/10.1016/B978-0-12-408059-1.00007-X.

[6] A. B. Chong, "Clock Domain Crossing Verification Challenges," *Proceedings - 2021 2nd International Conference on Electronics, Communications and Information Technology, CECIT 2021*, pp. 383–387, 2021, doi: 10.1109/CECIT53797.2021.00075.

[7] A. Yadav, P. Jindal, and D. Basappa, "Study and analysis of RTL verification tool," *2020 IEEE Students' Conference on Engineering and Systems, SCES 2020*, Jul. 2020, doi: 10.1109/SCES50439.2020.9236747.

[8] Y. Lee, N. Kim, J. B. Kim, and B. Min, "Millions to thousands issues through knowledge based SoC CDC verification," *ISOCC 2012 - 2012 International SoC Design Conference*, pp. 391–394, 2012, doi: 10.1109/ISOCC.2012.6407123.