2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM) | 979-8-3315-4172-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/AIIM64537.2024.10934572

# Clock Domain Crossing Aware RTL Synthesis using Fusion Compiler in Commercial Display Chip Design

BolunLi
ESWIN
Xi'an, China
libolun@eswincomputing.com

Weiding Cao
ESWIN
Xi'an, China
caoweiding@eswincomputing.com

Bowei Wu
XUPT
Xi'an, China
xawubowei@chinatelecom.cn

***Abstract* – As the performance requirements of display chips increase, less area, less power consumption, good performance display chip become more critical factors for LCD panel manufacturer. So this puts a lot of demands on chip development. Chip designer need speed up the chip development iteration rate. There are many methods to achieve this goal by the way introduce new problem. For example, to get better performance design house will use same RTL code do process transfer. But same RTL may not suitable in timing aspect for difference design process. RTL designer use high level Synthesis tool create RTL more fastly. Some company may purchase soft encrypted RTL IP from IP vendor. But high level synthesis rtl & encrypted rtl make RTL design unreadable. It increase clock domain crossing risk during design optimization. These give more challenge for design synthesis & implementation engineer.**

**Fusion Compile (FC) is an EDA tool from Synopsys. It combined synthesis stage and place route. Make implementation step more efficiency. More compact implementation method speed up design cycle. More ever It make design timing closure more easily and get good power area. Design house can use this tool achieve difficult design goal.**

**We present a fusion compile design flow. To solve clock domain crossing issue in implementation design stage and apply this flow for one of arithmetic block in commercial display chip. Get good optimize results avoid any clock domain crossing risk.**

***Keywords: Power Performance Area (PPA); Clock Domain Crossing (CDC); Fusion Compile (FC); Intellectual Property (IP); Synopsys Design Constraint(SDC); High-Level Synthesis (HLS); register-transfer level (RTL).***

## I. INTRODUCTION

High-Level Synthesis (HLS) is a design process in digital integrated circuit (IC) development that automates the conversion of a high-level algorithmic description of a system into a register-transfer level (RTL) implementation.It is widely used in digital circuit rtl design and extensively described in these papers [1][2][3][4][10][11][12]. This high-level description is typically written in languages such as C, C++, or SystemC, which are more abstract and easier to work with compared to traditional hardware description languages like VHDL or Verilog. The HLS tool generates the RTL code, which is a detailed description of the digital circuit in terms of registers, data paths, and control logic. This RTL code can then be used as input for traditional logic synthesis tools to produce a gate-level netlist. The HLS has many advantages, Such as it transfer high level code to rtl more fastly and efficiency. HLS allows designers to work at a higher level of abstraction, reducing the time and effort required to develop complex digital circuits. In other words digital circuit designer no need focus on RTL coding instead they may pay more effort on circuit algorithm. Moreover high-level descriptions are often easier to understand and maintain, especially for complex algorithms. In code level, high-level code can be more portable and reusable across different projects and platforms compared to low-level RTL code. Designers can quickly explore different architectural options and trade-offs (e.g., area vs. performance) by modifying the high-level code and re-running the HLS tool such as Simens tool catapult.

Despite it has many advantages, it still faces some technical challenges. While HLS tools have improved significantly, they may still require manual intervention for highly specialized or performance-critical designs.The effectiveness of HLS tools in producing efficient hardware can vary. Designers may need to guide the tool with directives or constraints to achieve optimal results. Let me explain in more detail, RTL need use synthesis tool such as Design Compile or Genus transfer to gate netlist. Different foundry and process has different gate delay. So if mos gate delay unsatisfactory and circuit combination logic too large. It make circuit timing disconvergence. We need synthesis tool put more effort on timing optimization. For synthesis tool combination logic timing optimization. The most direct and effective way is that ungroup method and retime flipflop.

Ungrouping refers to the removal of hierarchical boundaries in a design. In RTL designs, modules are often organized hierarchically for better readability, modularity, and reuse. However, during synthesis, maintaining these hierarchies can sometimes prevent the synthesis tool from performing optimizations across module boundaries. Ungrouping eliminates these boundaries, allowing the synthesis tool to treat the entire design as a flat netlist. By flattening the design, the synthesis tool can perform global optimizations across module boundaries, which might not be possible if the hierarchy is preserved. It enables better resource sharing, logic simplification, and timing optimization. Ungrouping can lead to area savings by eliminating redundant logic or merging equivalent logic blocks across modules. Flattening the design allows the synthesis tool to optimize critical paths that span multiple hierarchical levels. The retime flip-flop method is a technique used to optimize the timing of a digital circuit by repositioning flip-flops (registers) within the design. This method is particularly useful for improving the performance of pipelined designs or meeting timing constraints in critical paths.

However, ungroup and retime flipflop may damage clock domain crossing circuit. Let whole SOC fail. So usually in engineering we don't use these two method from a conservative perspective. So is there a way to best of both worlds, neither can protect CDC logic and fully optimized circuit?

Clock Domain Crossing (CDC) occurs when a signal crosses from one clock domain to another. In digital designs, different parts of the circuit may operate on different clocks, and transferring data between these domains can lead to metastability, data loss, or functional errors if not handled properly. Proper synchronization techniques (e.g., using synchronizers) are required to ensure reliable data transfer across clock domains. It is an important quality indicator as show in [5][6]. CDC issue must clean in each stage of digital circuit design. Circuit designer must have enough understand of self CDC circuit. Design house may buy soft encrypte IP code from IP vendor to speed up development. Some times IP vendor cannot make the CDC circuit clear in they document. It make design CDC issue uncontrollable.

In this paper, we analyzed CDC issue in digital circuit design synthesis stage. Present a design synthesis flow for timing difficulty high level synthesis code. Decrease CDC risk during design optimization. The outline of the paper is as follow, In Section 2, we analysis the problem of current synthesis flow. In section 3, we describe new CDC aware synthesis flow, In section 4, we apply this synthesis flow for one of arithmetic block in commercial display chip. Compare design result to tradition synthesis flow. In section 5, we make a conclusion for the full text.

## II. Problem of Foundation Synthesis Flow

Foundation synthesis flow is described in these paper [7] [8] [9] and include these steps. Elaborate and translate RTL to getch cell. Map getch cell to technical library. Do netlist optimization iteration and initial floorplan and place. Do incremental optimization. Save netlist and give report. In optimization step, we can do these step optimize netlist. Ungroup hierarchical. Do boundary optimization. Merge flipflop. Register replication. Register retime, among them Register retime is most effective way to decrease combination logic between timing path. However, it may break CDC logic and pose a risk to the circuit. Fig 1 shows one arithmetic block CDC circuit in my design, I disable ungroup, boundary optimization and register retime. We can see two back to back flipflop. It is synchronous logic. When enable boundary optimization and register retime. We can got Fig2 result. There are combination logic between two synchronous flipflop. It's CDC violation. To protect CDC circuit, designer need pre instantiation gate cell to build CDC circuit. set gate cell don't touch during synthesis. However, this may introduce some problem for soft IP integration. Next section we will introduce flow to solve it.

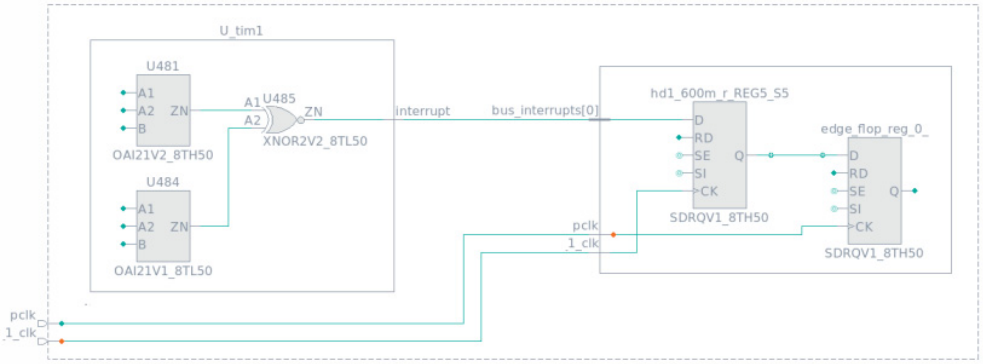


Fig.1. Back to back Flipflop Sync Logic in my block

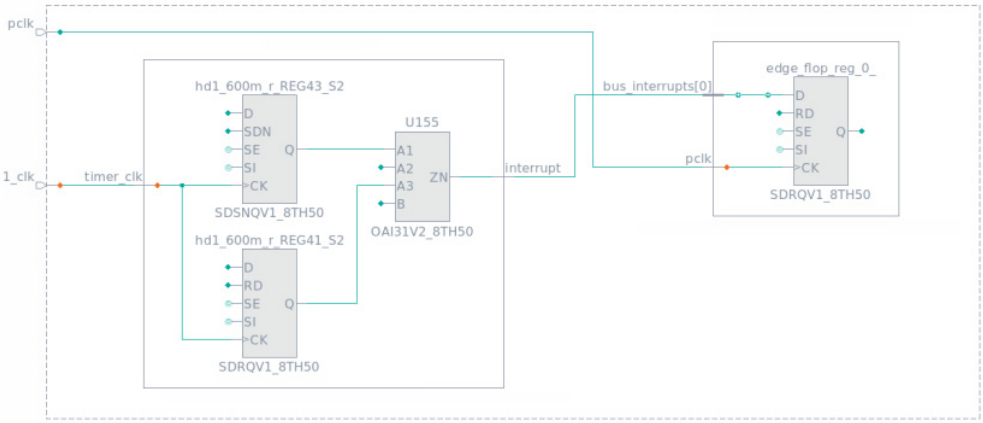


Fig2. Combination logic insert between synchronous flipflop

## III. CDC AWARE SYNTHESIS FLOW

We need let synthesis tool fusion compiler aware CDC netlist. So that CDC netlist can be protected during optimization. Designer can use register retime widely regardless of CDC issue. First we need use rtl code run Spyglass. Synopsys Spyglass is a popular static analysis tool used in the design and verification of digital circuits, particularly for checking design integrity and ensuring compliance with various design rules. One of the key features of Spyglass is its ability to analyze Clock Domain Crossing (CDC) issues in complex digital designs. Spyglass automatically identifies all CDC paths in the design by analyzing the clock domains and the signals that cross between them. The clock domain information is from user defined SDC. Synopsys Design Constraints(SDC) is a standardized format used in static timing analysis. Static timing analysis is described detailed in book [13] . The SDC writing of this article is strictly based on this. But in Spyglass , we not use SDC do timing analysis . Tool get design clock domain from SDC then detect signals that transition from one clock domain to another. The tool checks whether appropriate synchronization techniques (e.g., multi-flop synchronizers, FIFOs, or handshake mechanisms) are used for CDC path. If not, tool will generates detailed reports highlighting CDC issues. Designer need fix it. If circuit has appropriate synchronization techniques tool will put these CDC module information to binary file (we call it CDC restriction file) in order to protect them during synthesis. The Spyglass and protect file generation flow is shown below. I annotate each command to make it easier to understand.

```
#set your technology db
set link_library “tech.db”

#set your verilog search path
set search_path “. $search”

##read your RTL
analyze -format sverilog -define $define “$sv_filelist”

##elaborate your design
elaborate TOP_DESIGN

##spyglass configuration setting
configure_cdc_nff_sync -allow_combo_logic yes
configure_cdc_glitch -multi_src_check_type all
configure_unconstrained_ports -all_bbox -input_model auto -output_model auto -auto_config {disable_virtual_diff} -user_inferred_domains
configure_unconstrained_ports -input_model auto -output_model no_cross -use_inferred_domains
configure_cdc_nff_sync -allow_combo_logic yes
configure_cdc_data_sync -enable_syncpoint_reporting true

###read analysis cdc sdc
source timing_constraint.sdc

##set fusion compiler install path
configure_static_aware_synthesis -app cdc -enable -fc_path /your/fusioncompiler/install_path

###check cdc and create database used for synthesis
###tool will create a dir vcst_rtdb which saved circuit cdc information.
check_cdc

###give cdc report
report_cdc -tag CDC_* -verbose -file report_cdc.log
```

After got CDC database then we need read this information during fusion compiler optimization the flow.
The old and new flow is shown below. We can see old flow we disable boundary optimization. Ungroup and register retime. In new flow CDC restriction file reassure us to use these optimization method.

```
##old flow
set_app_options -name compile.flow.boundary_optimization -value false
set_app_options -name compile.flow.autoungroup -value false
set_optimize_registers false
compile_fusion -to final_opto

##cdc aware fusion flow
set_app_options -name compile.flow.boundary_optimization -value true
set_app_options -name compile.flow.autoungroup -value true
set_optimize_registers true
##read cdc information created by spyglass
set_cdc_restrictions -input /yourcdc/database/vcst_rtdb/static_aware_synthesis_db -verbose > FC_restrictions.log

compile_fusion -to final_opto
```

Next section we will use old and new flow for one of arithmetic block in commercial display chip. In a business matter I can't elaborate on what design is this. But I can explain is that HLS is widely used in arithmetic block. Such as DSP applications, audio and video processing, where complex algorithms need to be implemented efficiently in hardware. The CDC synchronizer logic has been insert correctly. We need to optimize it sufficiently without touching CDC logic. After that we will give an analyze for the result.

## IV. RESULT AND ANALYSIS

We apply cdc aware synthesis flow in one of arithmetic block in commercial display chip. Old flow with boundary optimization ungroup, register retime disable. The reason I have explained in section 3. CDC aware flow let me free my hand to optimize netlist.

I almost ungroup all design hierarchical. Tool can move combination logic between hierarchical. Register retime is apply on full block. From TABLE I, which include two method synthesis and place route result. We can see that block cell total area decrease from 4922056.04um2 to 4676910.42um2. The

design average combination logic level is decrease due to retime flipflop. With the combination logic level decrease the LVT ratio is also decrease 5%.

Both version design floorplan shape size is freeze. The old result design density as high as 75.9% after place. There are no enough space for routing and timing optimize. It directly cause different timing group A_1group, A_2 group A_3_group, REG2REG group degraded after clock tree synthesis and timing eco (STA stage). The new flow the block total cell area decrease significantly. So the place density is better that old flow which is 62.35%. After clock tree synthesis and timing eco. The total violation path is less than one thousand. The WNS is less than 100ps. It means this complex and timing tightly arithmetic block is nearly timing clean and signoff with the help of CDC aware synthesis flow. It certainly that this result is design based. But CDC aware synthesis flow can make sure designer can optimize netlist sufficiently regardless of CDC issue.

TABLE I. CDC AWARE SYNTHESIS PR STA RESULTS WITH BOUNDRY OPTIMIZATION UNGROUP REGISTER RETIME ENABLE

| | **Ungroup false Retime false Boundary opt false** | | **Ungroup true Retime true Boundary opt true CDC AWARE** |
|---|---|---|---|
| **Syn stage** | **Fusion Compiler** | | |
| **Syn logic area** | | 4922056.4 | 4676910.42 |
| **Syn timing(ns)** | Reg2reg | -0.19/500 (WNS/NVP) logic level 40 | -0.04/90 (WNS/NVP) logic level 33 |
| **Cell usage** | lvt | 50.16% | 45.75% |
| **PR stage** | **Innovus** | | |
| **Block fp area** | | 750490.52 | 750490.52 |
| **place** | density | 75.9% | 62.35% |
| **Place timing(ns)** | A_1 group | -0.223/188 | -0.013/117 |
| | A_2 group | -0.076/403 | -0.07/184 |
| | A_3 group | -0.045/1713 | -0.079/1532 |
| | Reg2reg | -0.56/200 | -0.25/100 |
| **Clock_opt timing(ns)** | A_1 group | -0.395/585 | -0.114/379 |
| | A_2 group | -0.369/1276 | -.148/861 |
| | A_3 group | -0.338/2163 | -0.185/932 |
| | Reg2reg | -0.387/16 | -0.273/84 |
| **STA stage** | **Prime Time** | | |
| **STA s125 corner (WNS/NVP)** | Reg2reg | -0.345/2050 | -0.56/850 |

## V. CONCLUSIONS

HLS generated arithmetic block is widely used in commercial display chip. But HLS code is hard for timing closure. We need synthesis tool fully optimize. But widely timing optimization may destroy CDC logic in design. So we present fusion compile CDC aware synthesis flow. It can protect CDC circuit during widely timing optimization. After apply this flow, we get more better timing and less area. CDC aware synthesis flow may get better performance in more complex high level synthesis algorithmic design.

In the field of artificial intelligence chip. HLS is increasingly used to design custom hardware accelerators for machine learning algorithms, which require high computational throughput. AI workloads (e.g., training and inference) require specialized architectures like GPUs, TPUs, or NPUs, which are more complex and more larger chip size than traditional CPUs. AI models are growing exponentially in size (e.g., large language models like GPT), requiring chips to scale in performance without a proportional increase in power consumption. All of this need chip fully optimization. The scheme in this paper provides a good direction for the development and optimization of these chip.

## REFERENCES


[1] Bertrand Le Gal; Caaliph Andriamisaina; Emmanuel Casseau "Bit-Width Aware High-Level Synthesis for Digital Signal Processing Systems" 2006 IEEE International SOC Conference

[2] Paulo Garcia "Preserving Power Optimizations Across the High Level Synthesis of Distinct Application-Specific Circuits" 2024 Tenth International Conference on Communications and Electronics (ICCE)

[3] Wei, C. , Li, G. , & Zhang, X. . Low power design method in high level synthesis with multiple voltages. 2010 5th IEEE Conference on Industrial Electronics and Applications. IEEE.

[4] Preeti Ranjan Panda; Namita Sharma; Srikanth Kurra; Khushboo Anil Bhartia; Neeraj Kumar Singh" Exploration of Loop Unroll Factors in High Level Synthesis" 2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)

[5] Karimi, N. , Chakrabarty, K. , Gupta, P. , & Patil, S. . (2012). Test generation for clock-domain crossing faults in integrated circuits. Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium.

[6] GUPTA Vrinda, KASIM Mohammad, & JEBIN Mohandas. (2021). Towards Improving Clock Domain Crossing Verification for SoCs. Journal of Electrical and Electronics Engineering, 14(2), 19–24.

[7] Gayathri, S. , & Taranath, T. C. . Rtl synthesis of case study using design compiler. Dept. of ECE, Sri Jayachamarajendra College of Engineering, Mysuru, Karnataka, India;Dept. of ECE, Sri Jayachamarajendra College of Engineering, Mysuru, Karnataka, India;.

[8] Pham, V. K. , Le, G. L. N. , Phu, Q. H. , & Thi, T. K. T. . Power Consumption Reduction for VLSI Design Using DC and ICC EDA Tools and CMOS 32nm EDK Synopsys Technology. 2024 7th International Conference on Green Technology and Sustainable Development (GTSD). IEEE.

[9] Suresh, K. , Isaac, E. E. , & Rajasekharababu, M. . Energy-aware system design compiler methods for multiprocessors and Voltage Scaling/frequency. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). IEEE.

[10] Garcia, P. . Preserving Power Optimizations Across the High Level Synthesis of Distinct Application-Specific Circuits. 2024 Tenth International Conference on Communications and Electronics (ICCE). IEEE.

[11] Guimaraes, N. , Saquetti, M. , Rodrigues, P. , Cordeiro, W. , & Azambuja, J. R. . Enabling Programmable Data Planes with C++ and High-Level Synthesis for Custom Packet Forwarding. 2024 37th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI). IEEE.

[12] Xu, H. , Hu, H. , & Huang, S. . (2024). Optimizing high-level synthesis designs with retrieval-augmented large language models. IEEE.

[13] Taraate, & Vaibbhav. (2016). Static timing analysis. Springer India, 10.1007/978-81-322-2791-5(Chapter 11), 277-298.