Project 2 Report

Date: 10-30-2023
MODULE I: SMARTTIME FOR SMARTFUSION2
Submitted by: Kanin McGuire, Shrinithi Venkatesan, Harinarayanan Gajapathy

Module 1

1.1. Summary
The primary aim of Module 1 was to conduct timing analysis on three distinct designs: a 32-Bit Shift Register, a Dual Edge Clock Design, and a 16-Bit Binary Counter using Libero SoC v12.6. Following Maximum and Minimum Timing Delay Analyses, the Fmax values achieved for these designs were 505.051 MHz, 295.334 MHz, and 377.644 MHz, respectively.

1.2. Objectives
The objective of this module was to become acquainted with the Microsemi FPGA development process and to gain experience in achieving timing closure with the SmartTime Static Timing Analyzer (STA). Module 1 provided inputs to work with Libero SoC v12.6 and to learn the intricacies of designing hardware using SmartDesign. It also offered insights into Timing Analysis and guided participants in evaluating system timing using SmartTime.

1.3. Procedure
Module 1 is organized into three sections, each focused on a specific design:

32-Bit Shift Register:

1. Initiated a project in Libero SoC, configuring it with the necessary project settings.
2. Imported the Verilog file and completed the project setup.
3. Carried out synthesis, placement, and routing in Libero, running Synplify Pro and compiling the project with default settings.
4. Established pin assignments using I/O Attributes and I/O Advisor.
5. Applied the necessary clock constraints and executed Timing-Driven Placement and Routing.
6. Performed both Maximum and Minimum Timing Delay Analyses.
7. Adjusted the timing constraints as needed until timing violations and negative slack were observed.

Dual Edge Clock Design:

1. Started by creating a project in Libero SoC, ensuring the project settings were configured as required.
2. Imported the Verilog file and completed the project setup.
3. Proceeded with synthesis, placement, and routing in Libero, running Synplify Pro and compiling the project with default settings.
4. Defined pin assignments using I/O Attributes and I/O Advisor.
5. Applied the necessary clock constraints and executed Timing-Driven Placement and Routing.
6. Conducted a Maximum Timing Delay Analysis.
7. Utilized Cross-Probing to select the cross-probe path within the Libero tool and used Chip-Planner to observe the highlighted cross-probed path.
8. Finally, generated the timing report for both clock edges.

16-Bit Binary Counter:

1. Initiated a project in Libero SoC, ensuring it was set up with the necessary project settings.
2. Imported the Verilog file and completed the project configuration.

3. Proceeded with synthesis, placement, and routing in Libero, running Synplify Pro and compiling the project with default settings.
4. Established pin assignments using I/O Attributes and I/O Advisor.
5. Imported the timing constraint file (.SDC) with a timing constraint of 5.714ns. Paid attention to this timing constraint.
6. Applied the required input and output delay constraints.
7. Executed the placement and routing operation and observed the generated Fmax.

1.4. Test Results

32-Bit Shift Register:

Fig 1.1: Successful compilation and observation of LUTs and percentage utilization(0.26%) in compilation report.

Maximum Timing Delay Analysis:

Fig 1.2: Fmax was recorded at 505.051 MHz
Fig 1.3: The Expanded Path View.
Fig 1.4: The Register to Register Display.

Minimum Timing Delay Analysis:

Fig 1.5: Fmax same at 505.051 MHz
Fig 1.6: The Expanded Path View.
Fig 1.7: The Register to Register Display.
Fig 1.8 Observed output without clock constraint
Fig 1.9, 1.10 For a clock frequency of 1000 MHz, negative slack/timing violations were observed.

Dual Edge Clock Design:

Maximum Timing Delay Analysis:

Fig 1.11: Successful compilation and observation of LUTs (8 used) and percentage utilization(0.31%) in compilation report.
Fig 1.12 Fmax achieved a value of 295.334 MHz.
Fig 1.13: The Register to Register Display.
Fig. 1.14: The Expanded Path View.
Fig 1.15: Chip-Planner with Netview.
Fig 1.16 The Timing Report.

16-Bit Binary Counter:

Maximum Timing Delay Analysis:
Fig 1.17 Successful compilation and observation of LUTs (18 used) and percentage utilization (0.15%) in compilation report. .
Fig 1.18 Pin Assignment Viewer
Fig 1.19 Fmax was measured at 377.644 MHz
Fig 1.20 Observed outputs without Clock Constraints
Fig 1.21, 1.22 Added input and output delay constraints .
Fig 1.23 Fmax after constraining clock
Fig 1.24 Report overview after clock constraining

1.5. Lessons Learned

1. We have learned that timing analysis is a critical aspect of digital circuit design. Conducting both maximum and minimum timing delay analysis is essential to ensure that the design complies with the required timing constraints, effectively eliminating timing violations.

2. Learned how frequency factors in timing analysis affects the slack timings and to analyze which ideal ranges are good enough to maintain a positive slack. The addition of input and output delay constraints significantly contributes to the performance optimization of the design.
3. The use of Cross-Probing proved to be a valuable technique for identifying timing issues and optimizing the overall performance of the design.

1.6. Conclusion

The successful implementation of digital circuits using FPGA hinges on the application of timing analysis and optimization techniques like dual-edge clock design and input/output delay constraints. These methodologies play a crucial role in enhancing performance and ensuring that the design aligns with its specified requirements. This module has served as a valuable resource for gaining familiarity with the Microsemi FPGA development flow and getting started with Libero as a tool.

Module 2

2.1. Executive Summary

Module 2 focused on the development of applications using the Microcontroller Sub-system (MSS) integrated into the SmartFusion2 SoC device. The module entailed the creation of a small application utilizing GPIO pins and the development of software to control these pins. This effort yielded an Fmax of 190.367 MHz and a logic utilization of 0.02%. In summary, the module demonstrated that application development for the MSS on the SmartFusion2 SoC device is straightforward with the use of Libero SoC.

2.2. Objectives

The primary objective of Module 2 was to gain proficiency in building applications with the Microcontroller Sub-system found within the SmartFusion2 SoC device. The module involved the creation of a small application employing GPIO pins and the development of software to control these pins. Additionally, the module aimed to educate participants on how to construct higher-level components using System Builder, without the need to delve into the lower-level HDL code responsible for driving the programmable logic.

2.3. Procedure

1. Developed a sample application that utilized GPIO pins to showcase the functionality of the Microcontroller Subsystem within the Microsemi SmartFusion2 SoC device.
2. Configured 8 GPIO pins as outputs within the system builder after initializing the project in Libero SoC.
3. Assigned the pin numbers of 8 on-board LEDs to each of the GPIO output pins, referencing the SoC device schematics.
4. Compiled and synthesized the design, generated a bit-stream, configured the firmware cores, and exported the firmware for programming the board.
5. Established a project in the SoftConsole workspace, configuring it for both debug and release modes for the software component.
6. Developed software code that, after programming the board, orchestrated a startup sequence wherein the LEDs alternated between turning on and off repeatedly, followed by an endless execution of a specific pattern.
7. Configured a push button to change the direction of LED behavior by adding another GPIO pin as an input and associating it with a push button according to the schematic.
8. Implemented a debouncing algorithm in the software to ensure accurate button input recognition.

2.4. Test Results

1. Fmax - 190.367 MHz
2. Input arrival time - 2.188 ns
3. Clock to Output - 10.4 ns
4. % Utilization - 0.02 %
5. The values of r13, r15, and memory addresses 0x00000000 and 0x00000008 are captured in the appendix section.

Behavior of board: The board exhibited the expected behavior once the FPGA was programmed. After flashing the application code, the LEDs performed a startup sequence, blinking eight times. Subsequently, the code entered a superloop where the LEDs blinked in a pattern as if they were moving in a single direction. During this process, pressing the push button resulted in a reversal of LED blinking direction. Notably, debouncing was implemented in a non-blocking mode, and the push button was polled without generating an interrupt. Therefore, when the LED pattern was executing, pressing the button did not immediately change the direction, but with a longer press, the direction eventually reversed
Code in appendix reference

2.5. Lessons Learned

1. The System Builder in Libero SoC is a powerful tool for efficiently configuring the Microcontroller Subsystem (MSS) of the SmartFusion2 SoC device.
2. System Builder simplifies the task of mapping physical pins to configured GPIO pins, enhancing the setup process, especially when referencing schematics.
3. The SoftConsole Integrating Development Environment (IDE), built on Eclipse,
4. Integrating with the FPGA fabric and becoming familiar with the FPGA development workflow, programming via JTAG interface, and testing and debugging the system.
5. The combination of Libero SoC and SoftConsole streamlines the configuration and utilization of the SmartFusion2 SoC device.

2.6. Conclusion

The development of a sample application involving GPIO pins illustrated the ease and speed with which the microcontroller subsystem can be configured using the System Builder within Libero SoC. System Builder's capability to assign pin numbers to configured GPIO pins simplifies the process of bringing up the entire application, particularly when utilizing schematics. While this application focused on GPIO pins, System Builder is versatile enough to support the development of even more complex systems. The creation of a software project to control the microcontroller subsystem on programmable logic was a seamless process. SoftConsole IDE, based on Eclipse, proved user-friendly. In summary, the combination of Libero SoC and SoftConsole facilitates the configuration and usage of the MSS in the SmartFusion2 SoC device, making it an excellent choice for embedded application development on programmable logic.

Module 3

3.1. Executive Summary

Module 3 revolved around the creation of a 16-bit counter module in Verilog and its integration into a top-level design for RTL simulation. The aim was to estimate the maximum number of counters that could be accommodated on Altera MAX10 DE10-lite, DE1-SoC, and Microsemi SmartFusion2 Maker kit boards using the FOR...GENERATE statement. The assessment indicated that the DE10 Lite and DE1 SOC boards could support a higher number of counters compared to the SmartFusion 2 board.

3.2. Objectives

The primary objective of this module was to develop a 16-bit counter module in Verilog and incorporate it into a top-level design for RTL simulation. Additionally, the module aimed to employ the FOR...GENERATE statement to estimate the maximum number of counters that could be accommodated on Altera MAX10 DE10-lite, DE1-SoC, and Microsemi SmartFusion2 Maker kit boards. The design was to be compiled using Quartus and Libero, with a focus on determining the maximum number of counters achievable on the DE1-SoC board while ensuring timing closure.

3.3. Procedure

1. Developed a 16-bit counter module featuring Clock and Enable input ports, as well as Terminal Count and Q output ports.
2. Verified the counter design using ModelSim Altera, with a particular emphasis on monitoring the Terminal Count signal.
3. Created a top-level entity that generated counters through the "for generate" construct in Verilog HDL and interconnected the counters.
4. Tested the counter design by linking one of the output signals to LED0 on the DE10 Lite. Verified with simulation on the DE1 SoC, and SmartFusion 2 boards.
5. Repeatedly compiled the design to ascertain the maximum number of counters that could be generated before compilation failed.
6. Assigned pins for each port of the top-level entity by referencing the design manuals for the DE10 Lite, DE1 SoC, and SmartFusion 2 boards.
7. Determined the number of counters that could be generated while still meeting timing closure, documented the Fmax in the timing report for each board, and estimated FPGA logic utilization.
8. This module provided valuable insights into the design and implementation of counters in Verilog, as well as the capacity limitations of various FPGA development boards concerning the accommodation of multiple counters.

3.4. Test Results

DE10 Lite:
Counters able to be included in design: 2100 counters
Logic Utilization: 46203/49760, 93% Logic Utilization
Fmax: 83.49 MHz
DE1 SOC:
Counters able to be included in design: 2673 counters
Logic Utilization: 31986/32070, 100% Logic Utilization
Fmax: 155.47 MHz
Smart Fusion 2:
Counters able to be included in design: 391 counters
Logic Utilization: 8997/12084, 74.45% Logic Utilization
Fmax: 353.607 MHz
See Appendix for Screenshots and figures.

3.5. Lessons Learned

Through this experience, I gained valuable insights into the diverse capacities and logic requirements of different FPGA boards. It became evident that various boards have significantly varying sizes in terms of logic resources. For instance, while the SmartFusion 2 board could accommodate approximately 391 instances of the counter, the MAX 10 exhibited a remarkable capacity of over 2000, with a maximum observed instance count of 2100. The DE1 board, even more impressively, could host 2673 counter instances. Notably, the DE10 Lite board encountered a unique limitation, as its logic utilization reached 93% (46,204/49,760), and yet it couldn't accommodate additional units due to the exhaustion of Logic Array Blocks (LABs), which reached a utilization of 99.9% (3108/3110). This observation underscored

the importance of considering multiple factors beyond mere logic availability when selecting an FPGA board for a particular application. This module utilized a binary search method to establish the maximum number of counters each module could support. The search involved iterative increments of counter numbers until compilation failure occurred, ensuring precise identification of the maximum counters for each module. Parameterization is employed to enhance design flexibility and facilitate code reuse, making the modules adaptable to different configurations. Rigorous testing is a crucial step to confirm that the design functions as intended and meets specified requirements, covering both functional and performance testing. Early consideration of timing constraints is vital in the design process to prevent any issues that might impact the functionality of the design, ensuring signals meet necessary setup and hold times.

3.6 Conclusion

In Module 3, I delved into the creation of a 16-bit counter module in Verilog, integrating it into an RTL simulation top-level design. The primary goal was to assess the maximum counter capacity achievable on FPGA boards—Altera MAX10 DE10-lite, DE1-SoC, and Microsemi SmartFusion2 Maker kit—utilizing the FOR...GENERATE statement. Results demonstrated that DE10 Lite and DE1 SOC boards outperformed SmartFusion 2 in counter capacity. The procedure encompassed module development, verification using ModelSim Altera, top-level entity creation, testing on different boards, and iterative compilation. Valuable insights revealed varying logic resources across boards, with MAX 10 accommodating over 2000 counters, DE1 hosting 2673, and DE10 Lite nearing limits due to Logic Array Block constraints. This experience emphasized the importance of comprehensive board evaluation to meet diverse design needs.

Extra Credit:

Developing embedded Python applications on the Jupyter notebook of the Xilinx PYNQ board is a powerful and convenient approach for creating and deploying applications for embedded systems and IoT projects.

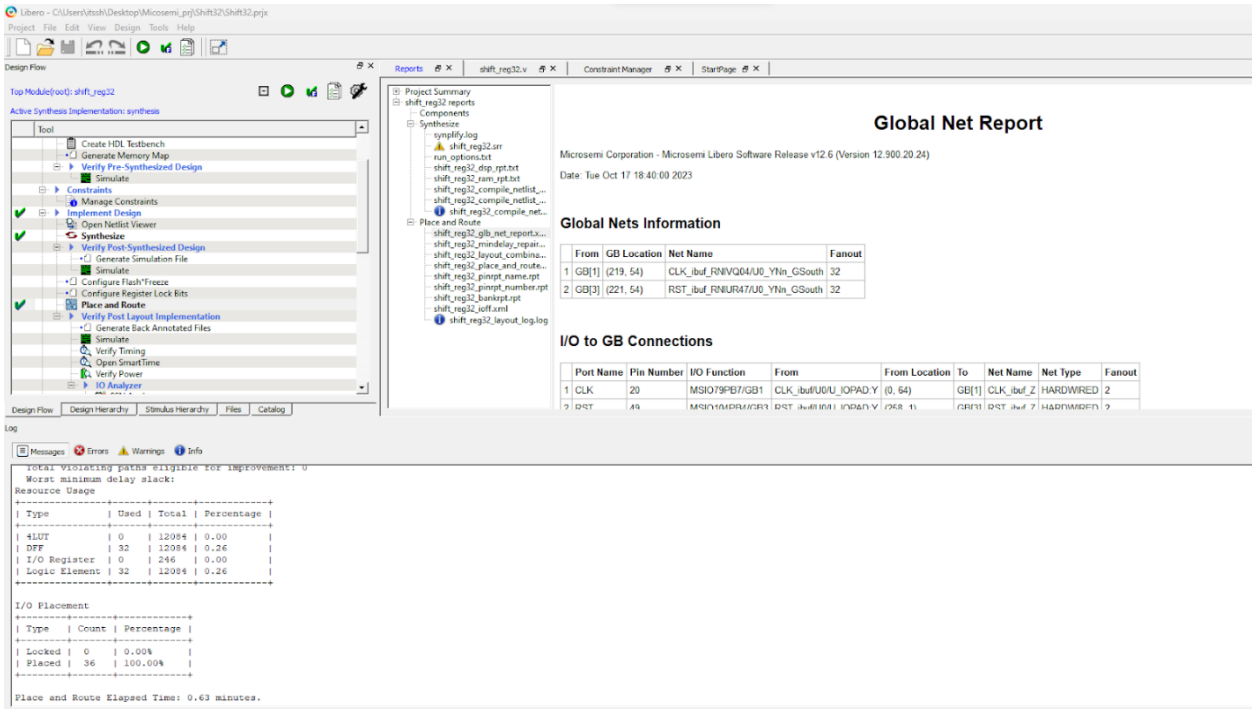
1. Connected the PYNQ board to a WiFi router and power it on, waiting for the green/yellow LED to stabilize.
2. Find the PYNQ board's IP address using an IP scanner tool.
3. Access the Jupyter notebook by entering the IP address in Google Chrome and using "xilinx" as the password.
4. Launch the Jupyter notebook interface on the PYNQ board.
5. Create a new Python notebook and import the necessary PYNQ Python packages and modules.
6. Write and run Python code cells to execute the program and observe the output on the board or terminal.
7. Debug the code using available debugging tools.
8. Save the Python notebook for future reference and sharing.

4.4. Test Results

We successfully created the Morse code application using Python on the Jupyter notebook of the Xilinx PYNQ board. Please see the "code for Morse code" in the appendix and attached figure for output for a screenshot of the output. To just verify the functionality we have configured 2 buttons and according to two inputs, A and B are printed. In the future, it can be developed and implemented for all morse code Characters

APPENDIX:

MODULE 1:



Global Net Report

Microsemi Corporation - Microsemi Libero Software Release v12.6 (Version 12.900.20.24)

Date: Tue Oct 17 18:40:00 2023

Global Nets Information

	From	GB Location	Net Name	Fanout
1	GB[1]	(219, 54)	CLK_ibuf_RNIVQ04/U0_YNn_GSouth	32
2	GB[3]	(221, 54)	RST_ibuf_RNIUR47/U0_YNn_GSouth	32

I/O to GB Connections

	Port Name	Pin Number	I/O Function	From	From Location	To	Net Name	Net Type	Fanout
1	CLK	20	MSIO79PB7/GB1	CLK_ibuf/U0/U_IOPAD:Y	(0, 64)	GB[1]	CLK_ibuf_Z	HARDWIRED	2
2	RST	49	MSIO104PB4/GB3	RST_ibuf/U0/U_IOPAD:Y	(268, 1)	GB[3]	RST_ibuf_Z	HARDWIRED	2

Total violating paths eligible for improvement: 0
Worst minimum delay slack:
Resource Usage

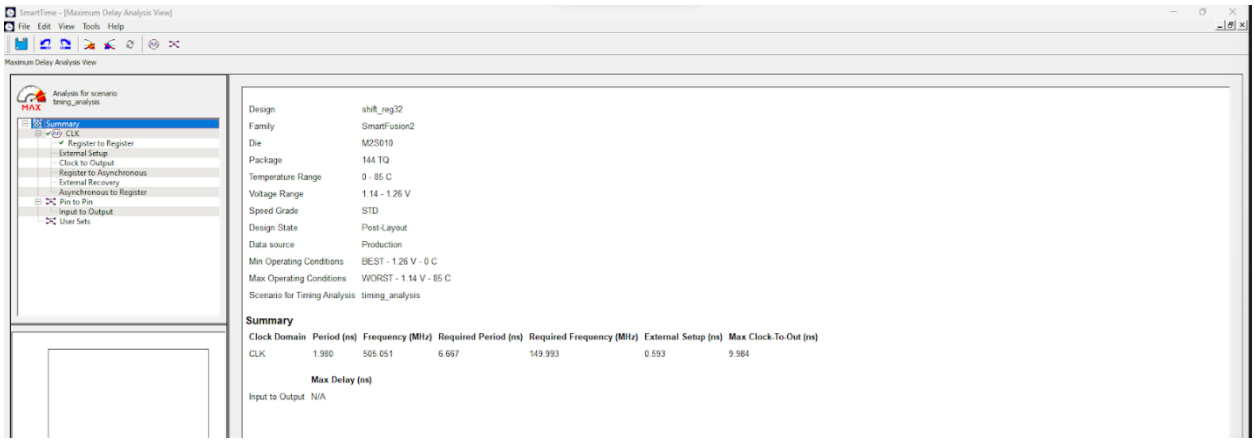
Type	Used	Total	Percentage
4LUT	0	12084	0.00
DFF	32	12084	0.26
I/O Register	0	246	0.00
Logic Element	32	12084	0.26

I/O Placement

Type	Count	Percentage
Locked	0	0.00%
Placed	36	100.00%

Place and Route Elapsed Time: 0.63 minutes.

Fig 1.1 Successful compilation and report



Design	shift_reg32
Family	SmartFusion2
Die	M2S010
Package	144 TQ
Temperature Range	0 - 85 C
Voltage Range	1.14 - 1.26 V
Speed Grade	STD
Design State	Post-Layout
Data source	Production
Min Operating Conditions	BEST - 1.26 V - 0 C
Max Operating Conditions	WORST - 1.14 V - 85 C
Scenario for Timing Analysis	timing_analysis

Summary

Clock Domain	Period (ns)	Frequency (MHz)	Required Period (ns)	Required Frequency (MHz)	External Setup (ns)	Max Clock-To-Out (ns)
CLK	1.980	505.051	6.667	149.993	0.593	9.984

Max Delay (ns)

Input to Output N/A

Fig 1.2 Fmax at 505 Mhz (Max)

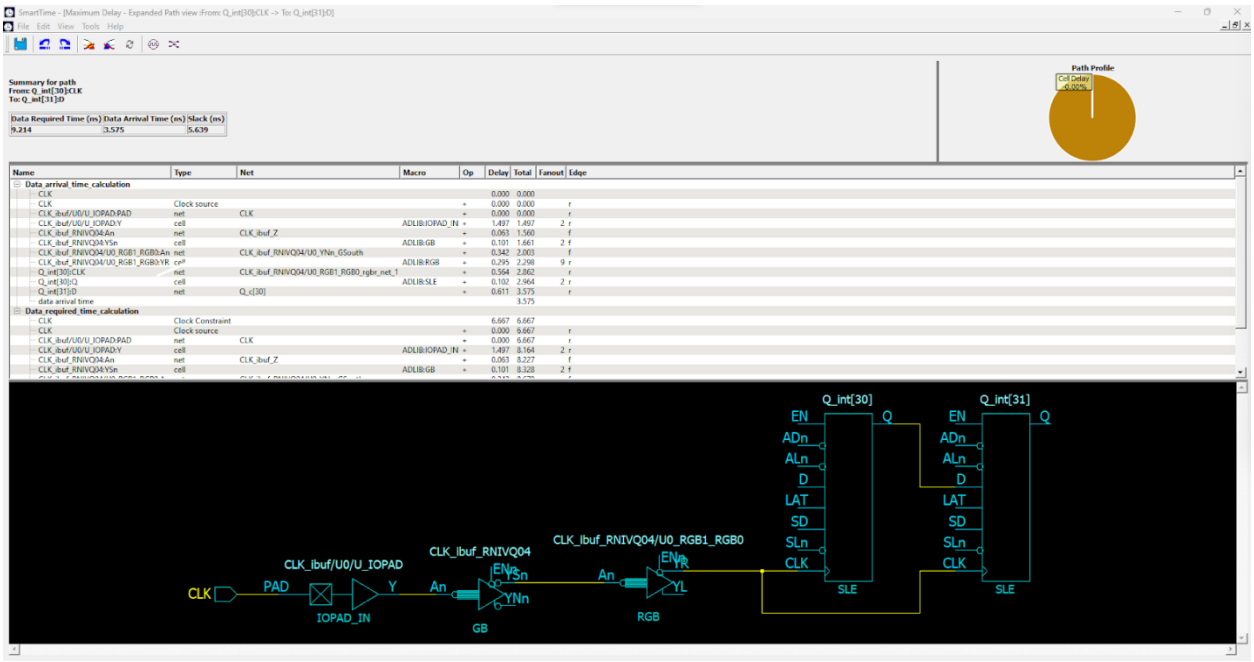


Fig 1.3 Expanded Path View (Max)

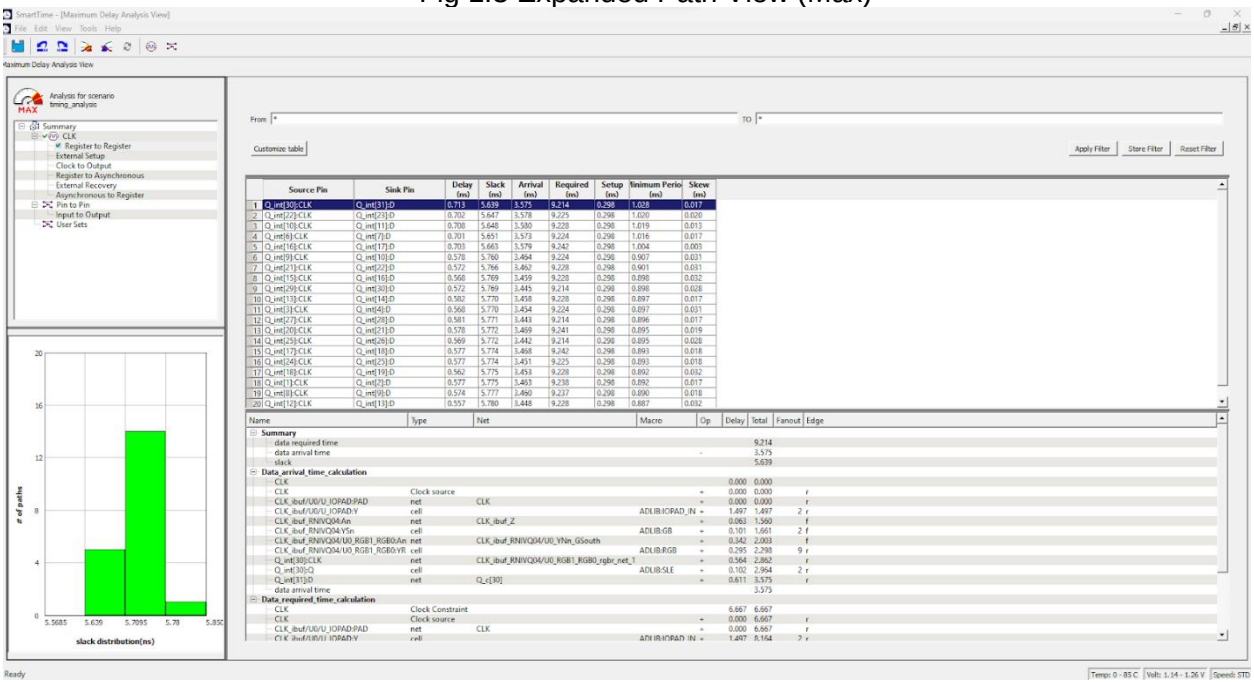


Fig 1.4 Register - Register Display (Max)

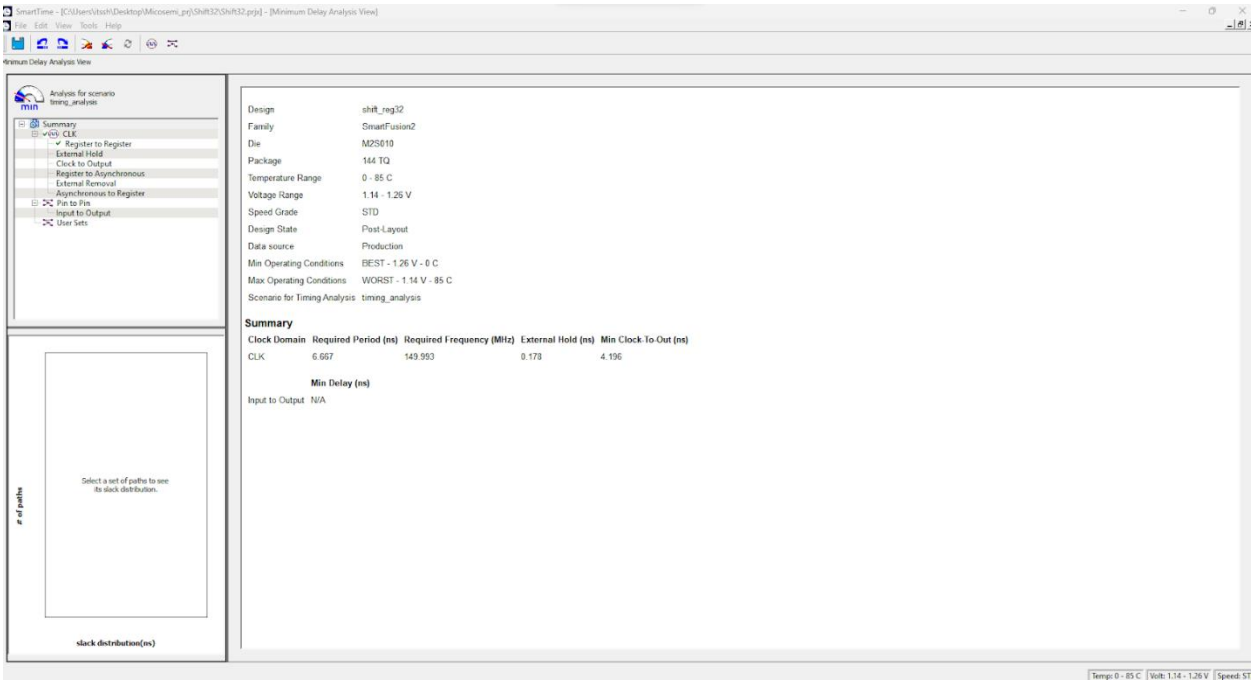


Fig 1.5 Fmax is 505 Mhz (Min)

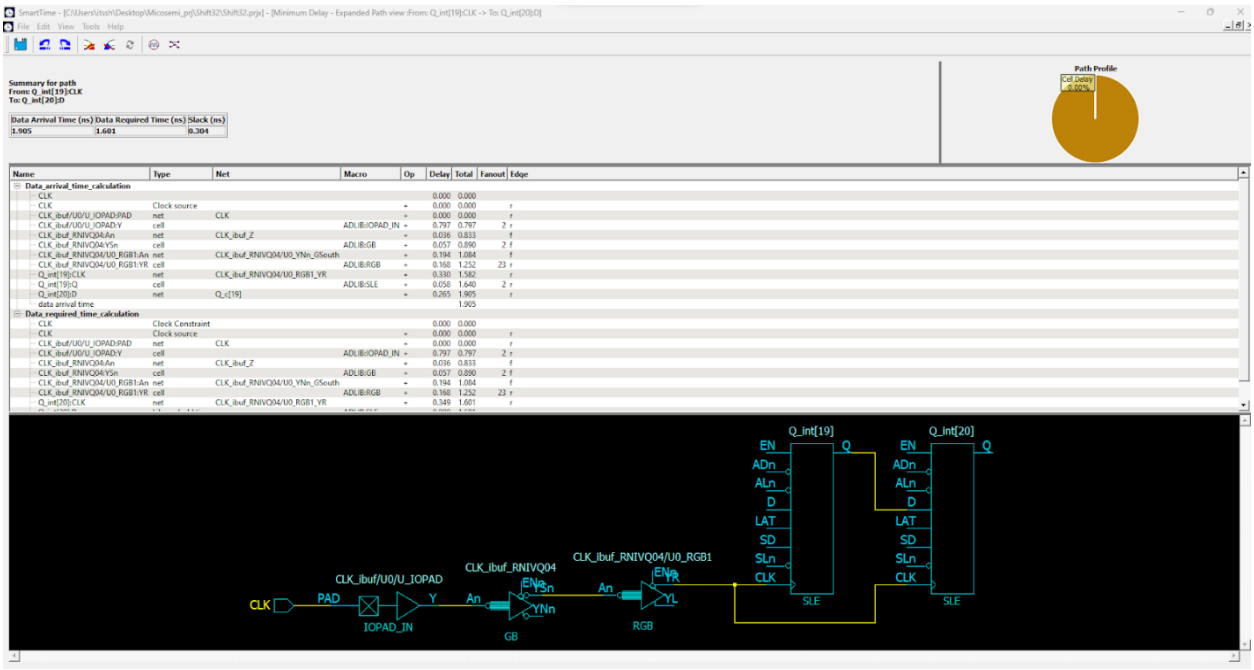


Fig 1.6 Expanded Path View (Min)

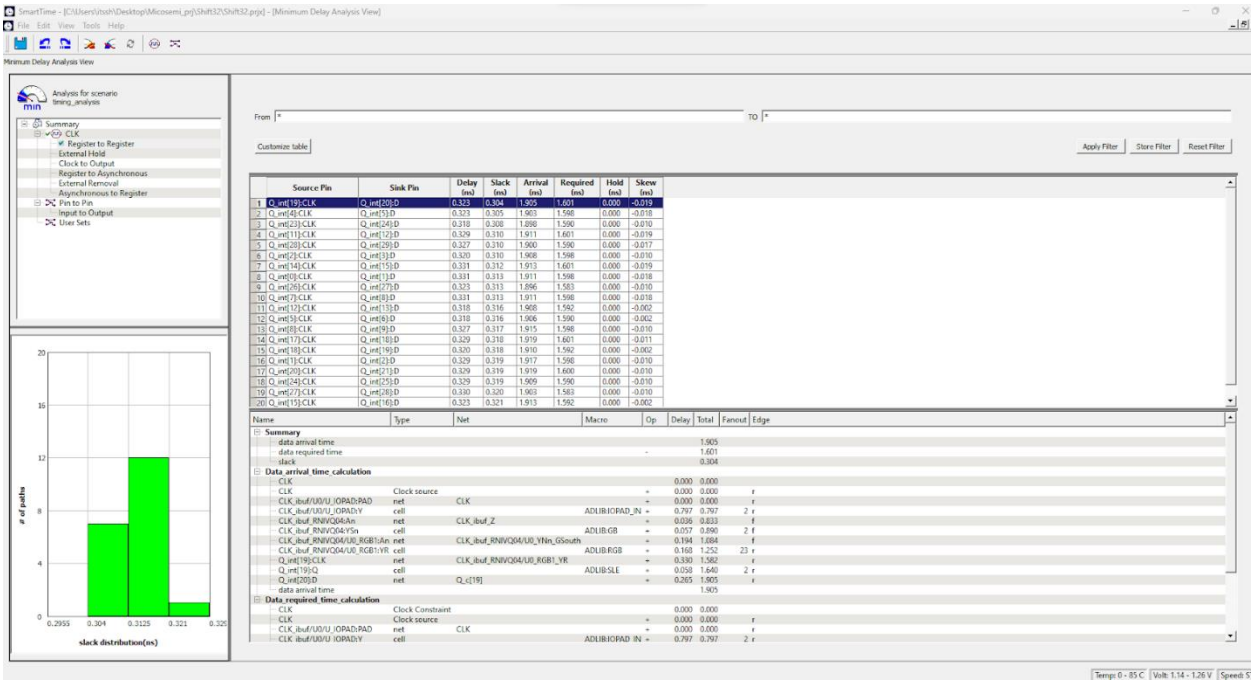


Fig 1.7 Register - Register display (Min)

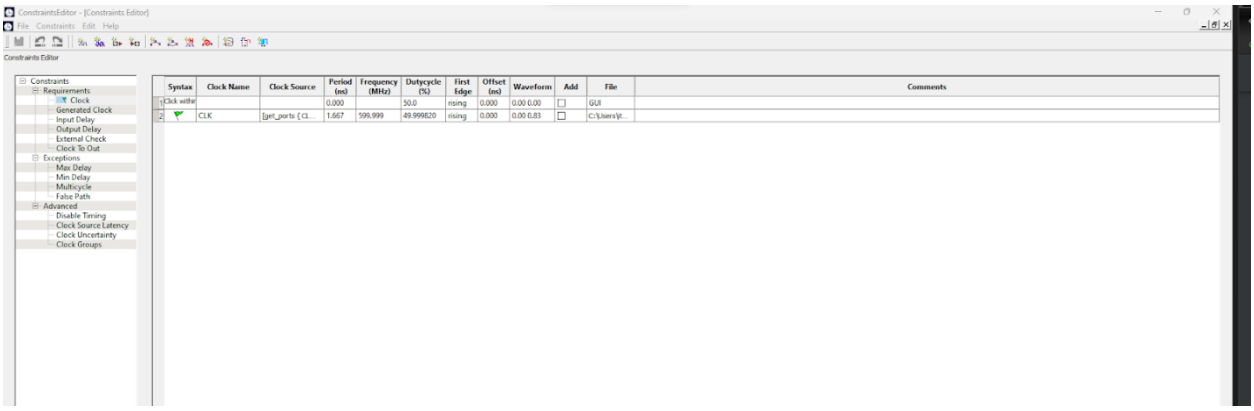


Fig 1.8 No clock constraint

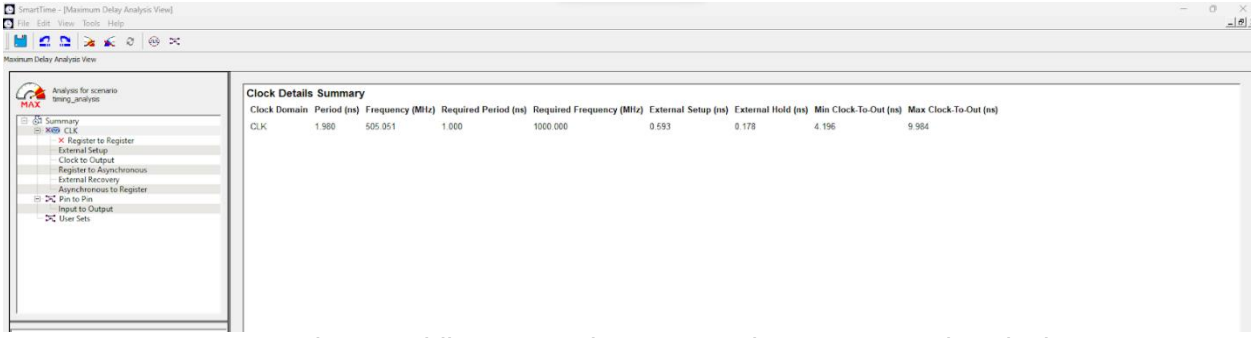


Fig 1.9 Adding constraint at 1000Mhz to get negative slack

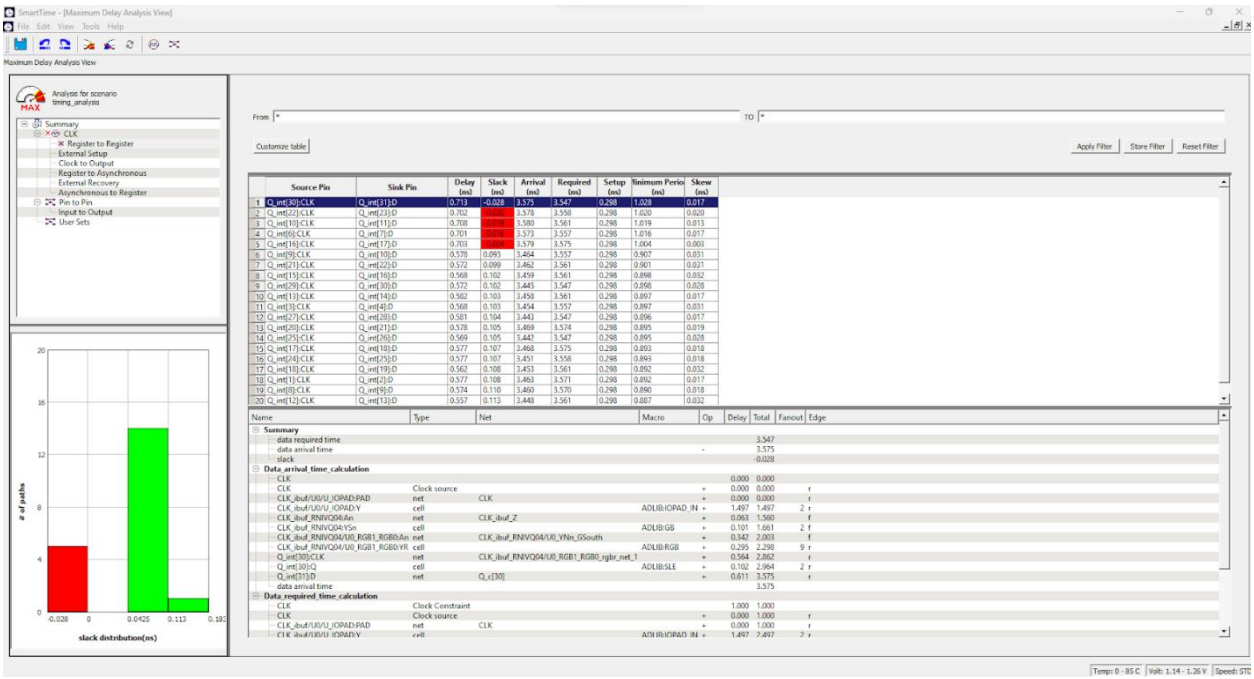


Fig 1.10 Negative slack observation

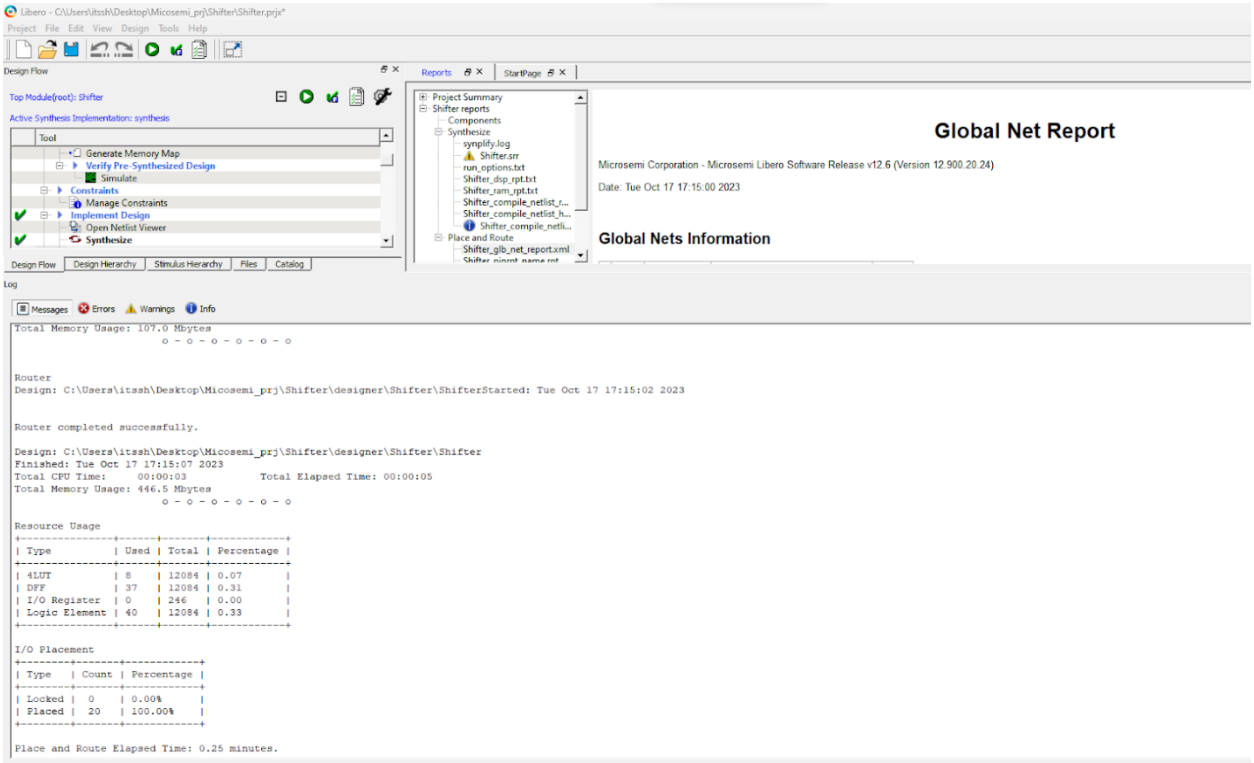


Fig 1.11 Successful compilation and report

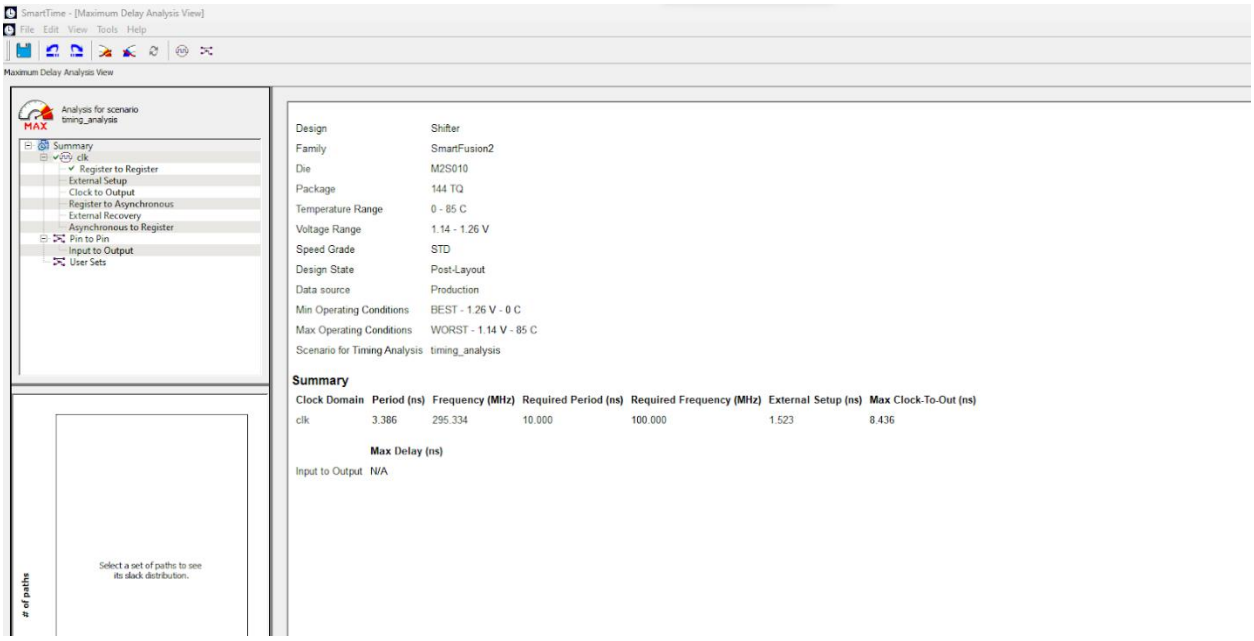


Fig 1.12 Fmax at 295 Mhz

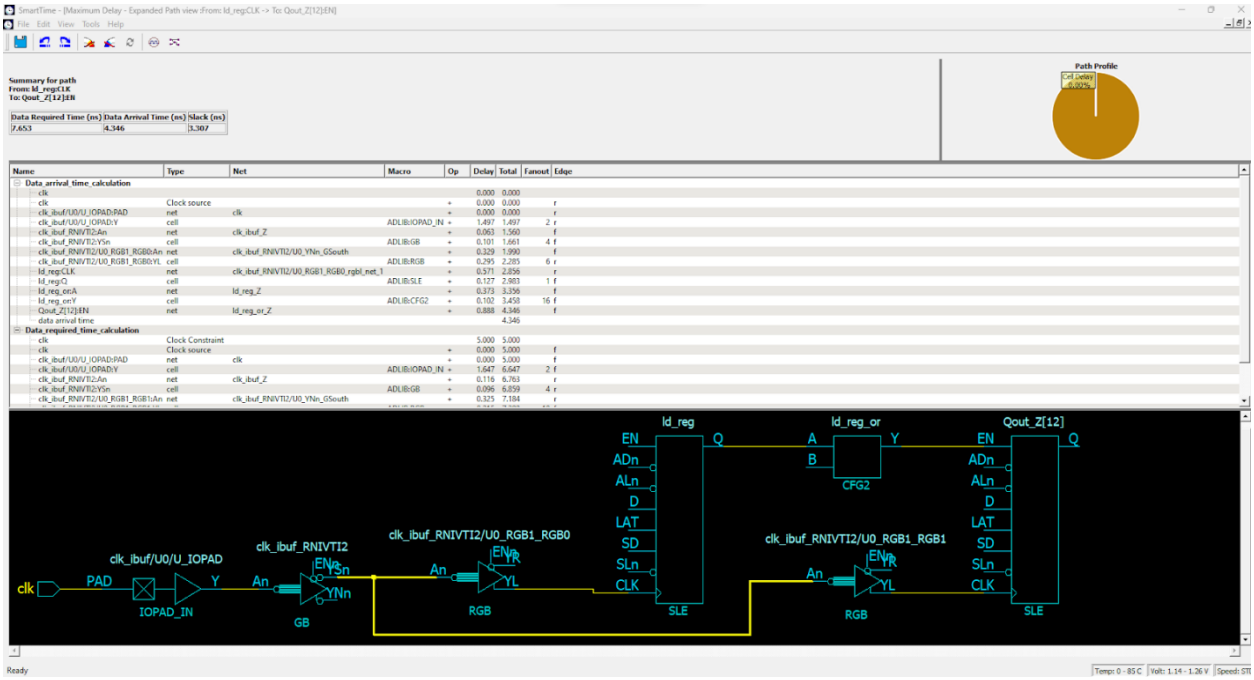


Fig 1.13 Expanded Path view

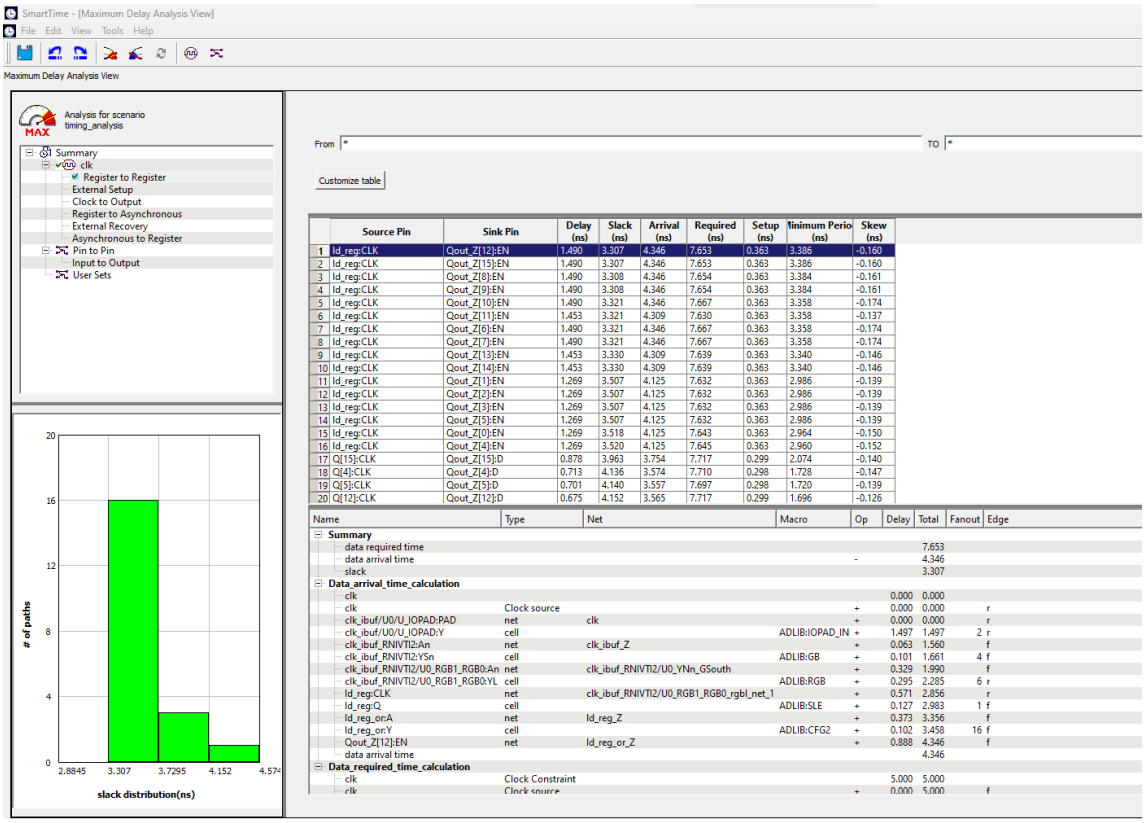


Fig 1.14 Register to Register Display

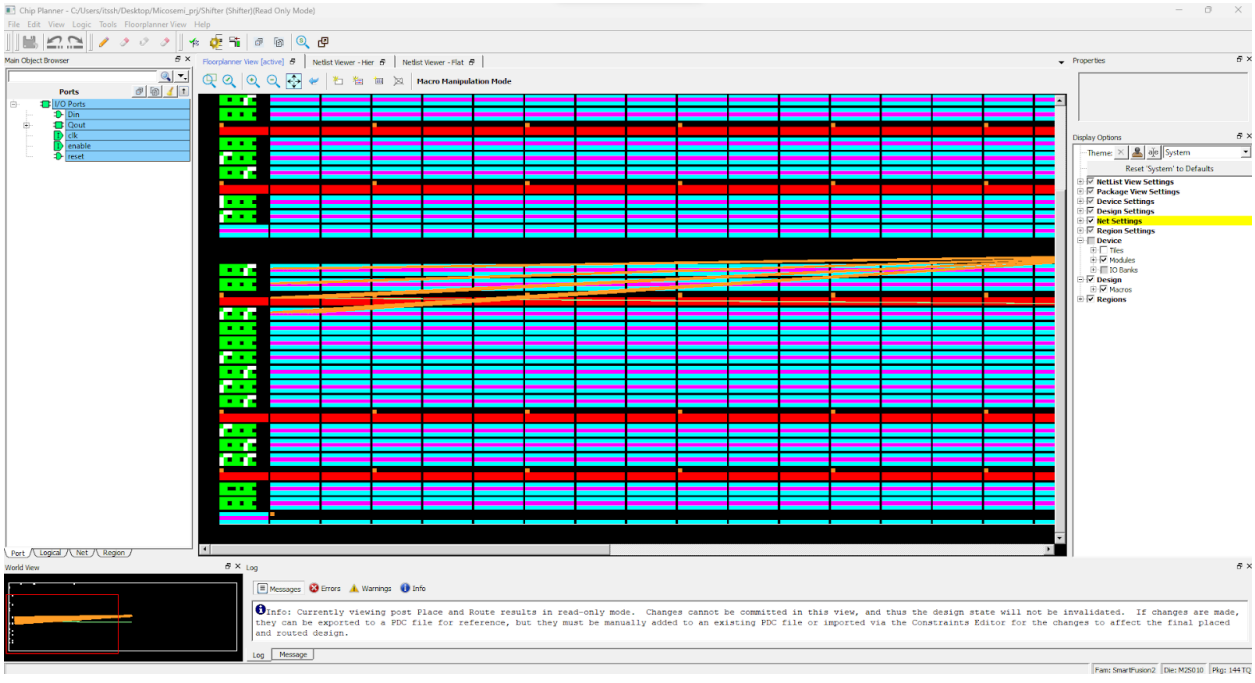


Fig 1.15 Chip Planner with netview

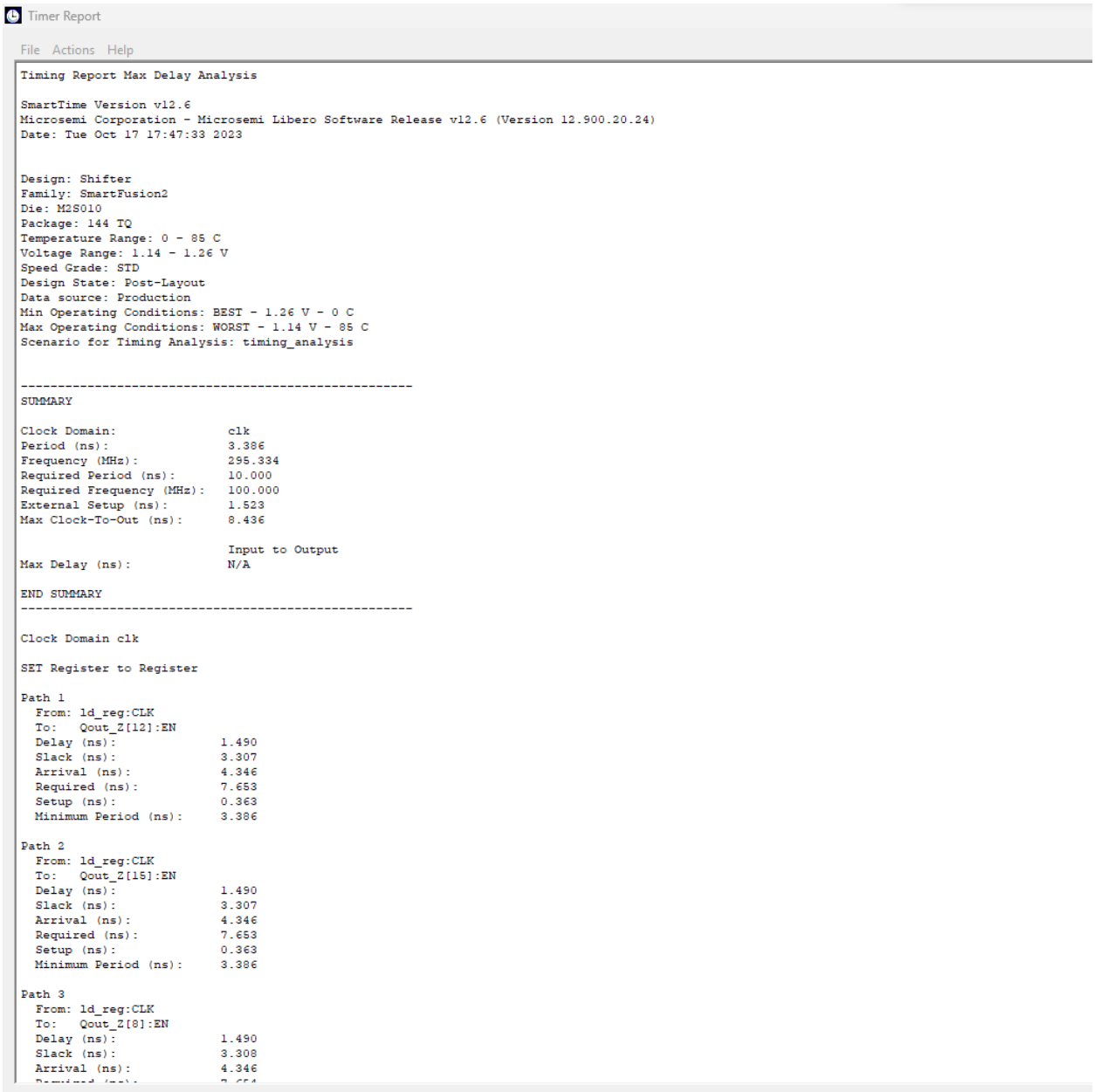


Fig 1.16 Timing Report

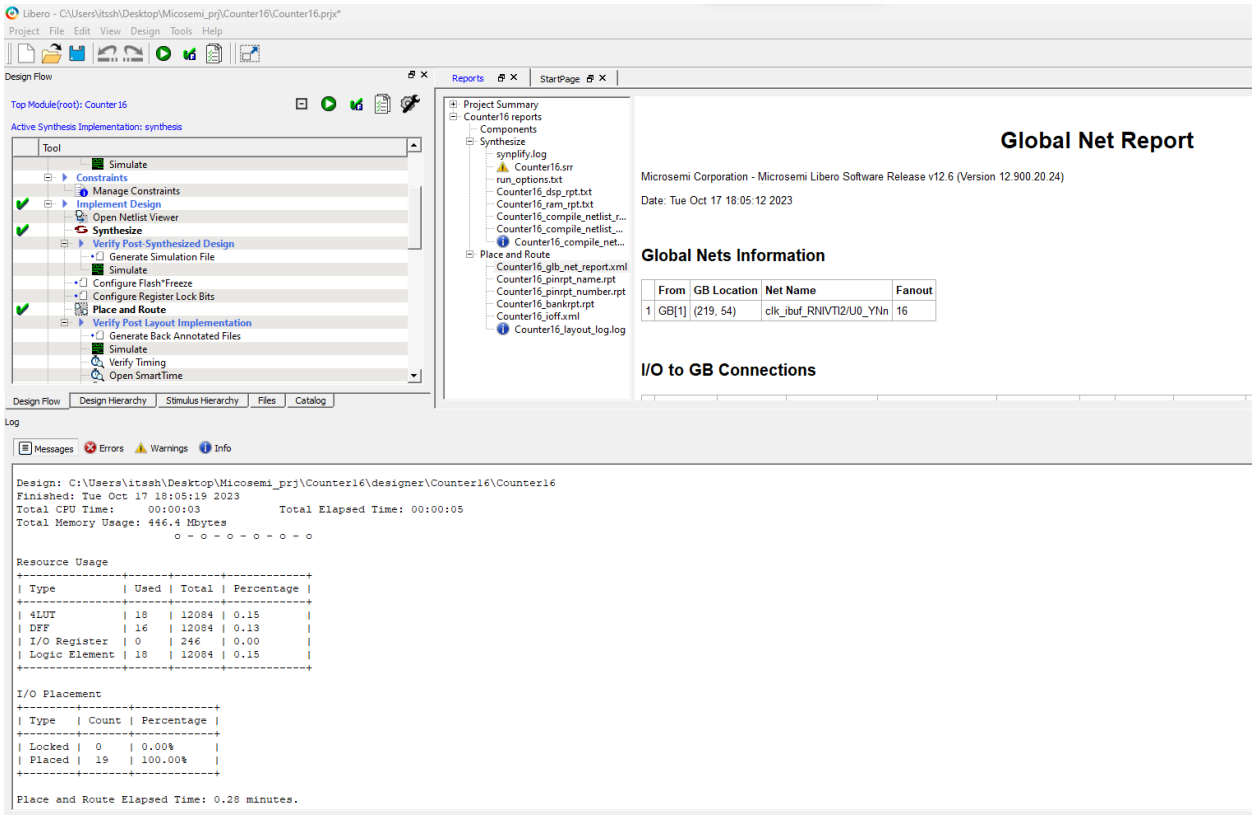


Fig 1.17 Successful compilation and report

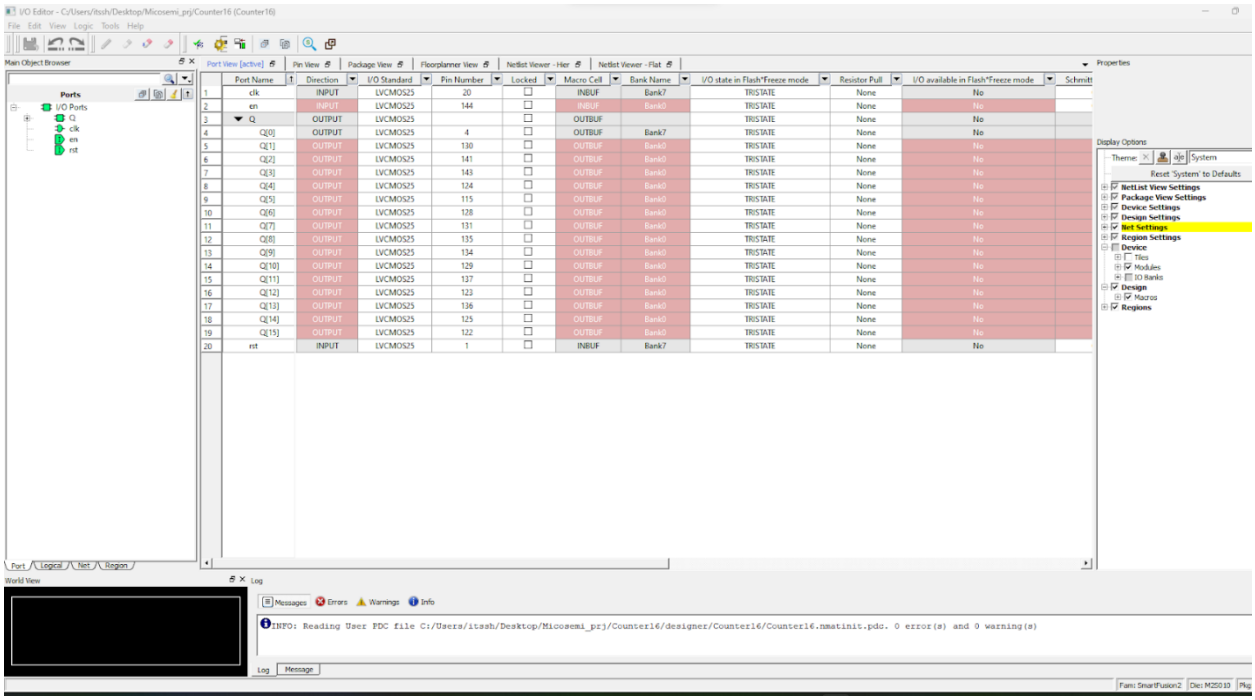


Fig 1.18 Pin assignment viewer

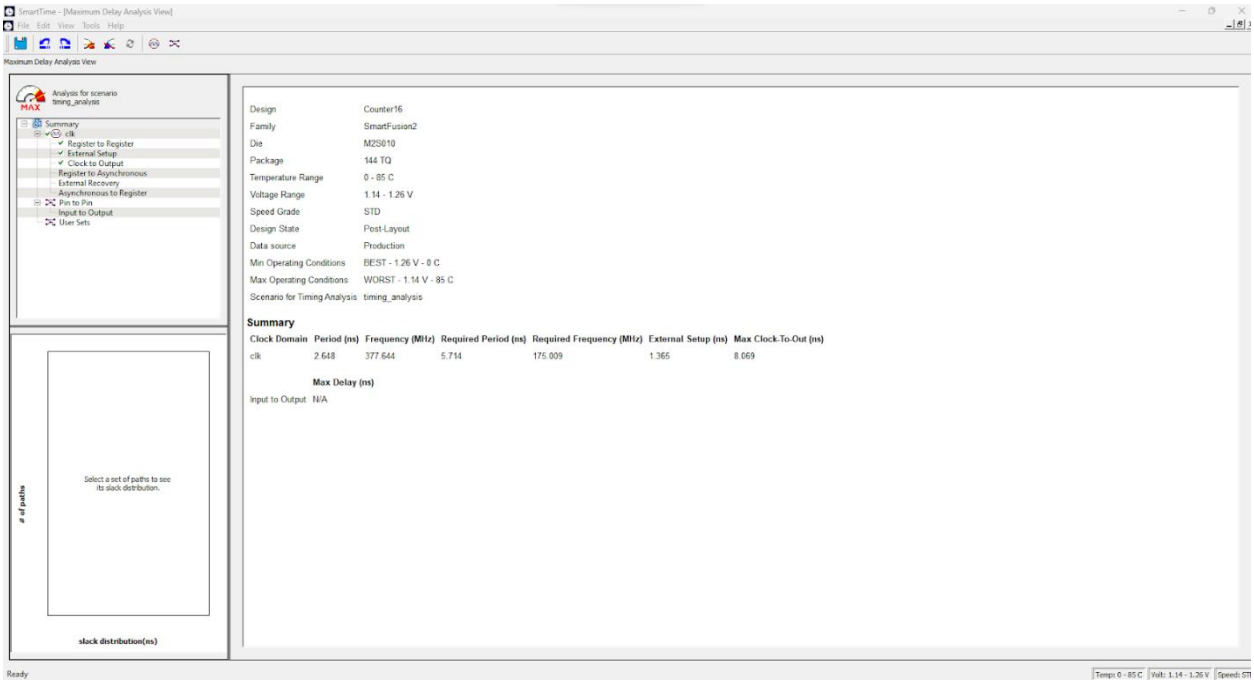


Fig 1.19 Fmax at 377Mhz

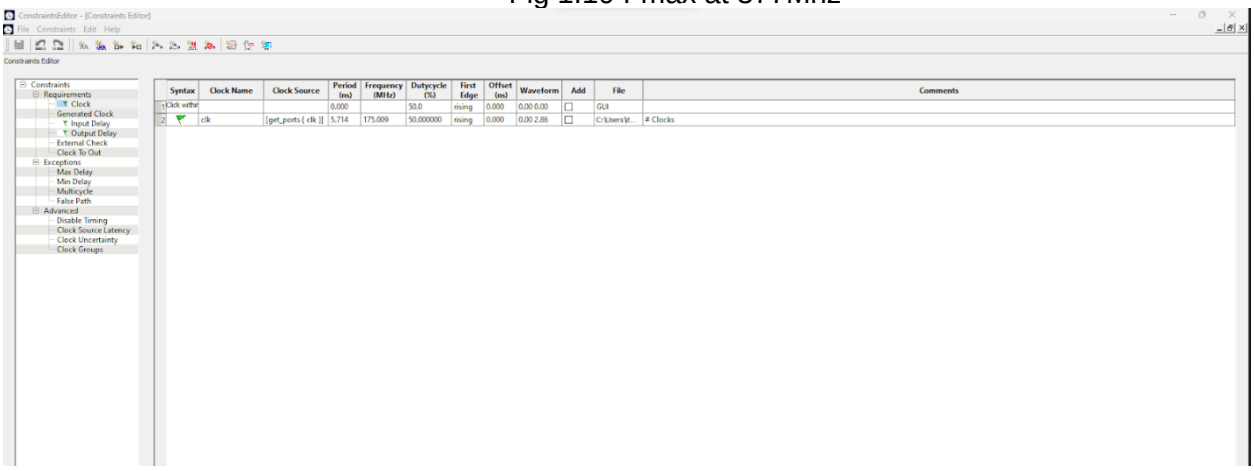


Fig 1.20 No clock constraint

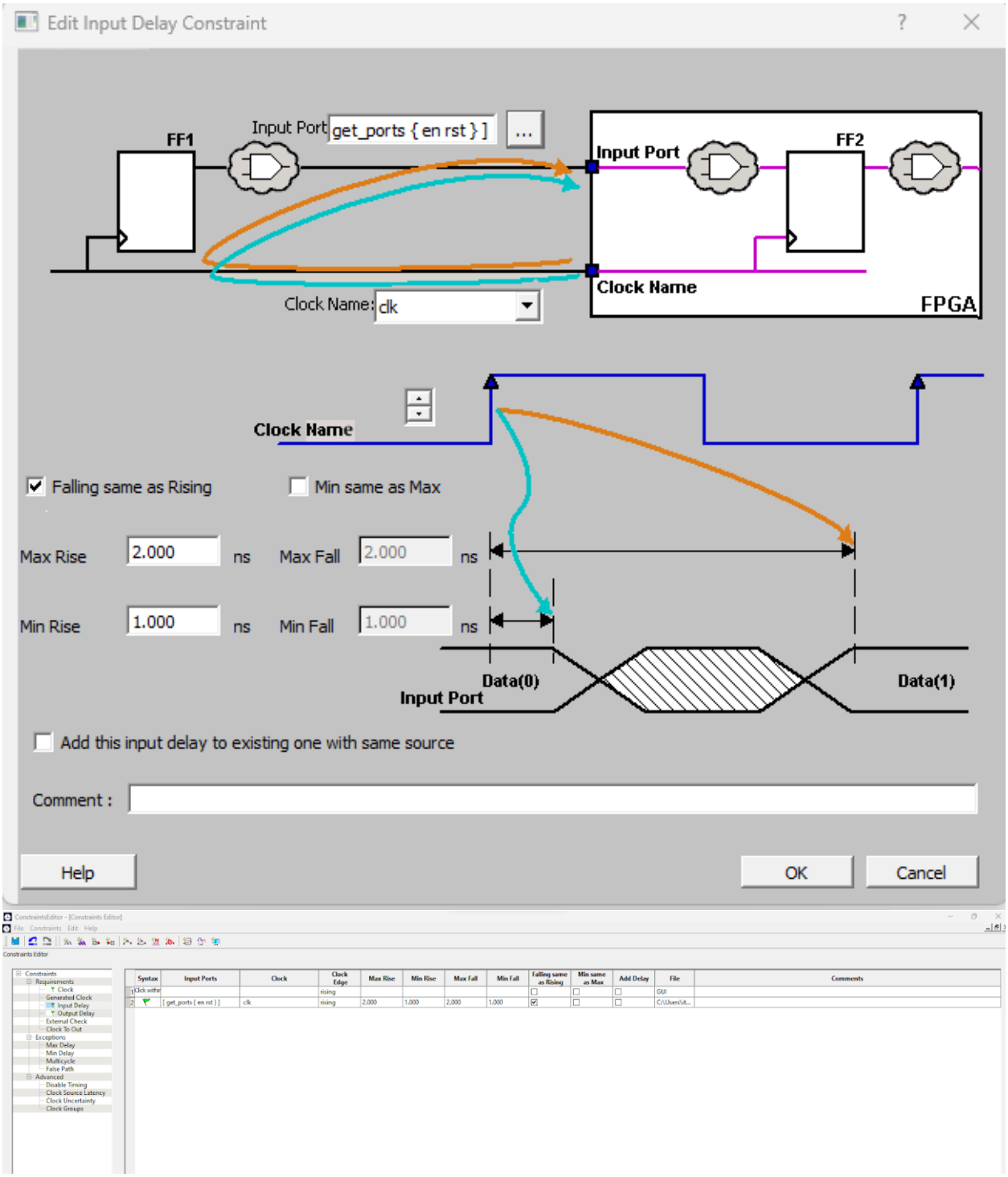


Fig 1.21 Adding constraint - Input Delay

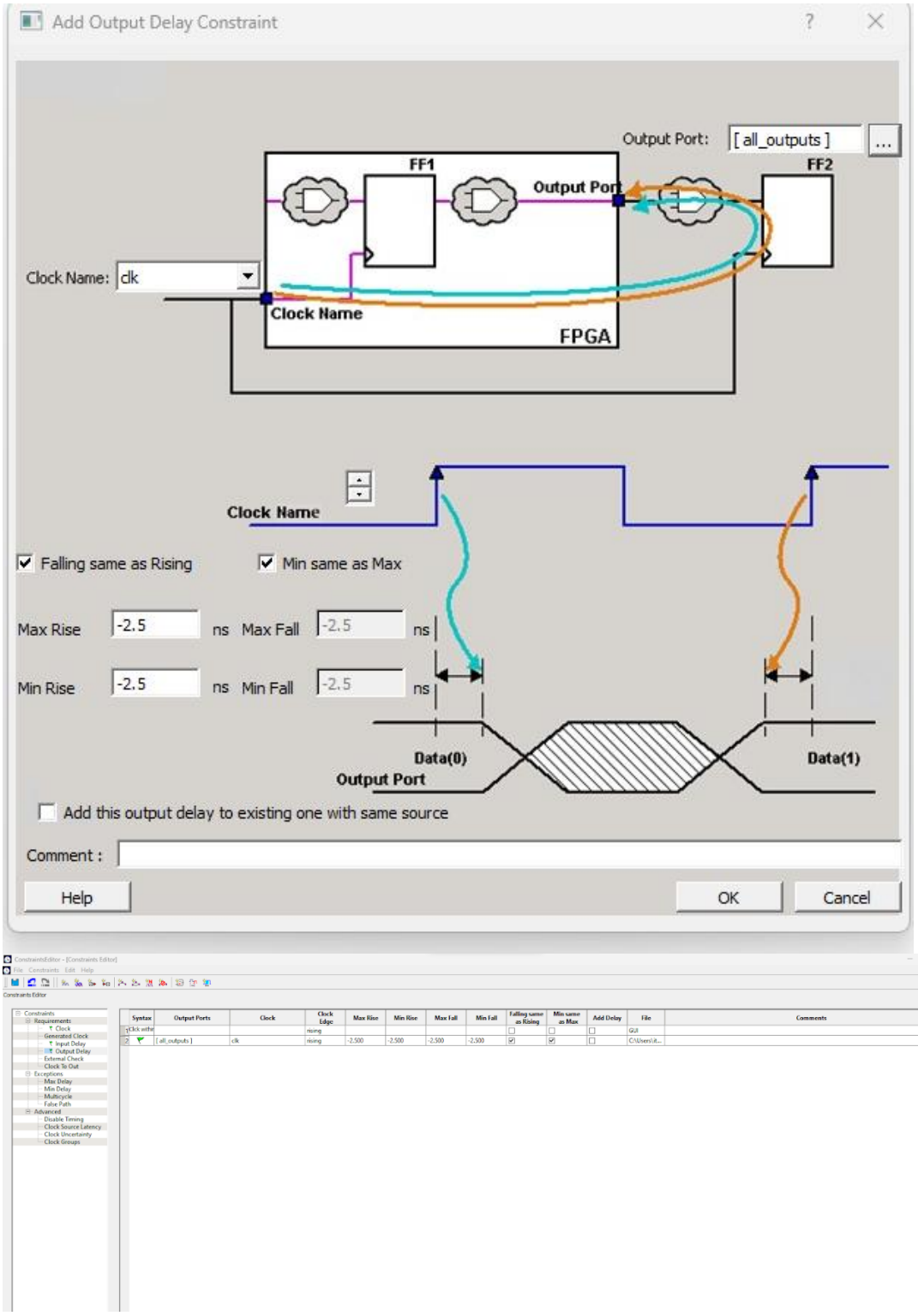


Fig 1.22 Adding constraint - Output Delay

Fig 1.23 Fmax after constraining

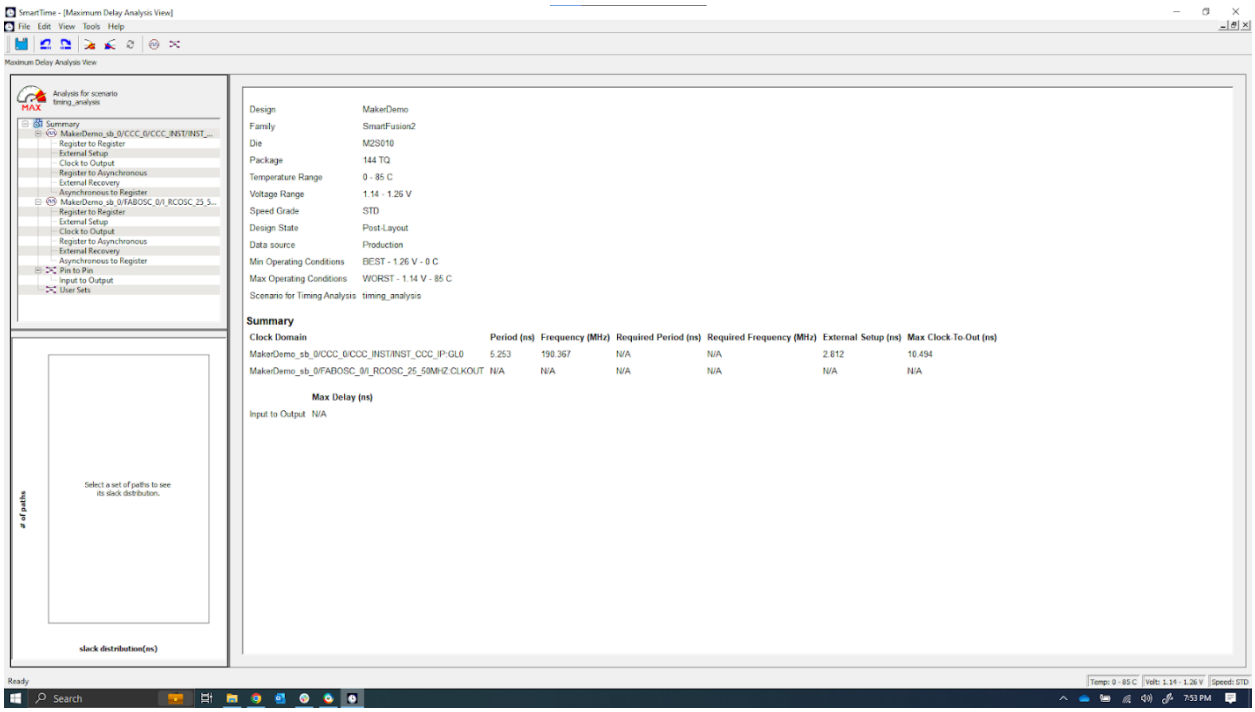
Resource Usage

Type	Used	Total	Percentage
4LUT	18	12084	0.15
DFF	16	12084	0.13
I/O Register	0	246	0.00
Logic Element	18	12084	0.15

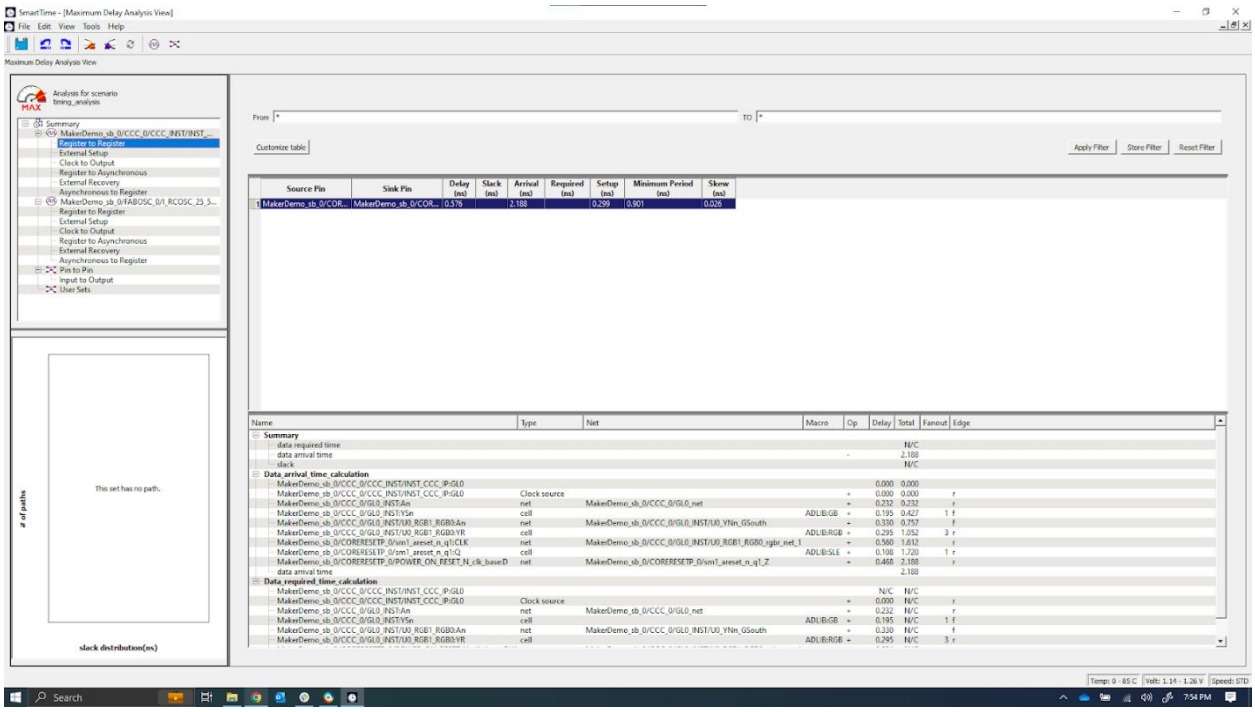
Fig 1.24 Resources after clock constraining

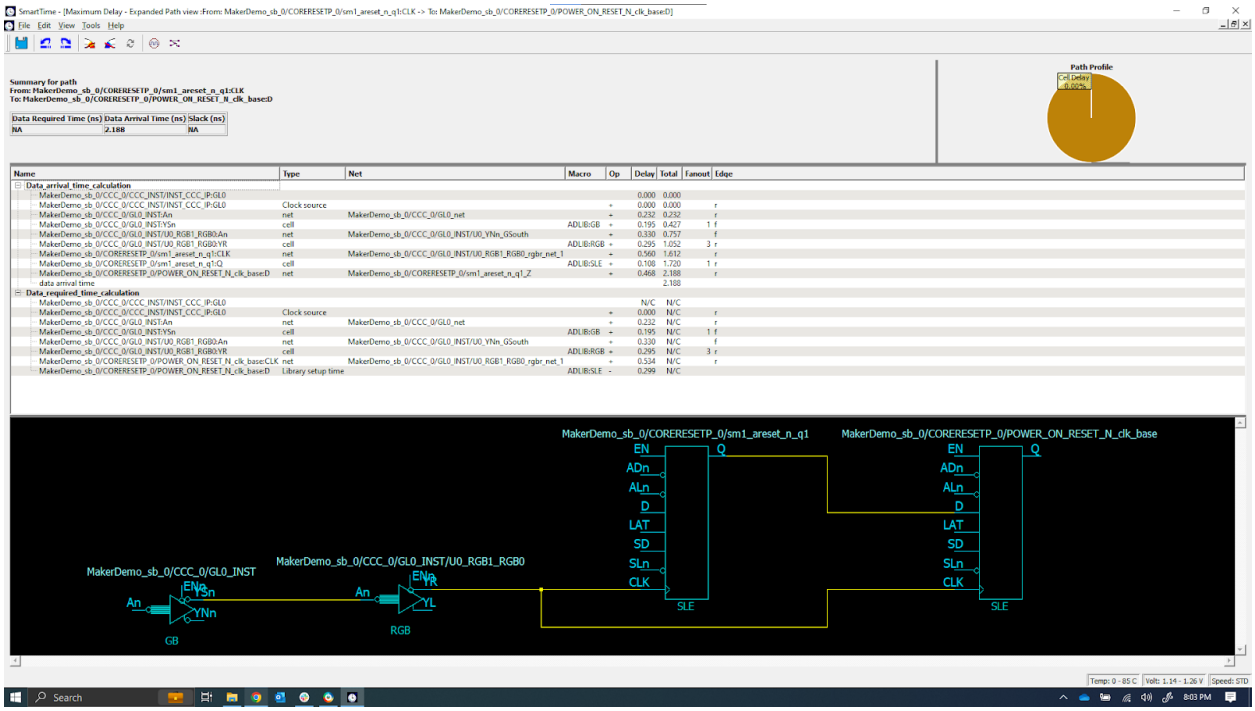
MOD 2:

Fmax

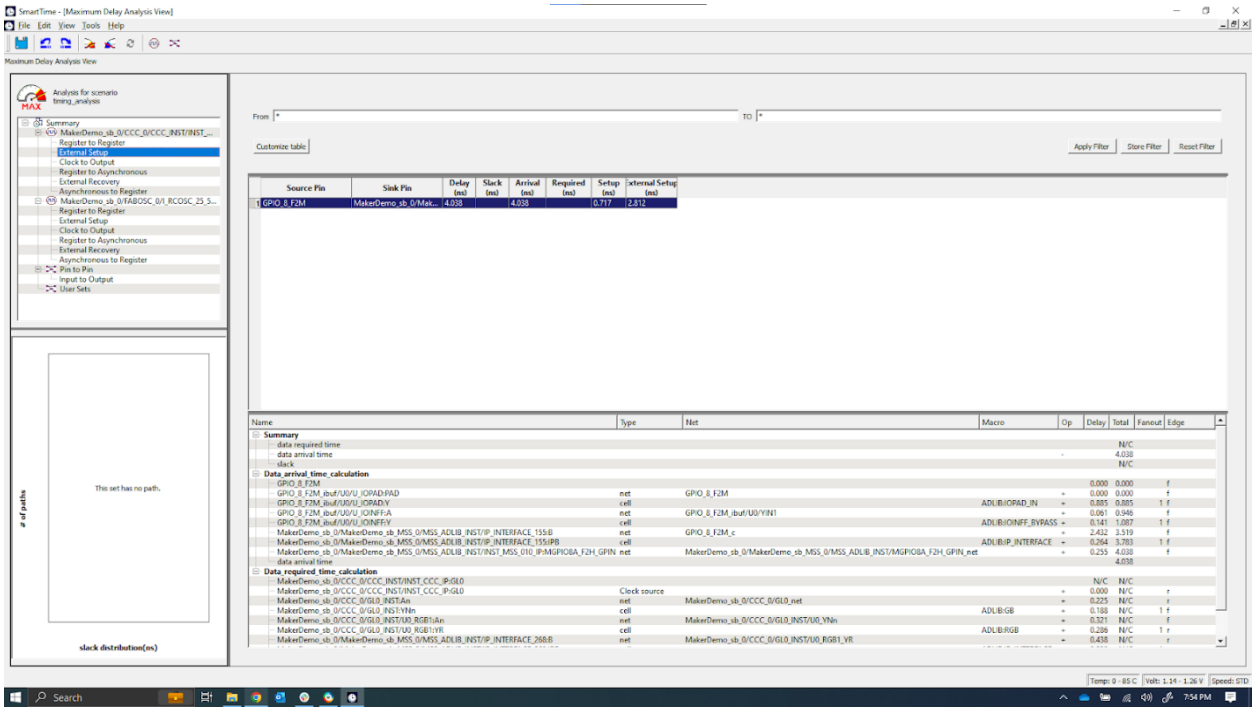


Register to Register Timing Details

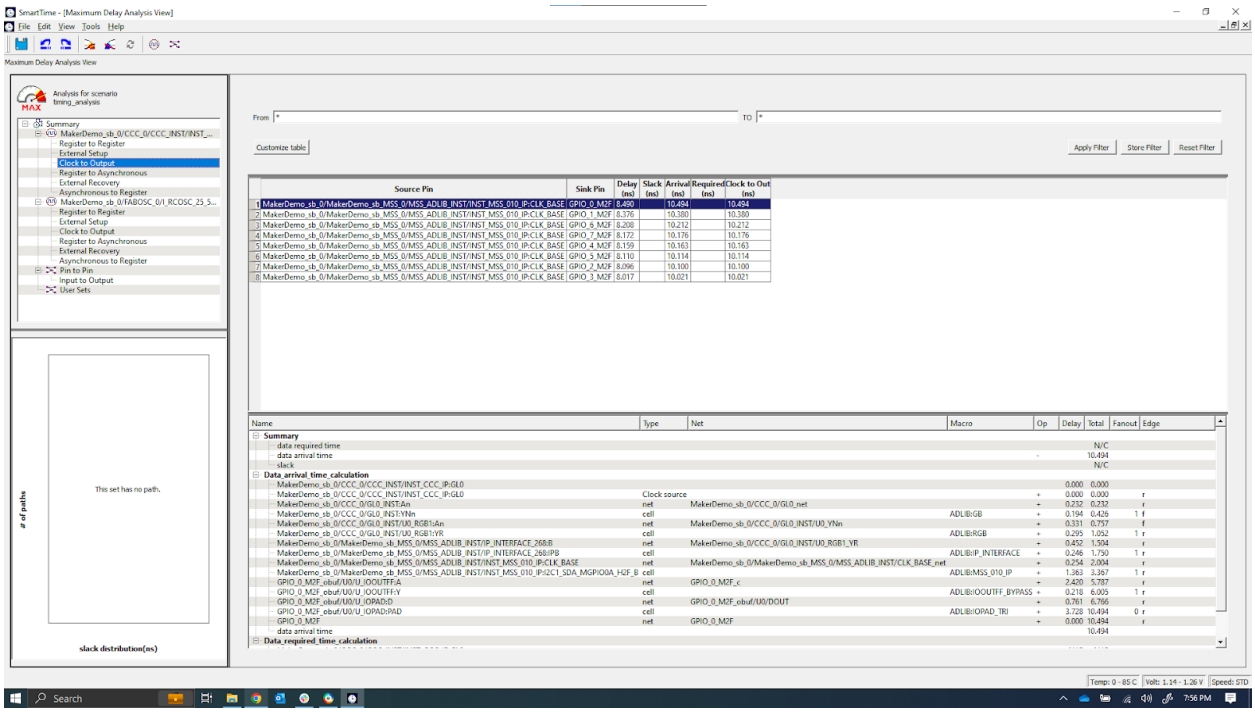




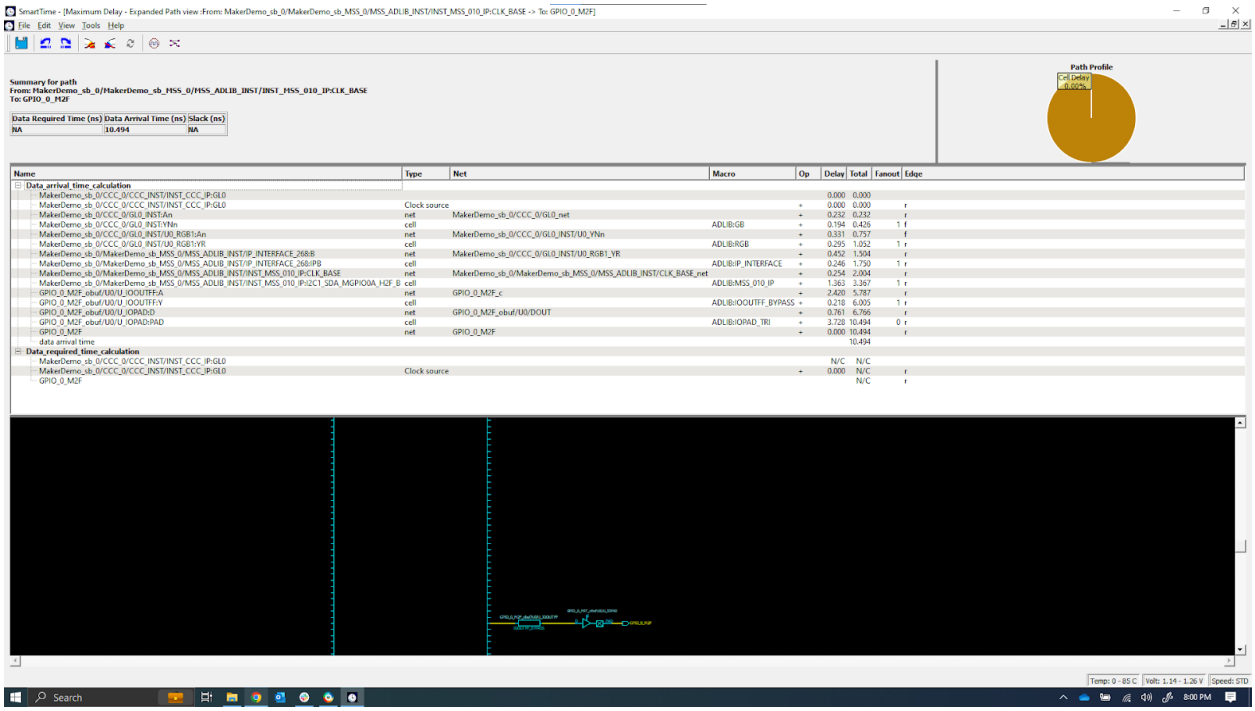
External Setup Timing Details



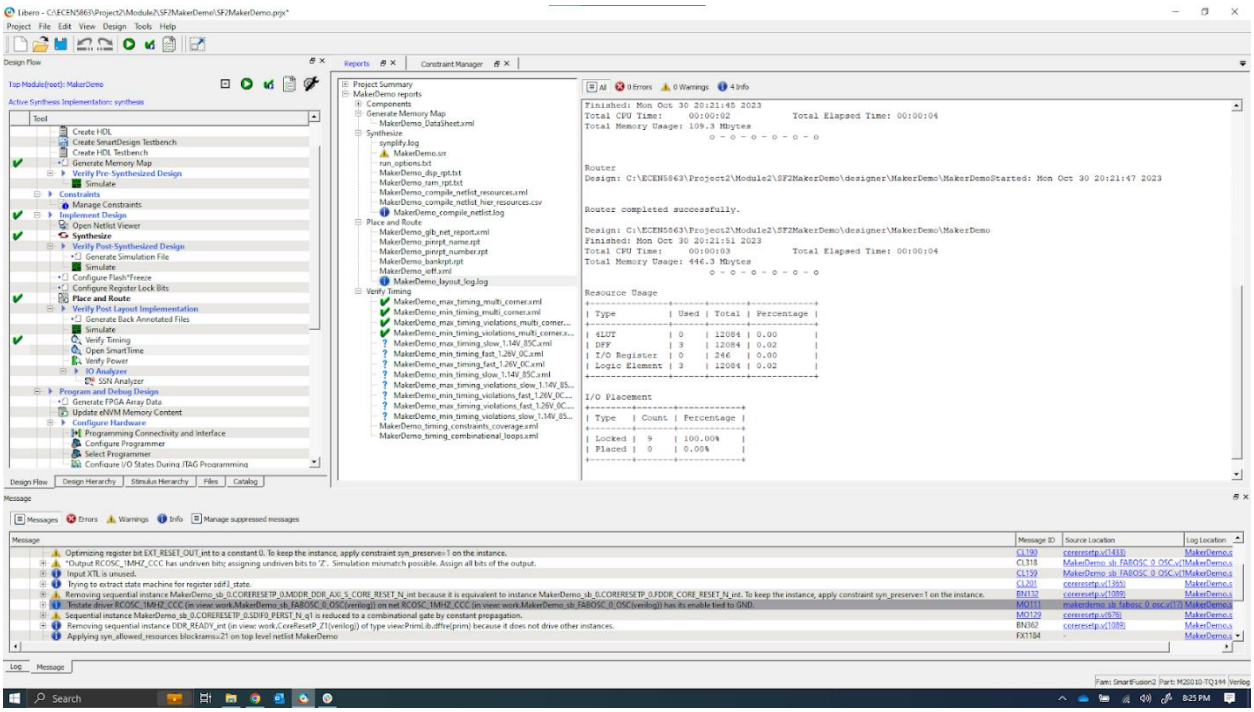
Clock to Output Timing Details



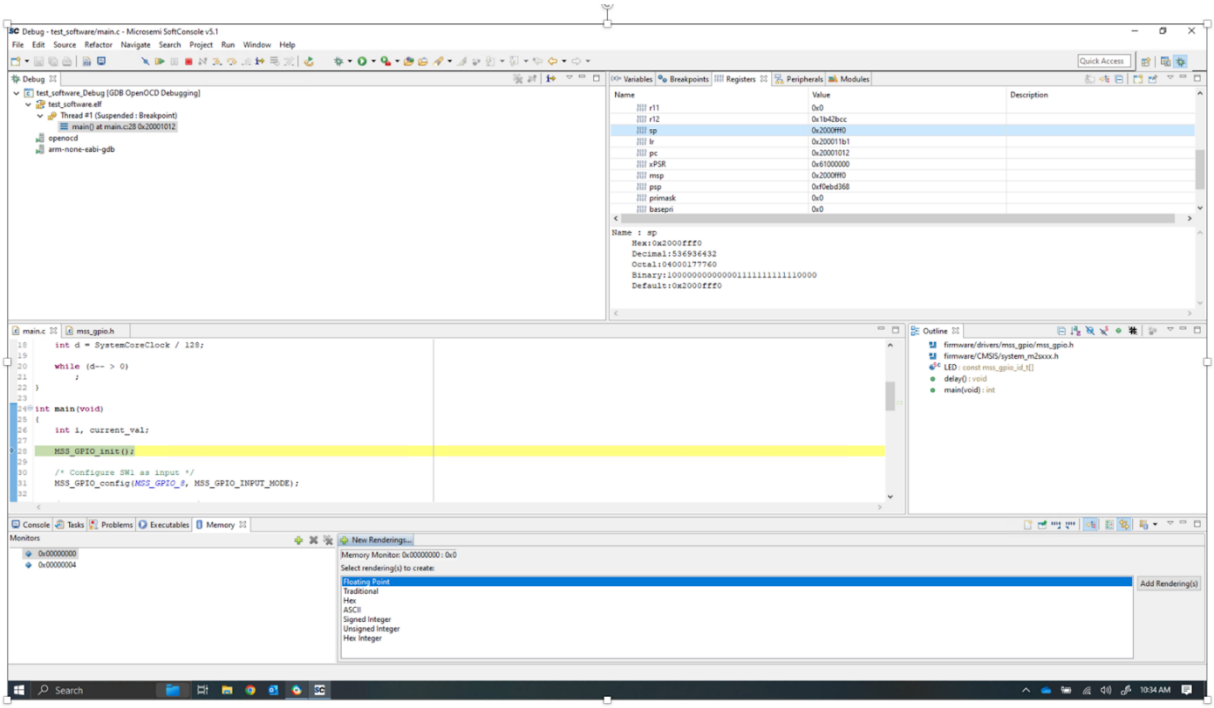
Clock to Output for GPIO_0_M2F



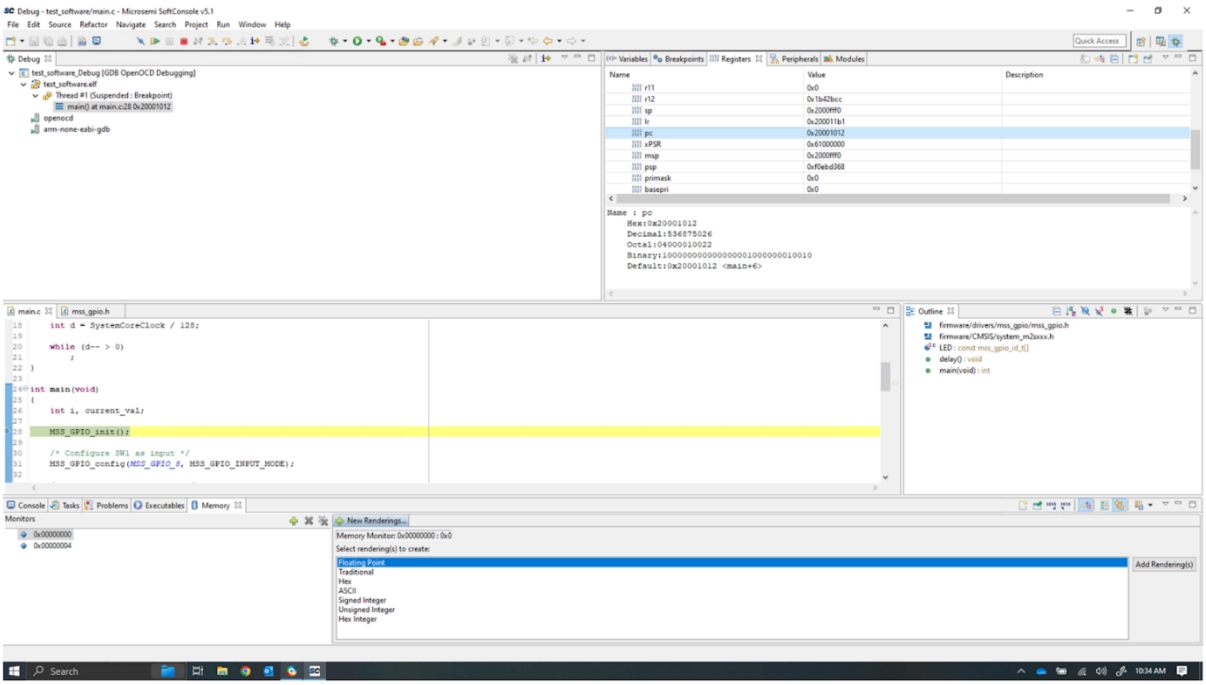
% utilization of the FPGA logic



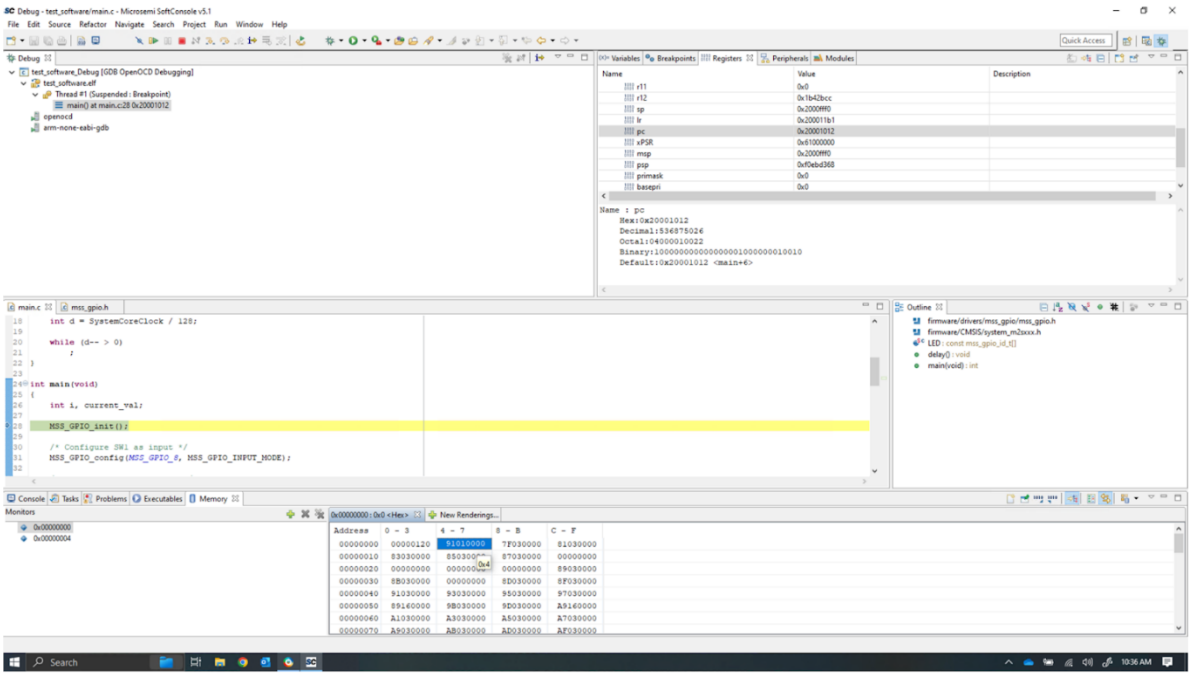
r13 (SP)

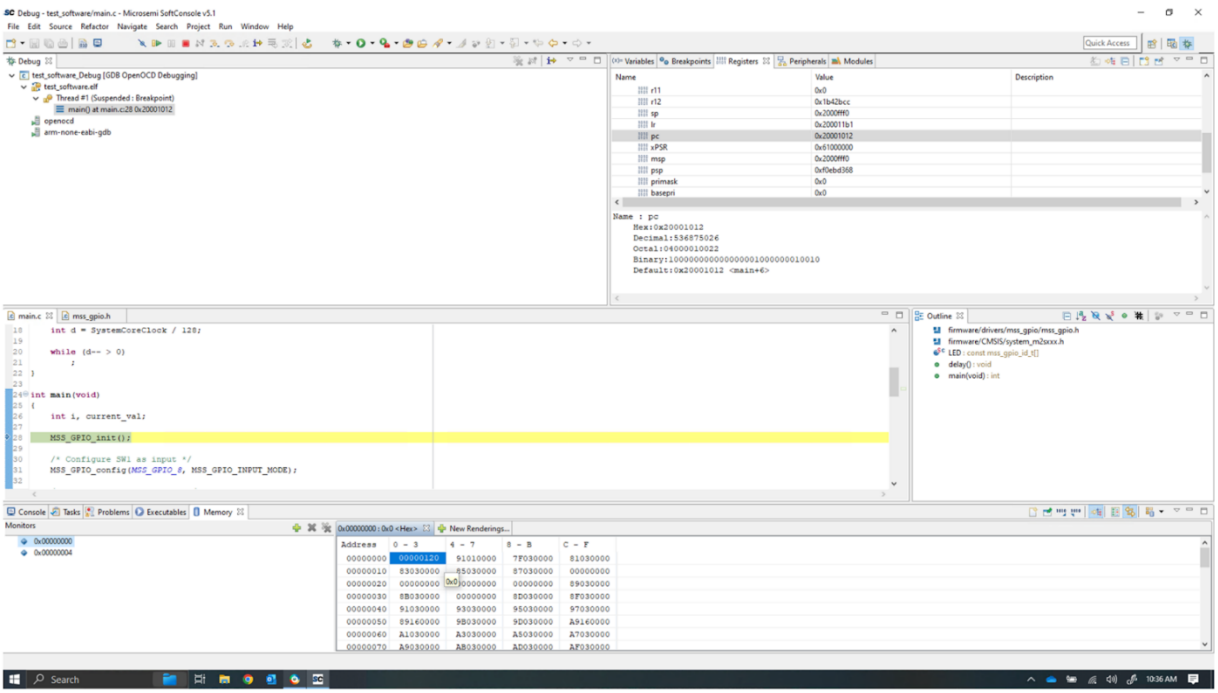


r15 (PC)



Memory Address (0x00000000 & 00000004)





Board Image After Flashing Image

Source code to map switch as input

```
/*
 * main.c
 *
 *  Created on: Oct 20, 2023
 *      Author: hgajapathy
 */

#include "firmware/drivers/mss_gpio/mss_gpio.h"
#include "firmware/CMSIS/system_m2sxxx.h"

static const mss_gpio_id_t LED[8] = {
                MSS_GPIO_0, MSS_GPIO_1, MSS_GPIO_2, MSS_GPIO_3,
                MSS_GPIO_4, MSS_GPIO_5, MSS_GPIO_6, MSS_GPIO_7,
};

void delay()
{
        int d = SystemCoreClock / 128;

        while (d-- > 0)
                ;
}

int main(void)
{
        int i, current_val;

        MSS_GPIO_init();

        /* Configure SW1 as input */
        MSS_GPIO_config(MSS_GPIO_8, MSS_GPIO_INPUT_MODE);

        /* Init & turn on all LEDs */
        for (i = 0; i < 8; i++) {
                MSS_GPIO_config(LED[i], MSS_GPIO_OUTPUT_MODE);
                MSS_GPIO_set_output(LED[i], 0);
        }

        delay();
        delay();
        delay();
        delay();
```

```
        /* Blink all LEDs on start up */
        for (i = 0; i < 8; i++) {
                for (current_val = 0; current_val < 8; ++current_val)
                        MSS_GPIO_set_output(LED[current_val], 1);

                delay();

                for (current_val = 0; current_val < 8; ++current_val)
                        MSS_GPIO_set_output(LED[current_val], 0);

                delay();
        }

        /* Sequence-blink */
        for (;;) {
                /* Switch LED pattern when SW1 is pressed */
                if ((1 << MSS_GPIO_8) & MSS_GPIO_get_inputs()) {
                        for (i = 7; i > -1; i--) {
                                current_val = (MSS_GPIO_get_outputs() & (1 <<
LED[i])) ? 1 : 0;
                                MSS_GPIO_set_output(LED[i], current_val ^ 1);
                                delay();
                        }
                } else {
                        for (i = 0; i < 8; i++) {
                                current_val = (MSS_GPIO_get_outputs() & (1 <<
LED[i])) ? 1 : 0;
                                MSS_GPIO_set_output(LED[i], current_val ^ 1);
                                delay();
                        }
                }
        }
}
```

MOD 3:

Counter Module Used, some code commented in and out depending on usage

```
module project2 (
  input wire clk,                 // Clock input
  output wire [15:0] count,       // 16-bit counter output
  output wire tc,                 // Terminal count output
  input wire enable,              // Enable input
  input wire reset                // Reset input
```

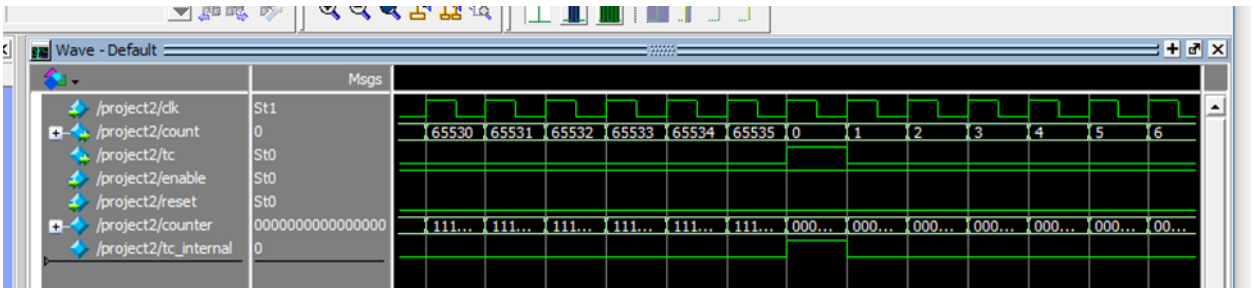
```
);
  reg [15:0] counter = 16'b0;  // 16-bit counter register
  reg tc_internal = 1'b0;   // Internal signal for terminal count
//  counter = 16'b0;

  always @(posedge clk or posedge reset) begin
     if (reset) begin
       counter <= 16'b0;  // Reset the counter to zero
     end else if (enable) begin
              counter <= (counter == 16'hFFFF) ? 16'b0 : counter + 1;  // Increment the
counter and reset at terminal count
       tc_internal <= (counter == 16'hFFFF) ? 1 : 0;  // Assert internal TC when terminal
count is reached
//      if (counter == 16'hFFFF) begin
//        counter <= 16'b0;  // Reset to zero at the terminal count
//        tc_internal <= 1'b1; // Assert internal TC when terminal count is reached
//      end else begin
//        counter <= counter + 1;  // Increment the counter
////        tc_internal <= 1'b0;      // Deassert internal TC when not at terminal count
//      end
    end
  end

  assign tc = tc_internal; // Connect the internal TC to the output
  assign count = counter;  // Connect the counter to the output

endmodule
```

TopLevel code used to instantiate all the counter blocks

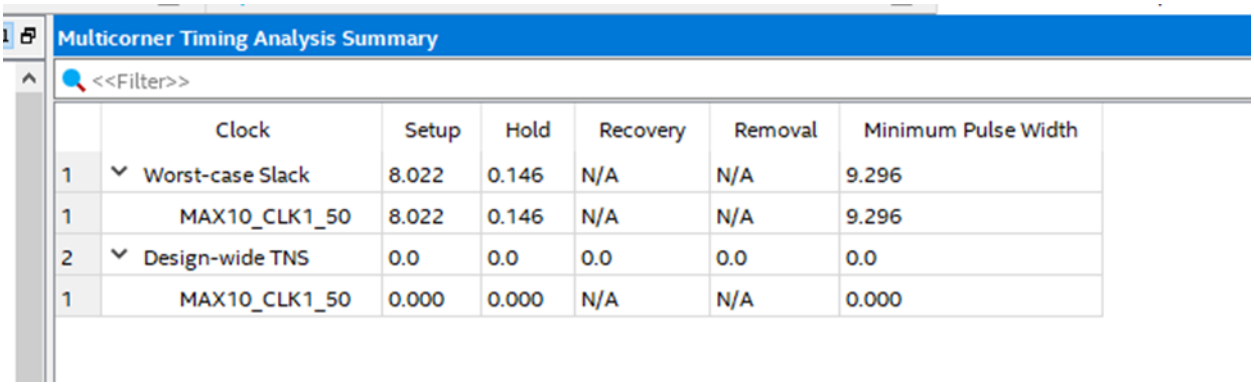
```
//DE10 Lite: 2100
//Smartfusion 2: 391
//DE1 Soc: 2673
module TopLevel
#(parameter num = 2100 )
(
  input [1:0] SW,
  input MAX10_CLK1_50,              // Clock input
  output [1:1] LEDR,
  output final_count  // Output from the last counter in the chain
);


  wire reset;              // Reset input
  wire [num-1:0] tc_signals;  // Array of terminal count signals for the counters
```

```
  wire [15:0] counter_arr [0:num-1];
  reg endtc;
  // Instantiate and connect a chain of counters using a generate loop
  genvar i;
  generate
    for (i = 0; i < num; i = i + 1) begin : counter_block
      project2 counter (
        .clk(MAX10_CLK1_50),
        .count(counter_arr[i]),
        .tc(tc_signals[i]),
        .enable(i == 0 ? SW[0] : tc_signals[i-1]),//enable_signals[i]),
        .reset(reset)
      );

    end
  endgenerate

  always @(posedge tc_signals[1] or posedge reset) begin
          if ( reset == 1 )
             endtc <= 0;
          else
             endtc <= 1;
  end


  assign reset = SW[1];
  assign final_count = tc_signals[num-1];  // Connect the final count to the output
  assign LEDR[1] = endtc;

endmodule
```

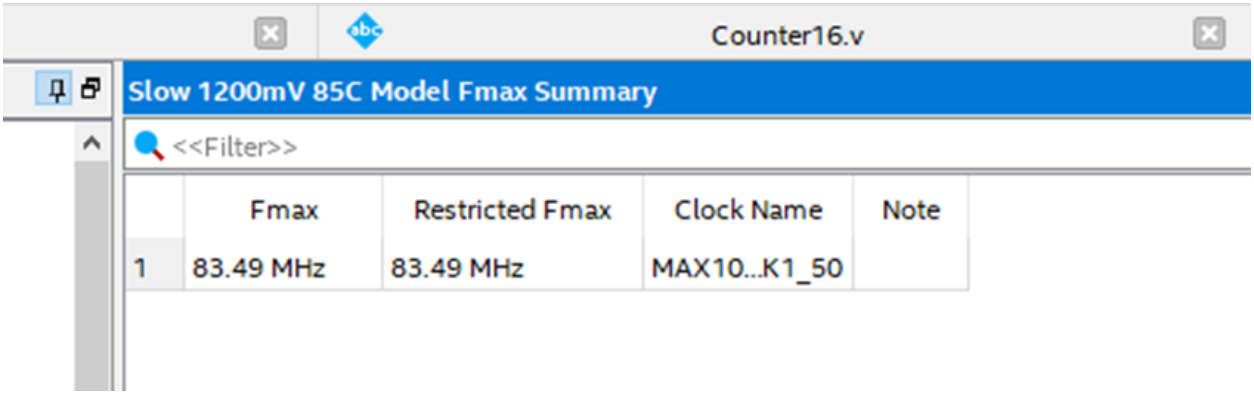
DE10 Lite



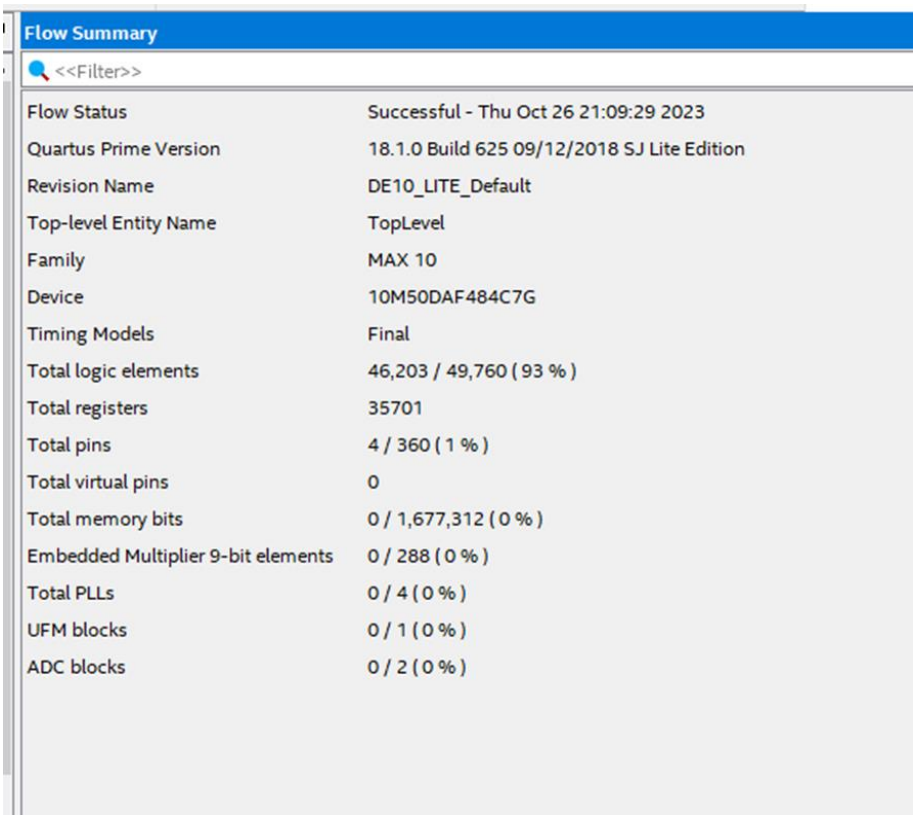
DE10 Lite: Simulation Results



DE10 Lite: Closed Timing



DE10 Lite: Fmax results

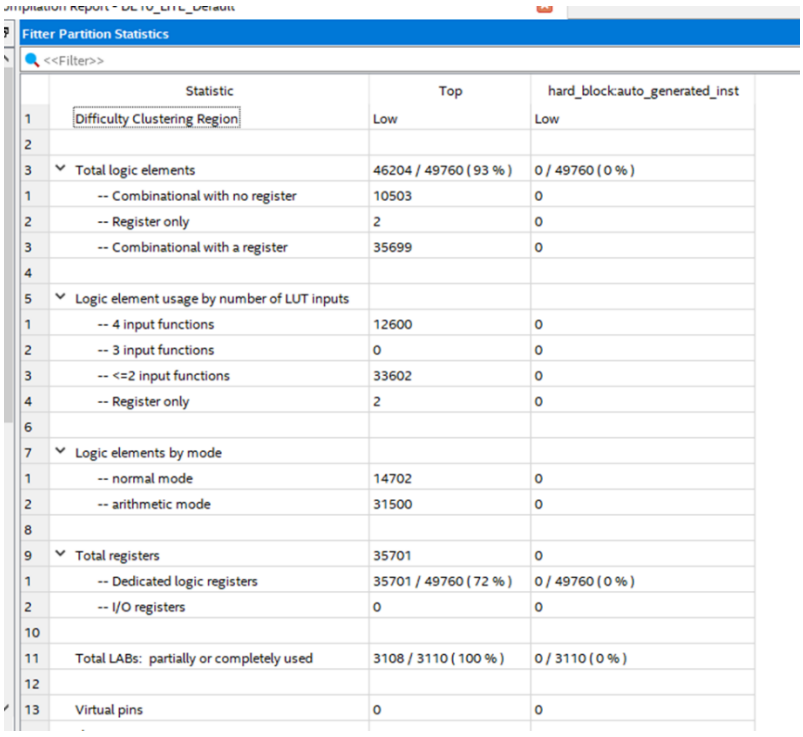


Flow Summary

<<Filter>>

Flow Status	Successful - Thu Oct 26 21:09:29 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	DE10_LITE_Default
Top-level Entity Name	TopLevel
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	46,203 / 49,760 (93 %)
Total registers	35701
Total pins	4 / 360 (1 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

DE10 Lite: Logic Utilization

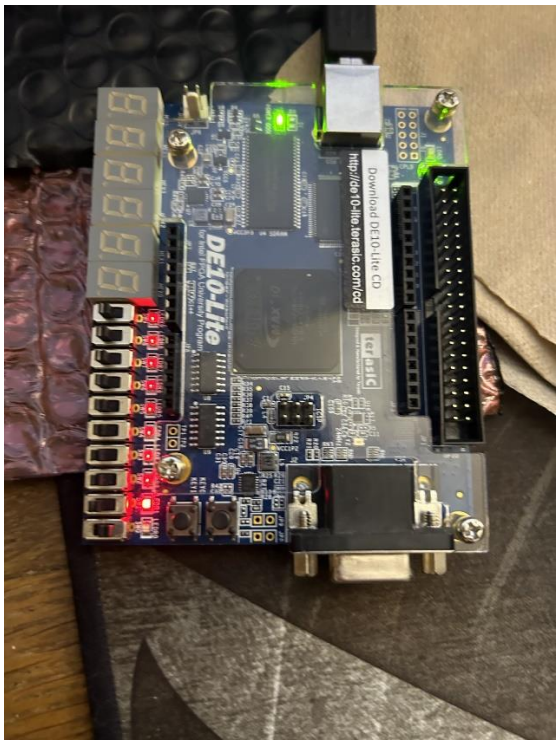


Fitter Partition Statistics

<<Filter>>

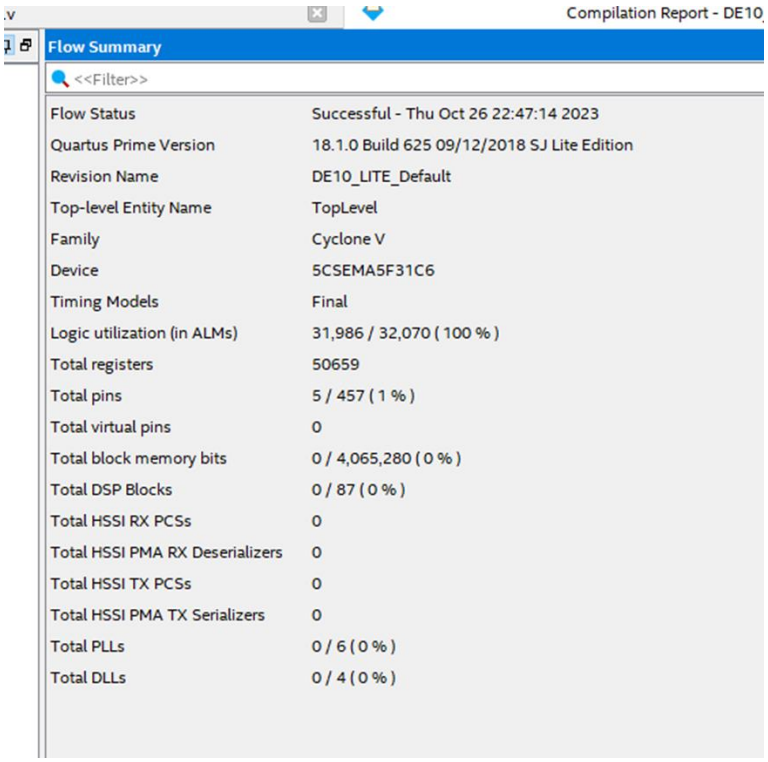
	Statistic	Top	hard_block:auto_generated_inst
1	Difficulty Clustering Region	Low	Low
2			
3	Total logic elements	46204 / 49760 (93 %)	0 / 49760 (0 %)
1	-- Combinational with no register	10503	0
2	-- Register only	2	0
3	-- Combinational with a register	35699	0
4			
5	Logic element usage by number of LUT inputs		
1	-- 4 input functions	12600	0
2	-- 3 input functions	0	0
3	-- <=2 input functions	33602	0
4	-- Register only	2	0
6			
7	Logic elements by mode		
1	-- normal mode	14702	0
2	-- arithmetic mode	31500	0
8			
9	Total registers	35701	0
1	-- Dedicated logic registers	35701 / 49760 (72 %)	0 / 49760 (0 %)
2	-- I/O registers	0	0
10			
11	Total LABs: partially or completely used	3108 / 3110 (100 %)	0 / 3110 (0 %)
12			
13	Virtual pins	0	0

DE10 Lite: Resource Utilization showing LABs being the limiting factor.

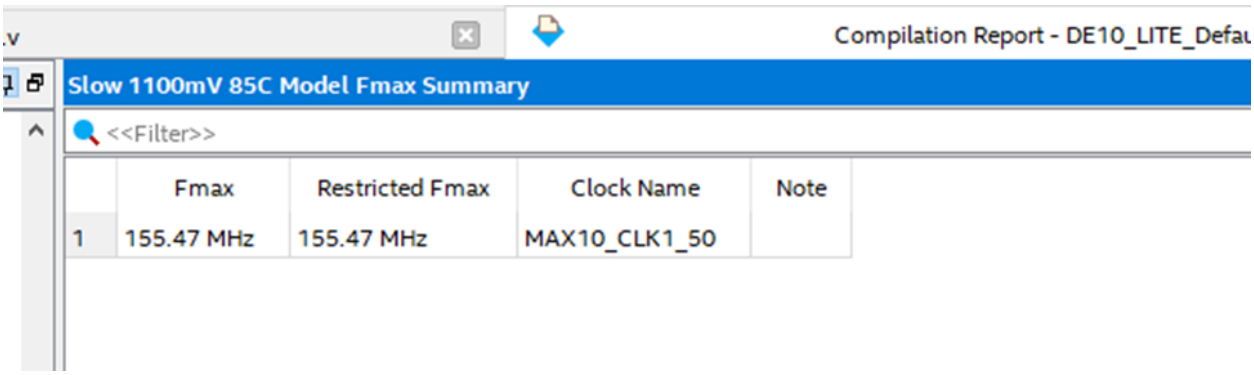


DE10 Lite: TC verification on board on LED[1]

DE1 SOC

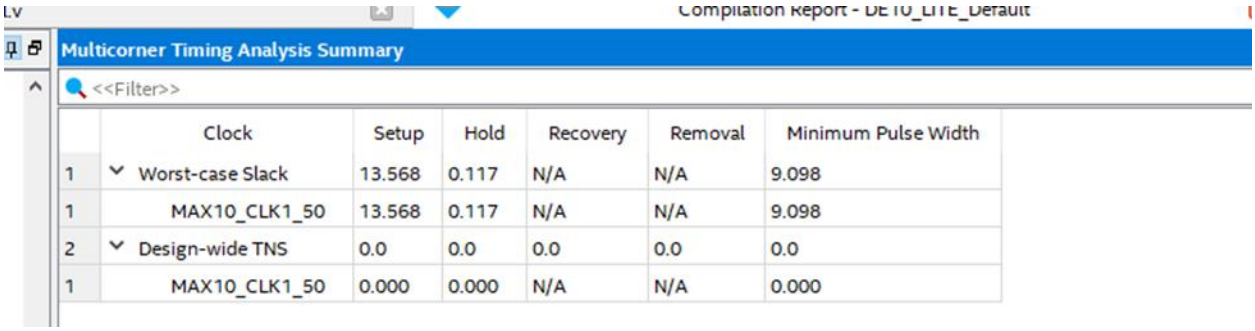


DE1 SOC: Logic Utilization



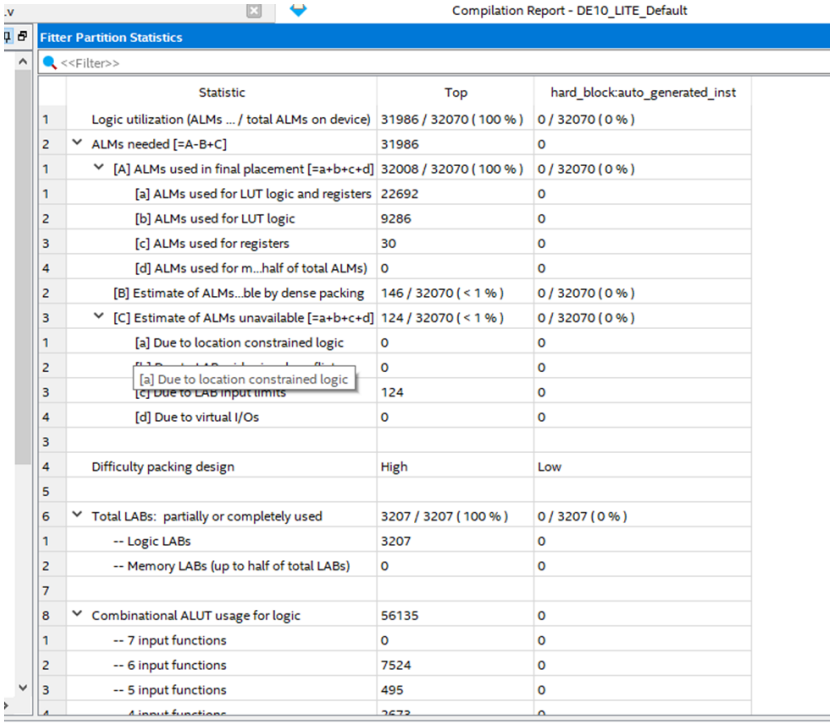
	Fmax	Restricted Fmax	Clock Name	Note
1	155.47 MHz	155.47 MHz	MAX10_CLK1_50	

DE1 SOC: Fmax results



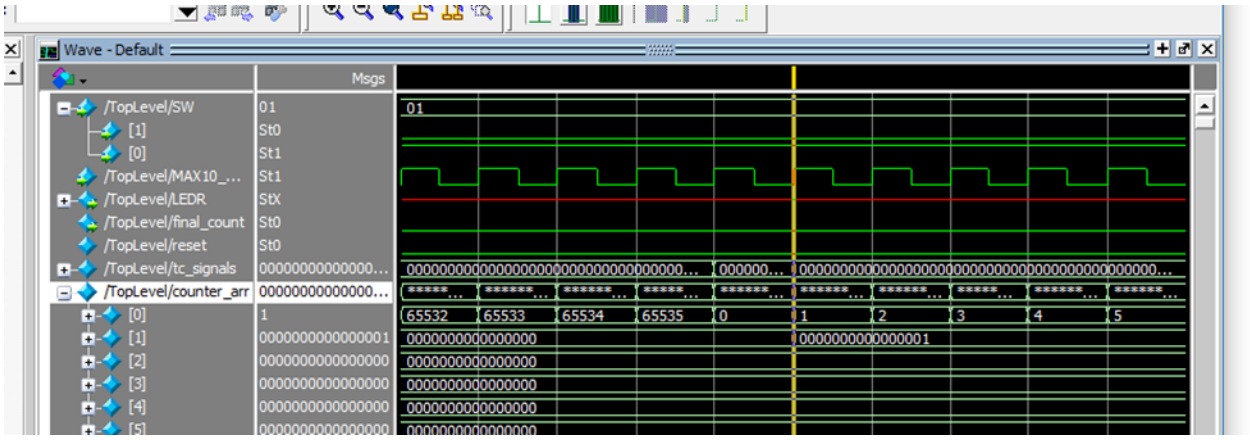
	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	13.568	0.117	N/A	N/A	9.098
1	MAX10_CLK1_50	13.568	0.117	N/A	N/A	9.098
2	Design-wide TNS	0.0	0.0	0.0	0.0	0.0
1	MAX10_CLK1_50	0.000	0.000	N/A	N/A	0.000

DE1 SOC: Closed timing results



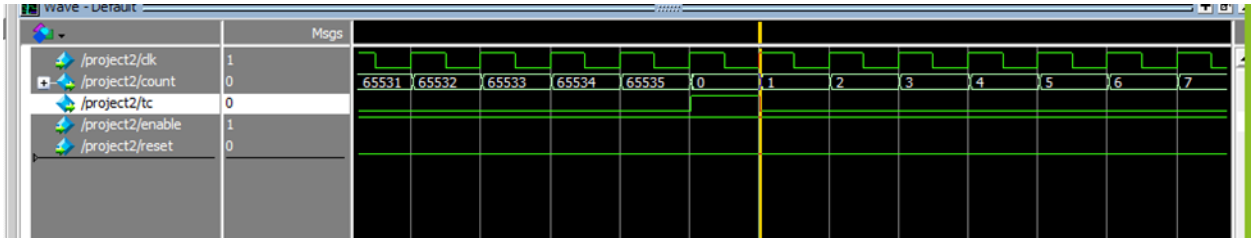
	Statistic	Top	hard_block:auto_generated_inst
1	Logic utilization (ALMs ... / total ALMs on device)	31986 / 32070 (100 %)	0 / 32070 (0 %)
2	ALMs needed [=A-B+C]	31986	0
1	[A] ALMs used in final placement [=a+b+c+d]	32008 / 32070 (100 %)	0 / 32070 (0 %)
1	[a] ALMs used for LUT logic and registers	22692	0
2	[b] ALMs used for LUT logic	9286	0
3	[c] ALMs used for registers	30	0
4	[d] ALMs used for m...half of total ALMs)	0	0
2	[B] Estimate of ALMs...ble by dense packing	146 / 32070 (< 1 %)	0 / 32070 (0 %)
3	[C] Estimate of ALMs unavailable [=a+b+c+d]	124 / 32070 (< 1 %)	0 / 32070 (0 %)
1	[a] Due to location constrained logic	0	0
2		0	0
3	[c] Due to LAB input limits	124	0
4	[d] Due to virtual I/Os	0	0
3			
4	Difficulty packing design	High	Low
5			
6	Total LABs: partially or completely used	3207 / 3207 (100 %)	0 / 3207 (0 %)
1	-- Logic LABs	3207	0
2	-- Memory LABs (up to half of total LABs)	0	0
7			
8	Combinational ALUT usage for logic	56135	0
1	-- 7 input functions	0	0
2	-- 6 input functions	7524	0
3	-- 5 input functions	495	0

DE1 SOC: Resource information showing LABS utilization

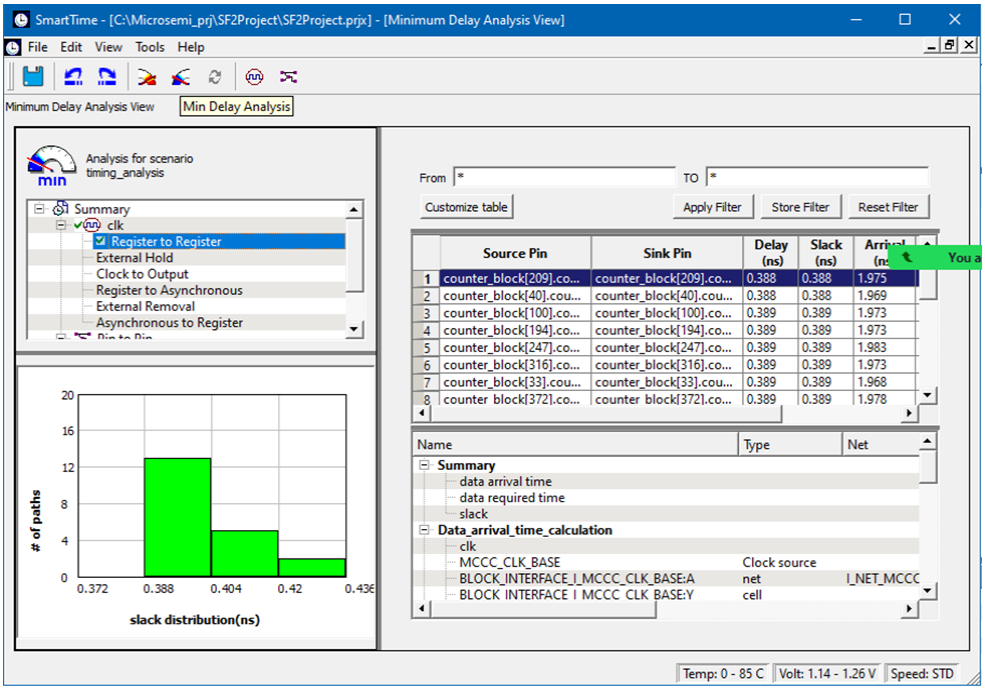


DE1 SOC: Simulation Results

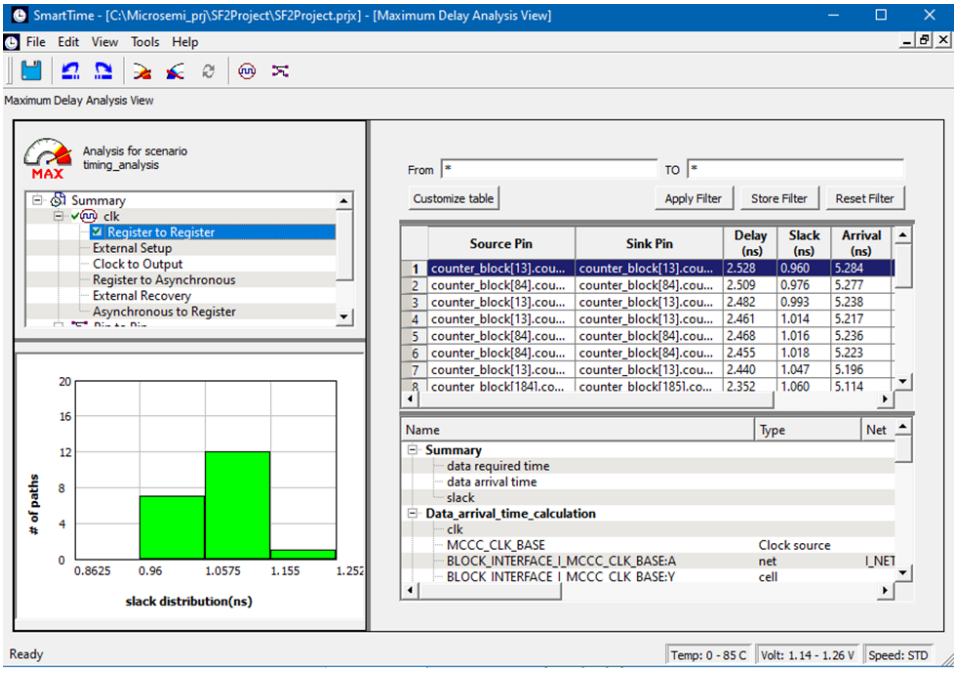
Smart Fusion 2



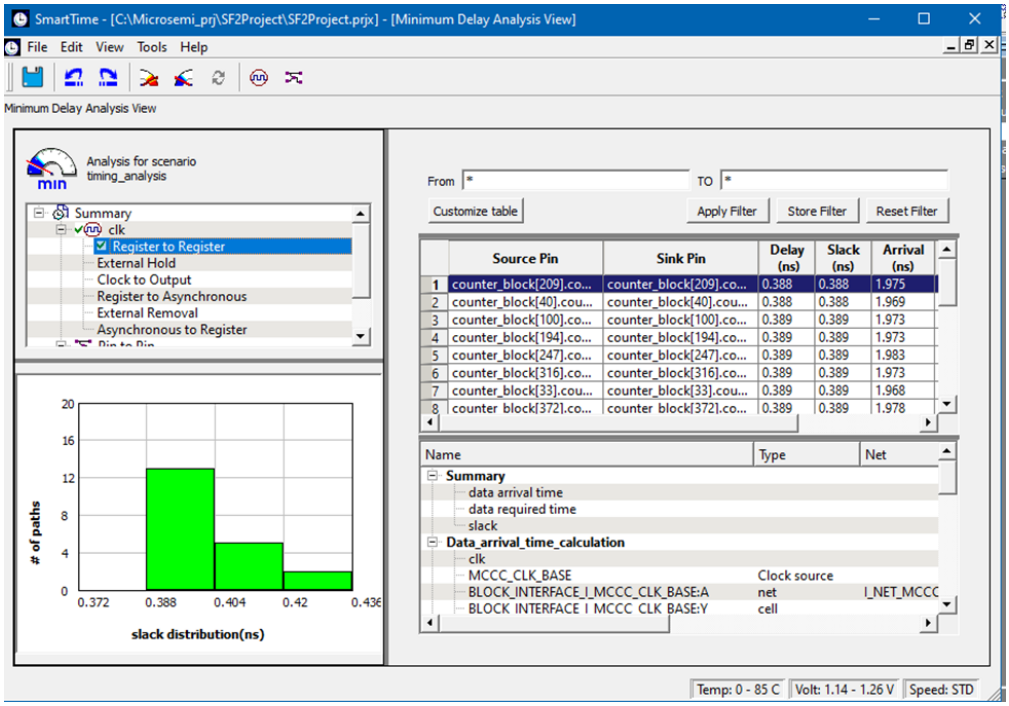
Smart Fusion 2: Simulation Results



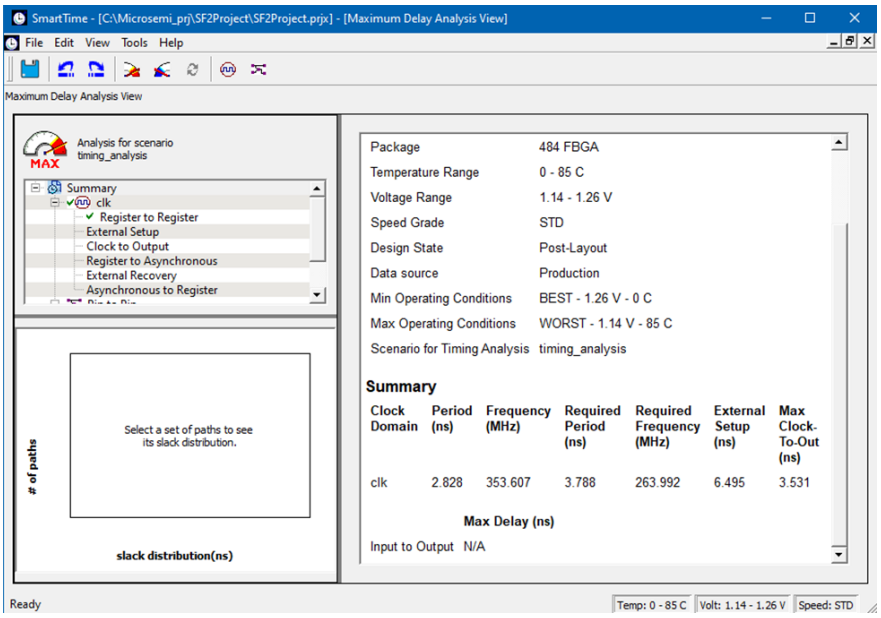
Smart Fusion 2: Closed Timing Verification #1



Smart Fusion 2: Closed Timing Verification #2



Smart Fusion 2: Closed Timing Verification #3



Smart Fusion 2: Fmax reporting through Libero

Resource Usage

Type	Used	Total	Percentage
4LUT	8997	12084	74.45
DFF	6647	12084	55.01
I/O Register	0	699	0.00
User I/O	0	233	0.00
-- Single-ended I/O	0	233	0.00
-- Differential I/O Pairs	0	116	0.00
RAM64x18	0	22	0.00
RAM1K18	0	21	0.00
MACC	0	22	0.00
Chip Globals	2	8	25.00
CCC	0	2	0.00
RCOSC_25_50MHZ	0	1	0.00
RCOSC_1MHZ	0	1	0.00
XTLOSC	0	1	0.00
MSS	0	1	0.00

Smart Fusion 2: Resource Utilization

Extra Credit:

```
from pynq.overlays.base import BaseOverlay
import asyncio

base = BaseOverlay("base.bit")

async def push_buttons(num):
    button_presses = []  # Initialize an empty list to store button press values
    while len(button_presses) < 2:  # Continue until 2 inputs are collected
        await base.buttons[num].wait_for_value_async(1)
        await asyncio.sleep(0.5)
        if num == 0:
            print("-")
            button_presses.append("-")  # Add the input to the list
        elif num == 1:
            print(".")
            button_presses.append(".")  # Add the input to the list

    # Print the collected button presses
    button_pattern = ''.join(button_presses)
    if button_pattern == "..":
        print("A")
    elif button_pattern == "--":
        print("B")
    else:
        print("Unknown pattern")

tasks = [asyncio.ensure_future(push_buttons(i)) for i in range(2)]

if base.switches[0].read():
    print("Set switch 0 low before running")
else:
    base.switches[0].wait_for_value(1)
```

```
In [*]:
```

```python
from pynq.overlays.base import BaseOverlay
import asyncio

base = BaseOverlay("base.bit")

async def push_buttons(num):
    button_presses = []  # Initialize an empty list to store button press values
    while len(button_presses) < 2:  # Continue until 2 inputs are collected
        await base.buttons[num].wait_for_value_async(1)
        await asyncio.sleep(0.5)
        if num == 0:
            print("-")
            button_presses.append("-")  # Add the input to the list
        elif num == 1:
            print(".")
            button_presses.append(".")  # Add the input to the list

    # Print the collected button presses
    button_pattern = ''.join(button_presses)
    if button_pattern == "..":
        print("A")
    elif button_pattern == "--":
        print("B")
    else:
        print("Unknown pattern")

tasks = [asyncio.ensure_future(push_buttons(i)) for i in range(2)]

if base.switches[0].read():
    print("Set switch 0 low before running")
else:
    base.switches[0].wait_for_value(1)
```

```
-
-
B
```

Extra credit output for module 3