Homework 7: VGA Circle Report
December 12, 2017
Sabbir Ahmed

1 Background

In this project students will explicitly implement a computational finite-state machine, utilize rescheduling and resource sharing, and become familiar with the concept of using an on-chip clock multiplier. Students will leverage the faster clock to implement computations in a serial fashion. In this project, students will display a circle on the screen, examine analysis reports, and modify synthesis options.

2 Implementation

Multiple designs were implemented to analyze their effects on resource sharing and timing constraints.

2.1 Single Cycle Computation Design

The initial design implemented the entire inequality in a single cycle. Since the design emphasized on the computation being performed in a single cycle, explicitly generating several registers to hold the constant state value was unnecessary. An additional state was included for the synthesizer to consider encoding the FSM. The single-state module successfully generated the circle on the VGA screen using the formula $(x - x_c)^2 + (y - y_c)^2 < 10000$. The module is initialized with an asynchronous reset.

2.1.1 States

Table 1 provides the FSM states encoded by the synthesizer. The automatic-encoding encoded states do not differ from the values assigned to them during initialization because of the small number of states. The states were intentionally assigned with 2-bit values to alert the synthesizer of the FSM.

1. **INIT (00):** Serves as a buffer to the computational state. This state serves no other purpose.

Table 1: FSM state encoding generated by the synthesizer for the single cycle design.

State	Encoding
00	00
01	01

2. **COMPUTE (01):** Computes the entire circle inequality.

2.1.2 Testbench

Figure 1 provides the waveforms generated by sample coordinates (x, y) to the module.

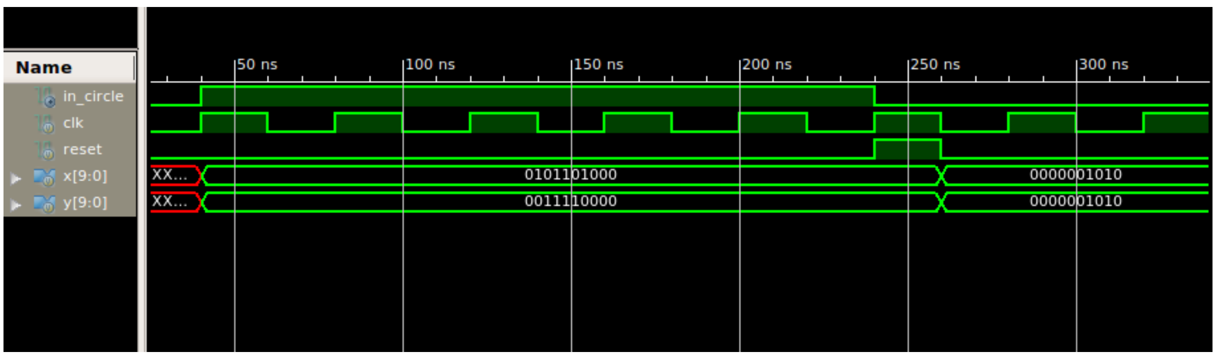

Figure 1: Single cycle computation design demonstrating the circle flag (`in_circle`) activated at $(360, 240)$ and deactivated at $(10, 10)$.

2.1.3 Synthesis

As expected, the design did not meet the timing constraints. Figure 2 provides the summary of the time constraints report, where the constraint `TS_uut_CLK0_BUF` was not met.

```
Derived Constraint Report
Review Timing Report for more details on the following derived constraints.
To create a Timing Report, run "trce -v 12 -fastpaths -o design_timing_report design.ncd design.pcf"
or "Run Timing Analysis" from Timing Analyzer (timingan).
Derived Constraints for TS_CLK_50MHZ
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|                               |   Period    |       Actual Period       |      Timing Errors        |      Paths Analyzed       |
|           Constraint          | Requirement |-------------+-------------|-------------+-------------|-------------+-------------|
|                               |             |   Direct    | Derivative  |   Direct    | Derivative  |   Direct    | Derivative  |
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|TS_CLK_50MHZ                   |     20.000ns|      7.500ns|     33.472ns|            0|            2|            0|       115653|
| TS_dcm_uut_CLK0_BUF           |     20.000ns|     33.472ns|          N/A|            2|            0|       114964|            0|
| TS_dcm_uut_CLK2X_BUF          |     10.000ns|      9.263ns|          N/A|            0|            0|          689|            0|
+-------------------------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+

1 constraint not met.
```

Figure 2: Screen capture of the timing constraint report showing failure of `TS_uut_CLK0_BUF` of the DCM.

Table 2: Timing slacks of `TS_uut_CLK0_BUF`

Check	**Worst Case Slack**
SETUP	-6.736 ns
HOLD	1.003 ns

Table 3 provides the macro statistics generated by the synthesizer.

Table 3: Macro statistics generated by the synthesizer for the single cycle design.

# Multipliers	**2**
11x11-bit multiplier	2
# Adders/Subtractors	**4**
10-bit adder	1
11-bit subtractor	2
23-bit adder	1
# Counters	**2**
10-bit up counter	2
# Registers	**8**
1-bit register	8
# Comparators	**1**
24-bit comparator less	1

The multipliers are used to multiply the two 21-bit squared coordinates. Several adders and subtractors are used in the design to handle `pos_v`, `pos_h` and the centers coordinates.

2.1.4 Files

The design for the single cycle computation was modularized and implemented as a standalone top-level design. The file is provided as `top_single_cycle.v` which utilizes the single cycle modules exclusively.

2.2 Multiple Cycle Computation Design with Resource Sharing

The multiple cycles design distributed the computations over multiple cycles in multiple states. The design utilized 4 states to compute the circle inequality. Since this design accumulated its final output over multiple cycles, several registers had to be initialized with sufficient capabilities. Since the coordinates are 10-bits in size, their highest decimal value is $2^10 - 1 = 1023$. Squaring that integer would result in a 20-bit number. Therefore, temporary registers of 20-bits were allocated to hold values for the computation.

2.2.1 States

Table 4 provides the FSM states encoded by the synthesizer. The synthesizer encoded the states with one-hot encoding, differing from the binary values initialized to them.

Table 4: FSM state encoding generated by the synthesizer for the multiple cycle design.

State	Encoding
00	0001
01	0010
10	0100
11	1000

1. **INIT (0001):** Subtracts the coordinates from the corresponding center coordinates. Parameters `XLEFT` of value 320 and `YBOTTOM` of value 240 were used as the center coordinates of a standard VGA monitor.

2. **SQUAREX (0010):** Multiplies the x coordinate with itself, and stores it in the temporary `mul_temp`.

3. **SQUAREY (0100):** Stores the previous value of `mul_temp` to a different variable `x_temp`. Multiplies the y coordinate with itself, and stores it in the temporary `mul_temp`.

4. **ADDCMP (1000):** Adds `mul_temp` to `x_temp`, and then compares with the radius. This comparison generates a 1-bit signal for the circle flag, terminating the computation.

2.2.2 Testbench

Figure 1 provides the waveforms generated by sample coordinates (x, y) to the module.

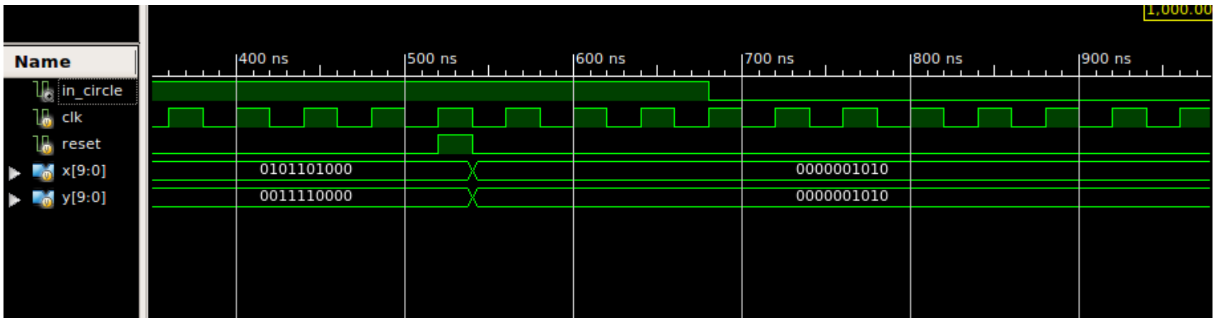

Figure 3: Multiple cycle computation design demonstrating the circle flag (`in_circle`) activated at $(360, 240)$ and deactivated at $(10, 10)$.

2.2.3 Synthesis

The design successfully synthesized while meeting the timing constraints. Figure 2 provides the summary of the time constraints report.

```
*************************
Generating Clock Report
*************************

+---------------------+--------------+------+------+------------+-------------+
|      Clock Net      |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|           CLK2X_OUT | BUFGMUX_X2Y11| No   |   50 |  0.057     |  0.182      |
+---------------------+--------------+------+------+------------+-------------+
|            CLK0_OUT | BUFGMUX_X1Y10| No   |   15 |  0.013     |  0.149      |
+---------------------+--------------+------+------+------------+-------------+
```

Figure 4: Screen capture of the timing constraint report showing the multiple cycle design meeting all timing constraints.

Table 5: Timing slacks for each clock domains.

Constraint	Check	Worst Case Slack
Autotimespec constraint for clock net CLK2X_OUT	SETUP	N/A
	HOLD	1.017ns
Autotimespec constraint for clock net CLK0_OUT	SETUP	N/A
	HOLD	1.200ns

In terms of resource sharing, the design utilized a single multiplier as intended. Table 6 provides the macro statistics generated by the synthesizer.

Table 6: Macro statistics generated by the synthesizer for the multi cycle design.

# Multipliers	**1**
20x20-bit multiplier	1
# Adders/Subtractors	**4**
10-bit adder	1
12-bit subtractor	2
21-bit adder	1
# Counters	**2**
10-bit up counter	2
# Registers	**11**
1-bit register	8
20-bit register	3
# Comparators	**1**
22-bit comparator less	1

The multipliers are used to multiply the two 21-bit squared coordinates. Several adders and subtractors are used in the design to handle `pos_v`, `pos_h` and the centers coordinates.