CSIE DIC Lab 2023

Lab08. Computing Intersection of Two Circles in 2D Space

Description

In this lab, within a two-dimensional coordinate system, the center coordinates (x, y) and radius (r) of two circles will be given, as shown in Figure 1. In Figure 1, for the smaller circle (r=2), the set of integer coordinate points covered by this circle within the two-dimensional coordinate system ranging from (x, y) = (1, 1) to (8, 8) is called **Set A**. The definition of being covered by a circle includes integer coordinate points inside the circle or on the circumference. For the larger circle (r=3), the set of integer coordinate points covered by this circle within the two-dimensional coordinate system ranging from (x, y) = (1, 1) to (8, 8) is called **Set B**.

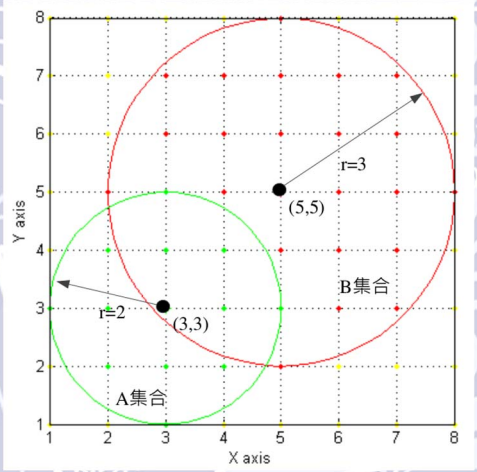


Figure 1: 8×8 coordinate plane

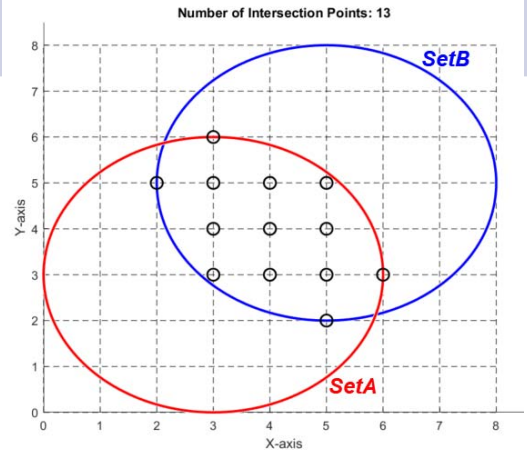


Figure 2: Intersection of integer coordinate points jointly covered by two circles

In this lab, the objective is to design a circuit that computes the number of elements in the intersection of **Set A** and **Set B** within a two-dimensional coordinate system, spanning from (x, y) = (1, 1) to (8, 8). As illustrated in Figure 2, the input consists of

(x1, y1) = (3, 3), r1 = 3, which generates **Set A**, and (x2, y2) = (5, 5), r2 = 3, which generates **Set B**. The intersection of these two sets contains the following elements: (2, 5), (3, 3), (3, 4), (3, 5), (3, 6), (4, 3), (4, 4), (4, 5), (5, 2), (5, 3), (5, 4), (5, 5), and (6, 3). Consequently, the total number of elements in the intersection amounts to 13.

Figure 3 presents an additional example. In this lab, the focus is solely on integer coordinate points within the range of (x, y) = (1, 1) to (8, 8). Consequently, in this particular example, the elements in the intersection of the two sets are calculated exclusively for integer coordinate points within the specified range, resulting in a total of 7 elements.

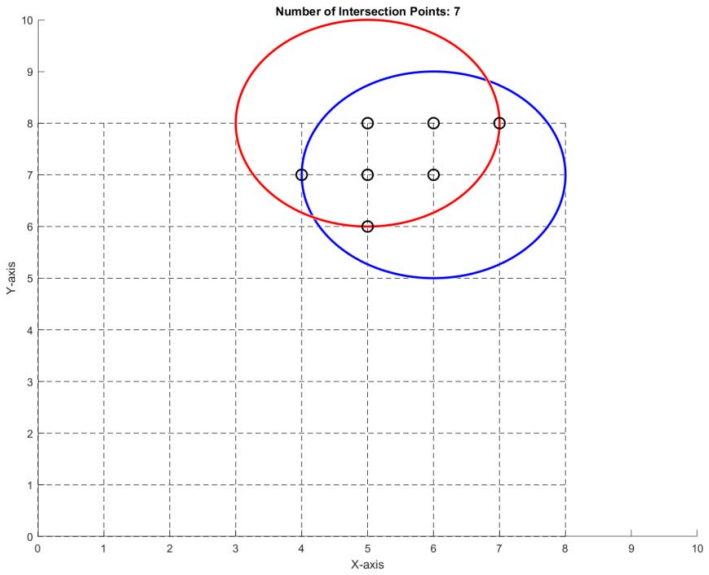


Figure 3: Consider only specified range of (1,1) to (8,8)

Specification

1. Top module name: SET
2. Top module filename: SET.v
3. Input/output definition

Signal Name	Direction	Width(bit)	Description
rst	input	1	Reset signal.
clk	input	1	Clock signal.
en	input	1	Data valid signal. When this signal is asserted, it means the input data is valid.
central	input	16	Coordinate data of set. **central[15:12]** : X axis coordinates of set A (**x1**).

			central[11:8] : Y axis coordinates of set A (**y1**). **central[7:4]** : X axis coordinates of set B (**x2**). **central[3:0]** : Y axis coordinates of set B (**y2**).
radius	input	8	Radius data. **radius[7:4]** : Radius of set A (**r1**). **radius[3:0]** : Radius of set A (**r2**).
busy	output	1	System busy indication signal. When this signal is asserted, it means the system is busy.
valid	output	1	Output indication signal for valid data. When the signal valid is at a high level, the data output from the candidate output port is valid data.
candidate	output	8	Output the number of elements in the intersection.

4. All input signal are **synchronized at clock rising edge.**
5. The reset scheme is **active high asynchronous reset.**
6. In the test pattern, there have 64 input patterns.
7. Following the completion of the circuit reset, the test module, upon detecting **busy=0** with the positive edge trigger of the clock signal, proceeds to raise the **en** signal to a high voltage and inputs new coordinate and radius data. Then, **en=1** signal is maintained for one cycle before being lowered back to a low voltage. The input timing diagram is depicted in Figure 4.

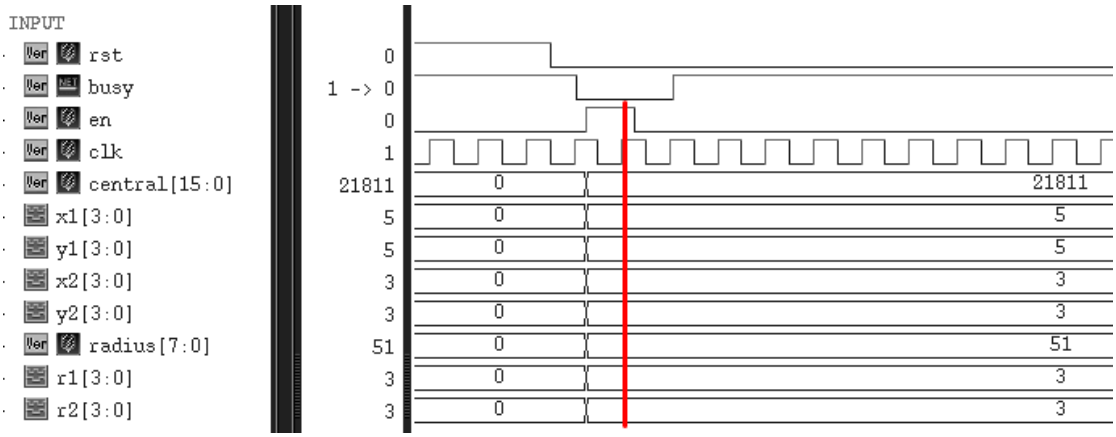


Figure 4: Input timing diagram

8. Upon receiving a set of input data, the circuit must raise the **busy** signal to a high voltage, indicating that the circuit is in the process of computation. Once the circuit completes the calculation and is ready to output the result, the **busy** signal is lowered to a low voltage, while simultaneously raising the **valid** signal to a high voltage. At this point, the **candidate** signal must be prepared to output the answer. The test module then captures the circuit output signal at the positive edge of the clock and compares it to the expected result. Following the lowering of the **valid** signal to a low voltage, the test module initiates the preparation of the next set of test data for input.

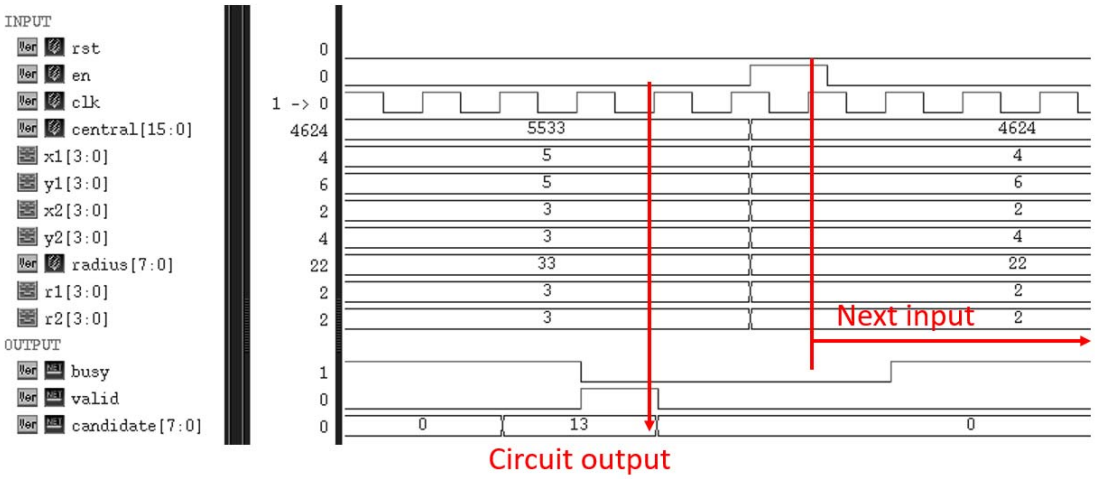
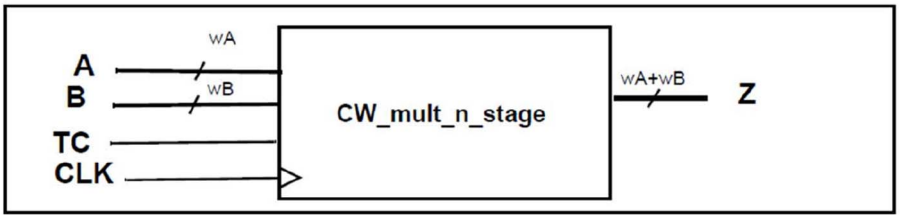


Figure 5: Output timing diagram

9. The output data will be checked only when **valid** is high.
10. The chipware component, **CW_mult_n_stage**, depicted in Figure 6, is employed to execute pipelined multiplication of two integer numbers **A** and **B**, generating the output **Z** with configurable pipeline stages. The bit widths of **A** and **B** are independently parameterizable, and the stages parameter governs the number of pipeline stages. **In this lab, the stages should be set to 3**, ensuring that the multiplication operates correctly within the timing constraints.

Pipelined Multiplier



Port Description

Port Name	Type	Size (bits)	Description
A	Input	*wA*	Integer multiplier
B	Input	*wB*	Integer multiplicand
TC	Input	1	0: A and B are interpreted as unsigned integers 1: A and B are interpreted as signed integers
CLK	Input	1	Clock
Z	Output	*wA+wB*	Integer pipelined product

Parameter Description

Parameter Name	Legal Range	Default	Description
wA	≥1	8	Bit width of A
wB	≥1	8	Bit width of B
stages	≥1	2	Number of pipeline stages

Figure 6: I/O ports of CW_mult_n_stage

11. You should perform low-power synthesis to optimize the power consumption of your circuit. In addition, you should provide information about the power of your circuit with and without low-power synthesis.
12. In logic synthesis, the timing constraint for the clock period **MUST** be **10ns**. Thus your design should operate correctly at this clock speed.

Data Preparation

1. Extract LAB data from TA's directory:
   % tar xzvf ~dicta/lab08.tar.gz
2. The extracted LAB directory (**lab08**) contains:
   a. **Democase/** :Design example, including RTL code and synthesis script
   b. **Exercise/** :Test pattern for this lab

Demo Case

3. Change directory to lab08:
   % cd lab08
4. Change to directory: **Democase/**
   % cd Democase
5. This directory contains:
   a. **RTL/** :RTL source code of design example

b. Synthesis/ :Synthesis script

c. Netlist/ :Synthesized netlist and test pattern for gate-level simulation

Generate the Toggle file (Switching Activities)

6. Change to directory: **RTL/**
7. Open and edit "TEST.v", add the following system tasks to generate TCF file during Verilog RTL simulation.

```
……
21  $toggle_count("TESTBED.I_STACK");
22  $toggle_count_mode(1);
……
200 $toggle_count_report_flat("STACK_rtl.tcf", "TESTBED.I_STACK");
……
```

(You only need to remove comments // in these three lines)

8. Run Verilog RTL simulation and then generate the required switching activities file "**STACK_rtl.tcf**"

   % ncverilog –f run.f

9. Check the "**STACK_rtl.tcf**" file

   You can find that a lot of pin/net information in this file, the format is

   "**PIN/NET NAME**" : "**Probability of being 1**" "**toggle count**"

   Please find the "Probability of being 1" of the "***CLK***" PIN:_____

   Please find the "toggle count" of the "***CLK***" PIN:_____

Synthesis without power optimization

10. Change to directory: **Synthesis/**
11. Open and read the synthesis script and design constraint file.

    % gedit SYN018_RC.tcl &

    % gedit SYN_RTL.sdc &

12. Start logic synthesis,

    % ./01_run_synthesis

13. Open and read the synthesis report

    % gedit report.area &

    % gedit report.timing &

    % gedit report.power &

    What is the area of the module STACK?_____

    What is the slack time of the critical path?_____

14. Change to the directory: **Netlist/**
15. Open and edit "**TEST_gate.v**" as following

……

20 $toggle_count("TESTBED.I_STACK");

……

200 $toggle_count_report_flat("STACK_gate.tcf", "TESTBED.I_STACK");

……

(You only need to remove comments // in these two lines)

16. Run the gate-level simulation and generate the required switching activities file "**STACK_gate.tcf**",
    % ncverilog –f run.f
17. Return to the directory: **Synthesis/**
18. Report the power consumption of this design after gate-level simulation.
    % ./02_report_power
19. Open and read the power report after gate-level simulation
    % gedit report.power.gate &
    What is the power consumption of the module STACK without power optimization?_____

Synthesis with power optimization

20. Change to directory: **Synthesis/**
21. Clean up previous synthesized results,
    % ./03_clean_up
22. Open and read the synthesis script and design constraint file with low-power synthesis
    % gedit SYN018_RC_LP.tcl &
    % gedit SYN_RTL.sdc &
23. Start low power synthesis,
    % ./04_run_low_power_syn
24. Open and read the synthesis report
    % gedit report.area &
    % gedit report.timing &
    % gedit report.power &
    What is the area of the module STACK?_____
    What is the slack time of the critical path?_____
25. Change to the directory: **Netlist/**
26. Run the gate-level simulation and generate the required switching activities file "**STACK_gate.tcf**",
    % ncverilog –f run.f
27. Return to the directory: **Synthesis/**
28. Report the power consumption of this design after gate-level simulation.

% ./05_report_power

29. Open and read the power report

    % gedit report.power.gate &

    What is the power consumption of the module STACK with power optimization?

    _____

30. Compare the power consumption to **″without power optimization″**. Does power decrease a lot? Why or why not? (hint: open and reading the report **report.clock_gating**)

    _____

Exercise

31. Change directory to lab08:

    % cd lab08

32. Change to the directory: **Exercise/**

    % cd Exercise

33. This directory contains:

    a. dat/Central_pattern.dat : Input data
    b. dat/Radius_pattern.dat : Input data
    c. dat/result.dat : Output data
    d. TEST.v : Design test bench for RTL simulation
    e. TEST_ gate.v : Design test bench for Gate-level simulation
    f. SET.v : Empty design module
    g. SET.rc : nWave saved signals file
    h. SYN_RTL.sdc : Design constraint file for logic synthesis

RTL Design

34. Write your RTL code in synthesizable coding style:

    % gedit SET.v &
    or % gvim SET.v &
    or % joe SET.v
    or % vim SET.v

35. Run RTL simulation:

    % ncverilog –f run.f

36. Open simulation waveform:

    % nWave &
    Choose File → Open, then select **“SET.fsdb”** and press **OK**

37. Restore signal record for debugging:

    Choose File → Restore Signal, then select **“lab08.rc”** and press **OK**

38. Meaning of signals in TEST group:
    result : correct answer for comparison
    error_cnt : accumulated errors for your answer

Synthesis without power optimization

39. Repeat step 10 to step 19 in Synthesis without power optimization described in the Democase. Also, you need to prepare all of your files required for logic synthesis. We only provide the design constraint file (**SYN_RTL.sdc**) for logic synthesis and the test bench (**TEST_gate.v**) for gate-level simulation.
40. Open and read the power report after gate-level simulation
    % gedit report.power.gate &
    What is the power consumption of the module Kmeans without power optimization?
    _____

Synthesis with power optimization

41. Repeat step 20 to step 30 in Synthesis with power optimization described in Democase. In addition, you need to prepare all of your files required for logic synthesis. We only provide the design constraint file (**SYN_RTL.sdc**) for logic synthesis and the test bench (**TEST_gate.v**) for gate-level simulation.
42. Open and read the power report after gate-level simulation
    % gedit report.power.gate &
    What is the power consumption of the module SET with power optimization?
    _____
43. Compare the power consumption to **"without power optimization"**. Does power decrease a lot? Why or why not? (hint: open and reading the report **report.clock_gating**)
    _____