NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC3050 Advanced Computer Architecture

LAB-1

In this lab, you need to find the area and time complexity of arithmetic circuits like adders and multipliers and circuits for basic logic operations for different bit-widths. Besides, you will experiment with the functionality of ALU as well as the area and time complexity of ALU of different bit-widths. You need to find out which of the arithmetic or logic circuits affect the area and time complexity of ALU significantly, and the impact of word-length on such complexity. You are provided with parametrizable Verilog code where the bit-width can be changed. You should also understand the Verilog code completely and should develop competence to write the code which would be required in your projects.

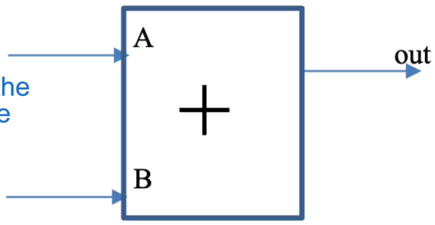
PART I: ARITHMETIC CIRCUITS

In this part of the Lab, we consider a simple adder and a multiplier. You will experiment with the functionality of each arithmetic circuit and find their area and time complexity for different bit-widths.

ARITHMETIC CIRCUITS DESCRIPTION

1. **Adder:** The function of an adder is to add two input operands (A and B) to produce output (out). You can change the bit width of inputs to find the area and delay complexity of the adders of different bit-width.

observation is that the increase is linear, where as the number of bits used increase, the LUT also increase to accomodate to the compute.



what is so special about this lab? The ALU scales exponentially, but not as a high rate as a multiplier. Is the intent to compare to a complex x86 architecture where it must handle multiplication vs a simple RISC that only does logical ops?

Figure-1: Block diagram of adder.

2. **Multiplier:** The function of the multiplier is to multiply two input operands (A and B) to produce the output (out). Assuming both A and B to be W-bit words in 2's complement representation, we can find that 'out' is a (2W)-bit word in 2's complement representation.

it is interesting how the multiplier unit doesnt scale well as we increase the number of bits used for compute Both the LUT and time taken drastically increases as we increase # bit.

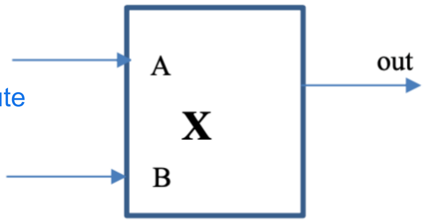


Figure-2: Block diagram of multiplier.

ARITHMETIC CIRCUITS SYNTHESIS, TESTING AND ANALYSIS

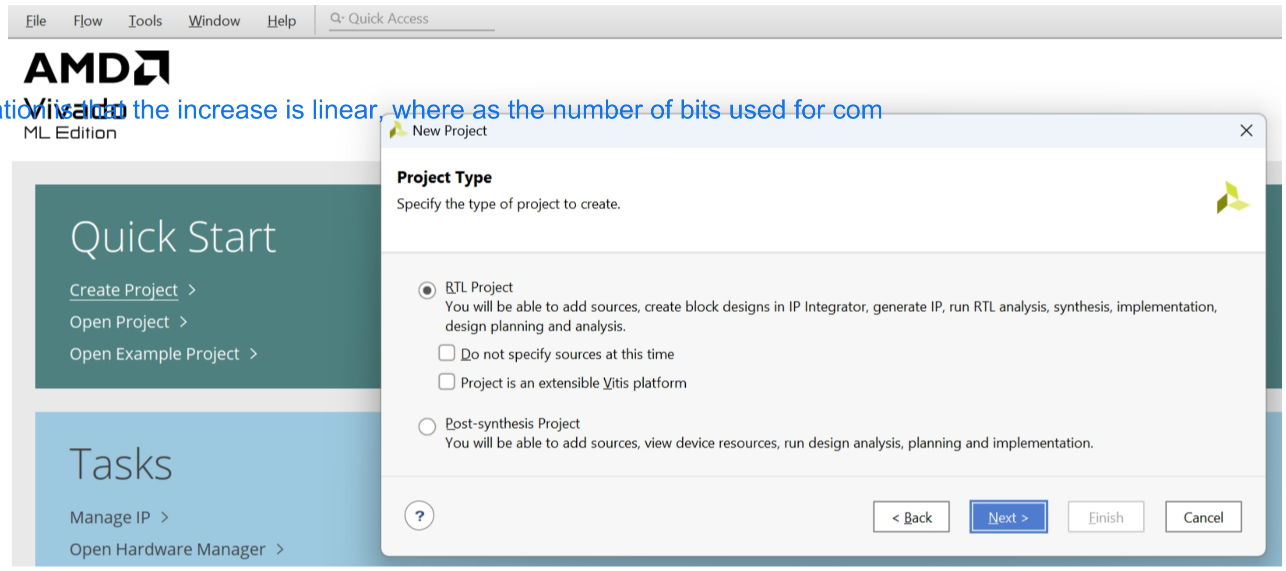
1) You will be given the Verilog codes of an adder and a multiplier. You have to generate the test bench and test whether the Verilog modules give correct results.
2) You need to take the input bit-width 8, 16, 32 and 64 and find out the number of slices used and the maximum combinational path delay of adders and multipliers of different bit-widths. You need to plot area (vs) bit-width as well as delay (vs) bit-width for the adder and the multiplier modules.

   To find out whether the design is functioning correctly you need to synthesis code and to see if that gives the correct results. Besides you need to analyze the increase in complexity (in terms of number of slices and computational delay) of adder and multiplier along with the increase in bit-width. For that you proceed as follows.

   Section A: Introduction to Vivado.

   In this section, we will introduce how to use Vivado for synthesis, simulation, testing and analysis for the **Adder** operator.

   1) Open Vivado to create a new project by clicking "Create Project" and name the project as "SC3050_lab1". Keep the default Project Type as RTL Project as shown below and click Next.



observation is that the increase is linear, where as the number of bits used for com

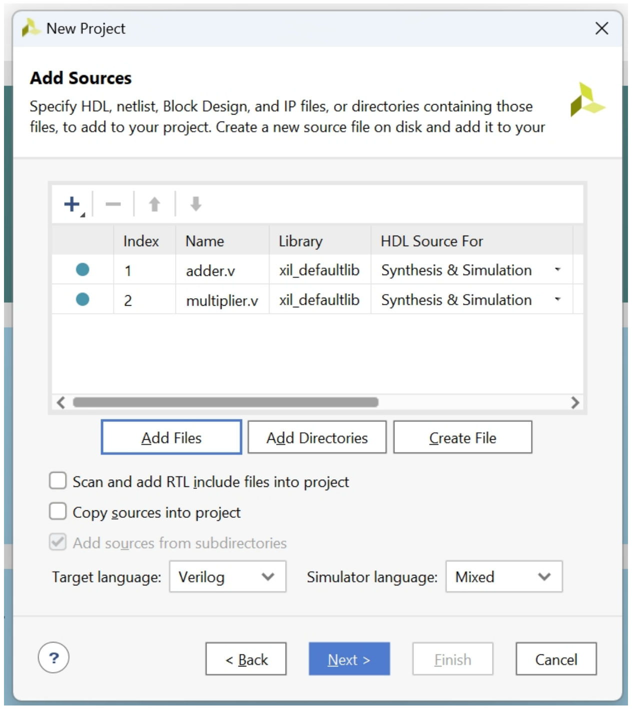
Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time
☐ Project is an extensible Vitis platform

(○) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.

   2) This will create a folder by the name "SC3050_lab1". You may click on "Add Files" the "+ symbol" to add the files "adder.v" and "multipler.v" to this folder in the "Add Source" window (Here, you are highly suggested to *copy all the project files to the project folder*, including the Verilog files (.v) and text files (.txt)):



New Project

Add Sources

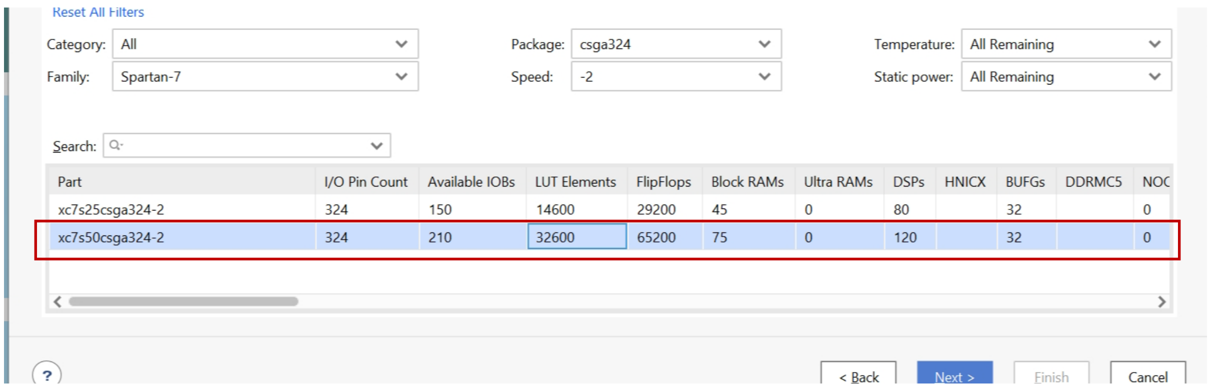
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your

	Index	Name	Library	HDL Source For
●	1	adder.v	xil_defaultlib	Synthesis & Simulation
●	2	multiplier.v	xil_defaultlib	Synthesis & Simulation

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories

Target language: Verilog　Simulator language: Mixed

3) Click Next and Add Constraints (optional) window will open. You may skip it by clicking on Next. This will bring you to choose the "Default Part" for project.
4) The project settings are as below:
   - In the field "Family": select > Spartan7
   - In the field "Package": select > csga324
   - In the field "Speed": select > -2
   - Click > Next to move to the page New Project Summary
   - Click > Finish in the page New Project Summary

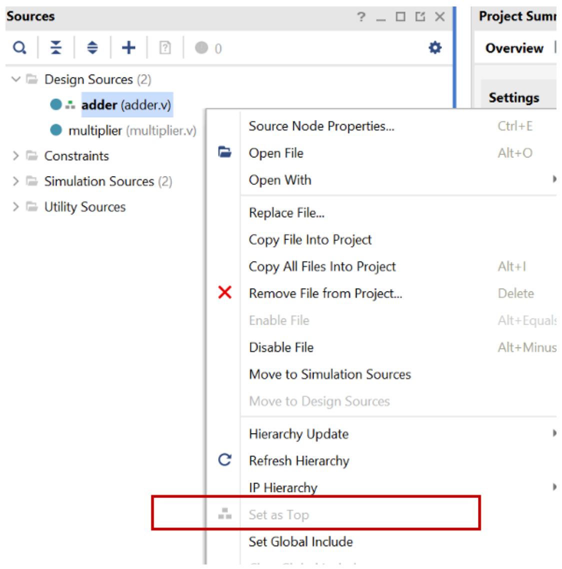


Category: All　Package: csga324　Temperature: All Remaining
Family: Spartan-7　Speed: -2　Static power: All Remaining

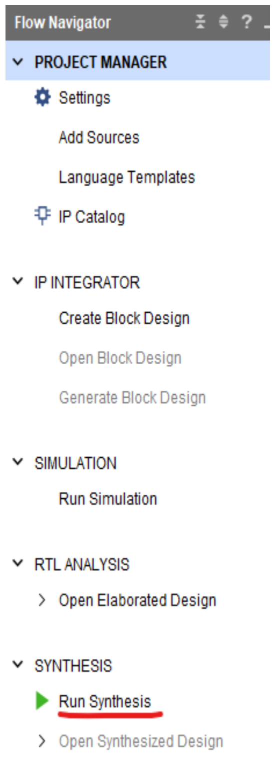
Search:

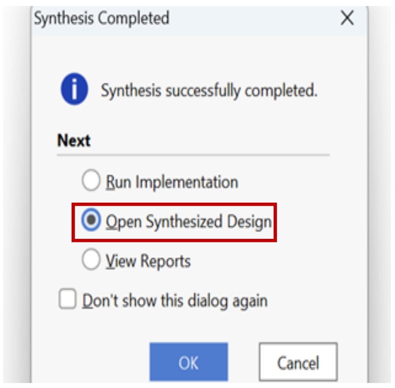
Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	HNICX	BUFGs	DDRMC5	NOC
xc7s25csga324-2	324	150	14600	29200	45	0	80		32		0
xc7s50csga324-2	324	210	32600	65200	75	0	120		32		0

5) Once the project is open, double click the adder module and go through your "adder.v" code to understand that. In order to synthesize "adder.v" right click it and select "Set as Top" to make it the top module. The tree like symbol beside "adder.v" indicates that it is the top module.



6) To synthesize the "adder.v" code and to see the schematic diagram of the adder, click on to the "Run Synthesis" tab under "SYNTHESIS" in the left most "Flow Navigator" column. This process will initiate the synthesis of the code as can be seen in the synthesis report (number of Look Up Table (LUT) slices which is a measure of area and delay of the circuit) and also will give the schematic of the code. When the synthesis is completed, select "Open Synthesized Design" to view the designs.



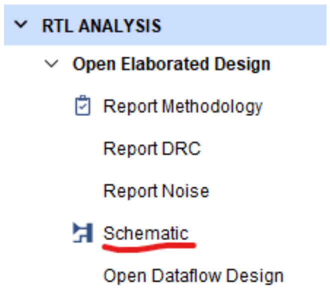

Synthesis Completed

Synthesis successfully completed.

Next

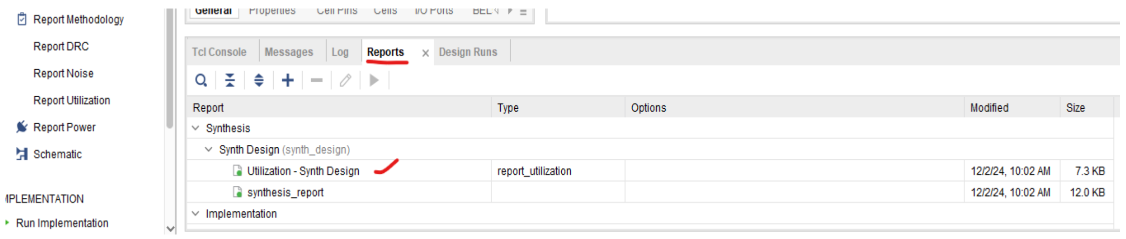
- ( ) Run Implementation
- (●) Open Synthesized Design
- ( ) View Reports
- [ ] Don't show this dialog again

7) View the RTL schematic, click on "Schematic" under "RTL ANALYSIS" in the left most "Flow Navigator" column.



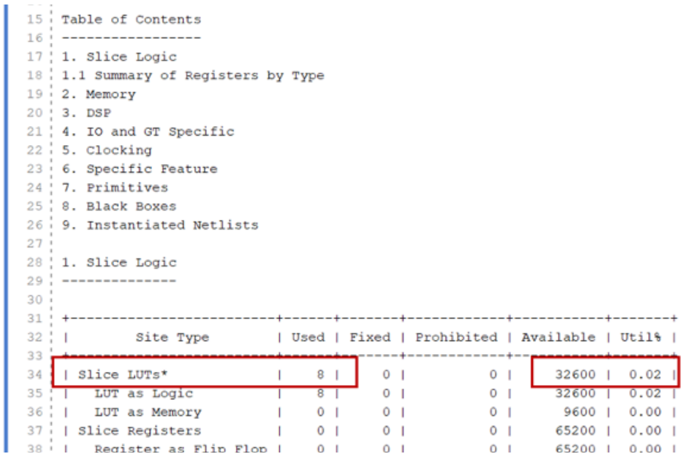
8) The schematic can be seen. Click into the main module diagram to go into the circuit diagram of adder. You can re-verify the functionality by checking the circuit diagram. You can also know the amount of hardware (LUTs) used by clicking "Schematic" tab under "SYNTHESIS" to view the technology schematic.

9) The synthesis result can be found by clicking on "Utilization – Synth Design" in the "Reports" tab at the bottom section.



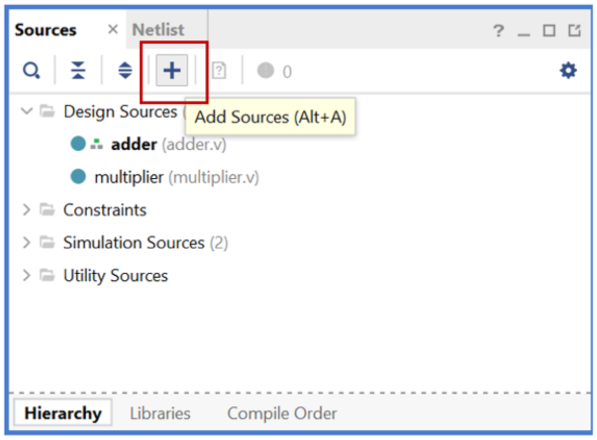
Report	Type	Options	Modified	Size
Synthesis				
Synth Design (synth_design)				
Utilization - Synth Design	report_utilization		12/2/24, 10:02 AM	7.3 KB
synthesis_report			12/2/24, 10:02 AM	12.0 KB
Implementation				

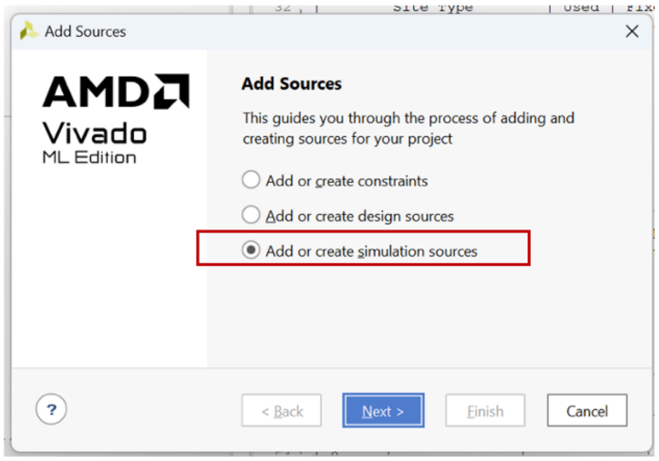
You can check the number of LUT slices used, and the corresponding utilization percentage.



```
Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |    8 |     0 |          0 |     32600 |  0.02 |
|   LUT as Logic          |    8 |     0 |          0 |     32600 |  0.02 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         |    0 |     0 |          0 |     65200 |  0.00 |
|   Register as Flip Flop |    0 |     0 |          0 |     65200 |  0.00 |
```

10) To do simulation of the code, we need a test bench. In Vivado, we can create a test bench by clicking on the "+" symbol in the "Sources" column to add sources, and then select "Add or create simulation sources" in the "Add Sources" window.



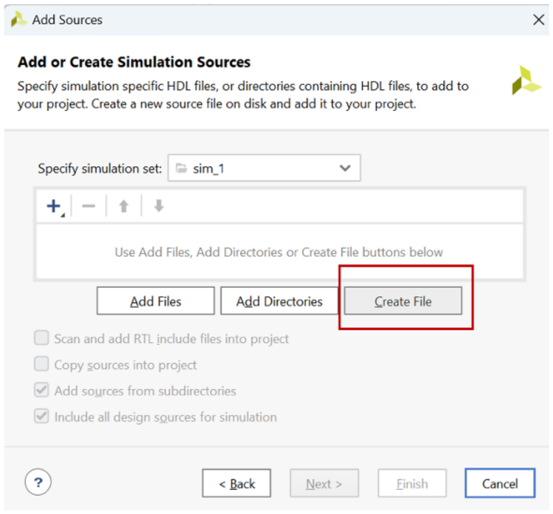


Add Sources

This guides you through the process of adding and creating sources for your project

Add or create constraints
Add or create design sources
[✓ Add or create simulation sources]

11) Select "Create File" and name the test bench as "adder_test.v" in the "Create Source File" window. Keep the file type as "Verilog" and File location as "<Local to Project>", and click on "OK" to create the test bench.



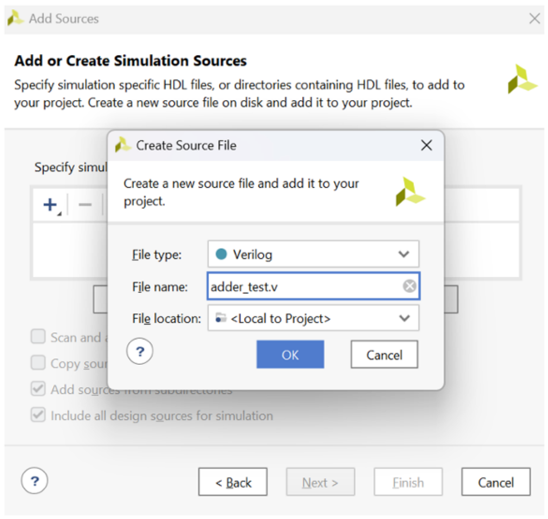
Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

Use Add Files, Add Directories or Create File buttons below

Scan and add RTL include files into project
Copy sources into project
[✓ Add sources from subdirectories]
[✓ Include all design sources for simulation]



Add or Create Simulation Sources

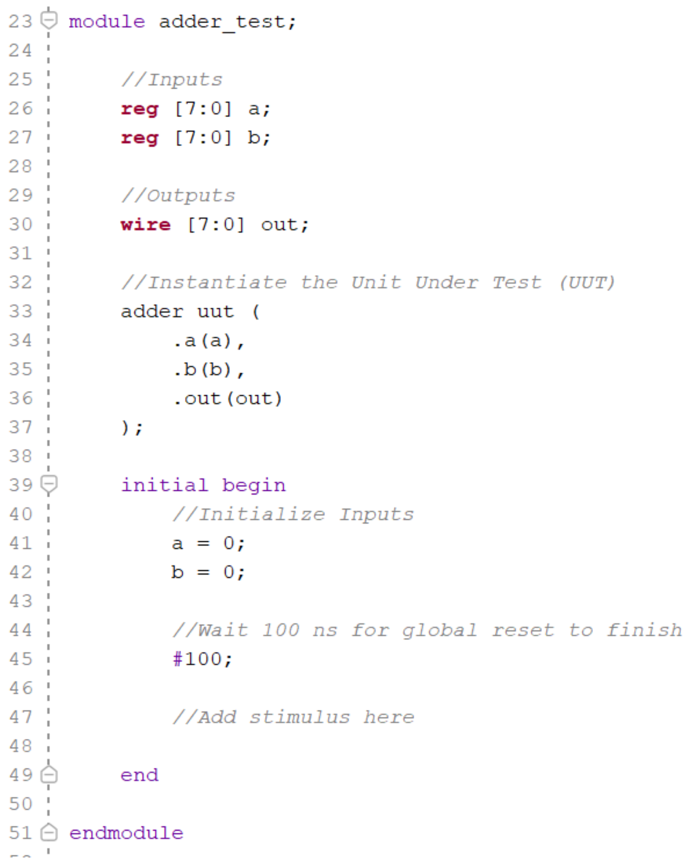
Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

File type: Verilog
File name: adder_test.v
File location: <Local to Project>

12) Once the test bench file is generated, you may double click on it and manually insert the main code (shown below) to the file. Then below the "//Add stimulus here" section, you may add the code inputs (next page, in the gray box) in the program:

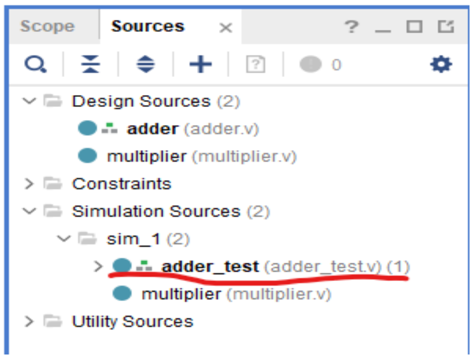


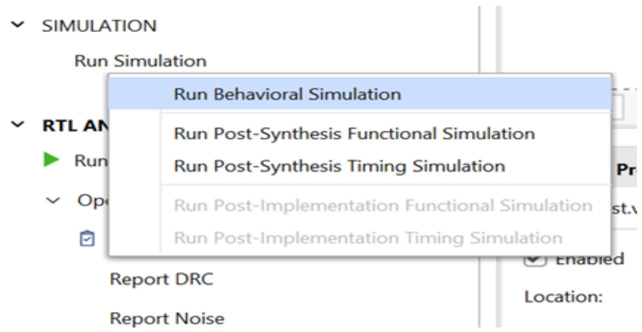
```
module adder_test;

    //Inputs
    reg [7:0] a;
    reg [7:0] b;

    //Outputs
    wire [7:0] out;

    //Instantiate the Unit Under Test (UUT)
    adder uut (
        .a(a),
        .b(b),
        .out(out)
    );

    initial begin
        //Initialize Inputs
        a = 0;
        b = 0;

        //Wait 100 ns for global reset to finish
        #100;

        //Add stimulus here

    end

endmodule
```

Verilog

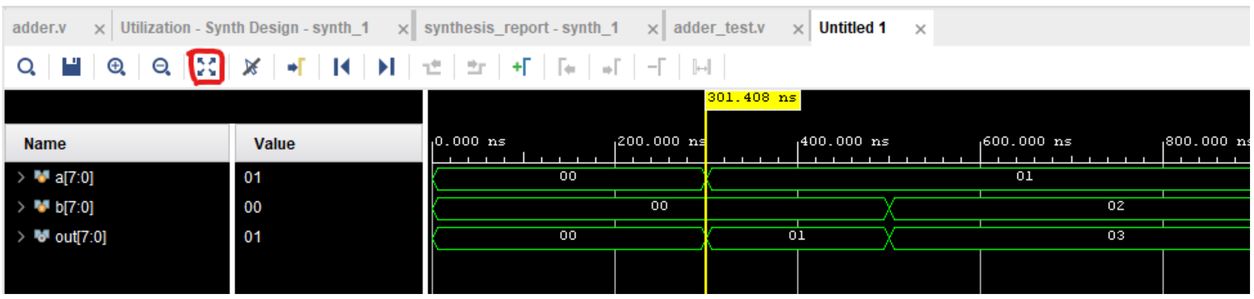
```
#200 a=8'h01; //after 200ns make a=1;
#200 b=8'h02; //after 200ns make b=2;
```

13) To do simulation of the same, select the file to be simulated and click "Run Simulation" under the "Flow Navigator ->SIMULATION" tab and select "Run Behavioral Simulation". This will generate the output waveform; you may verify whether the functionality of adder is correct.




SIMULATION
Run Simulation
Run Behavioral Simulation
Run Post-Synthesis Functional Simulation
Run Post-Synthesis Timing Simulation
Run Post-Implementation Functional Simulation
Run Post-Implementation Timing Simulation
RTL AN
Run
Op
Report DRC
Report Noise
Enabled
Location:

14) You may verify the output by clicking zoom fit (marked with a red box) on the simulation window and by seeing the value of **a** changes to 1 at 300ns and the value of **b** changes to 2 at 500ns in the waveform.



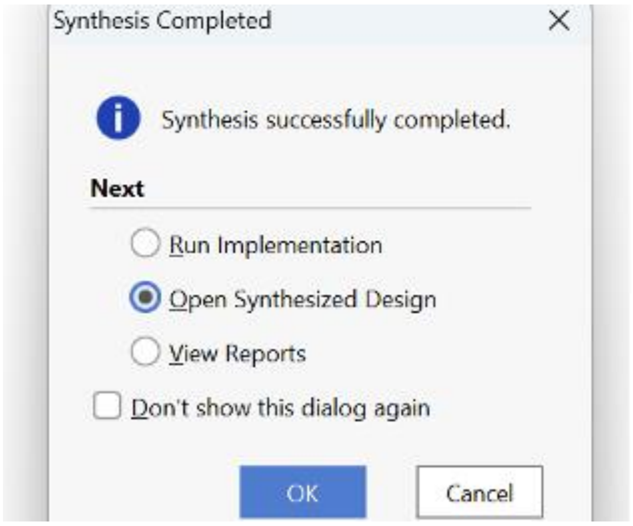
Name	Value
a[7:0]	01
b[7:0]	00
out[7:0]	01

15) Note that the underlying adder.v code was 8 bit and the simulation also was done for 8 bit adder. If we are simulating for a 16 bit adder, then the adder.v should be for 16 bit and the simulation file also should be for 16 bit adder

Section B: Experiments on Adder

The **adder** code is parametrizable. That means you can change the bit-width of the adder by changing the value for parameter **DSIZE**. If parameter **DSIZE** is set to 8, then after synthesis an eight-bit adder circuit will be created in the FPGA. If the parameter **DSIZE** =64, then the bit-width of the operands of the adder would be 64 bit.

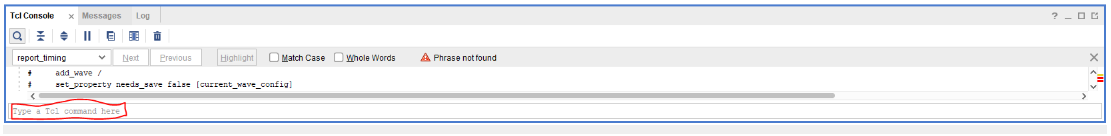
1) Set the parameter **DSIZE** 8, 16, 32 and 64 one by one and note the number of slices used and delay (in ns) in different cases. You need to synthesis each time (steps 5-13) after you change the parameter value to find out the number of slices and delay in ns. After synthesis "Open Synthesized design to update the design and its RTL and timing.


Synthesis Completed

Synthesis successfully completed.

Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the adder module.

2) To view the "Delay in ns", you may manually type command "report_timing -max_paths 1 -delay_type max" in the "TCL Console" as shown to access the "Timing Report" of each synthesis.

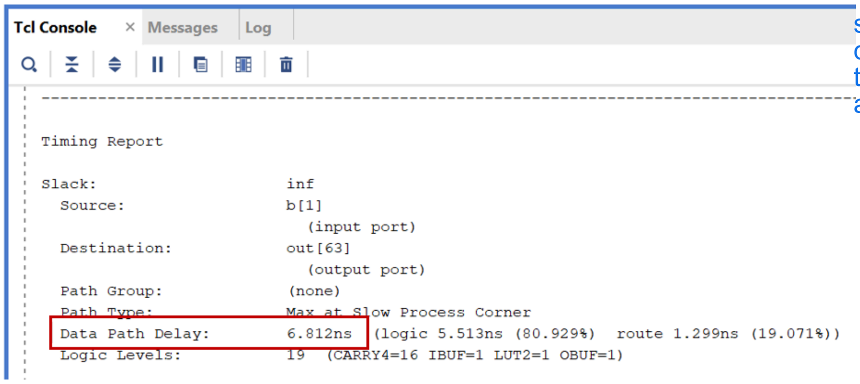


```
#     add_wave /
#     set_property needs_save false [current_wave_config]
```

After **EACH** synthesis, you need to insert and run the following command to see the timing report

```
report_timing -max_paths 1 -delay_type max
```

Then in the generated "Timing Report", you can find the value of "Delay in ns" in the "Data Path Delay" row as shown below:



```
--------------------------------------------------------------------------------

Timing Report

Slack:                    inf
  Source:                 b[1]
                            (input port)
  Destination:            out[63]
                            (output port)
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        6.812ns  (logic 5.513ns (80.929%)  route 1.299ns (19.071%))
  Logic Levels:           19  (CARRY4=16 IBUF=1 LUT2=1 OBUF=1)
```

so delay in this context refers to the data path delay which is also related to LUT. since LUT determines the area and delay of the circuit.

3) To synthesize and to see its circuit diagram you need to make it as the top module and **repeat from steps 5-13.** You may have to provide input values of proper bit-width to the testbench program for adder in order to verify the same. Note that the bit-width of the inputs is same as the bit-width parameter that you have set in the code. To view the technology schematic, follow the **step 8**, and click 'Schematic' under "RTL ANALYSIS".

Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8			
16	16			
32	32			
64	64			

Table 1: Slices and delay for adder

(Please make sure to implement all the experiments and fill out Table 1 in the last page of this manual. You'll need to submit the outcomes at the end of this lab.)

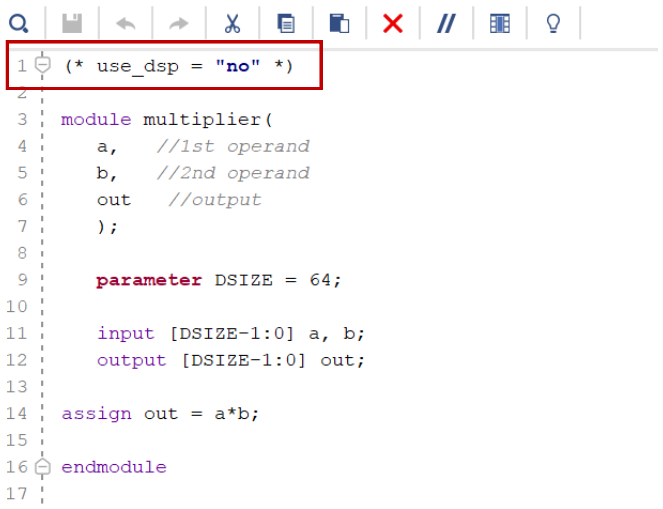
Section C: Experiments on Multiplier

The given **Multiplier** code is also parametrizable. That means you can change the bit-width of the operands of multiplier by changing the value for parameter **DSIZE**; If parameter **DSIZE**=8, then the bit-width of the operands in multiplier is 8 bit. If parameter **DSIZE**=64, then the bit-width of the operands of the multiplier would be 64 bit.

1) Set the parameter DSIZE 8, 16, 32 and 64(one by one) and note the number of slices occupied and delay in terms of (ns) for all cases. You need to synthesis each time (steps 5-13) after you change the parameter value to find out the number of slices and delay in ns. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the multiplier module.
2) To view the "Delay in ns", you may manually type command in the "TCL Console" to access the "Timing Report" of each synthesis. After **EACH** synthesis, you may insert and run the following command.

```
report_timing -max_paths 1 -delay_type max
```

3) To synthesize and to see its circuit diagram **set it as the top module** (indicated in step 5) and repeat from steps 5-13. You have to add respective input values to the testbench program for multiplier in order to verify the same. Note that the bit-width of the inputs is the same as the bit-width parameter that you set in the code. To view the technology schematic and the increased bit-width of the multiplier, follow the step 8, and click 'Schematic' under "RTL ANALYSIS".
4) Before doing step 6 (the synthesis), you may double click on the "multiplier.v" to view the code, make sure the top line looks like the following:

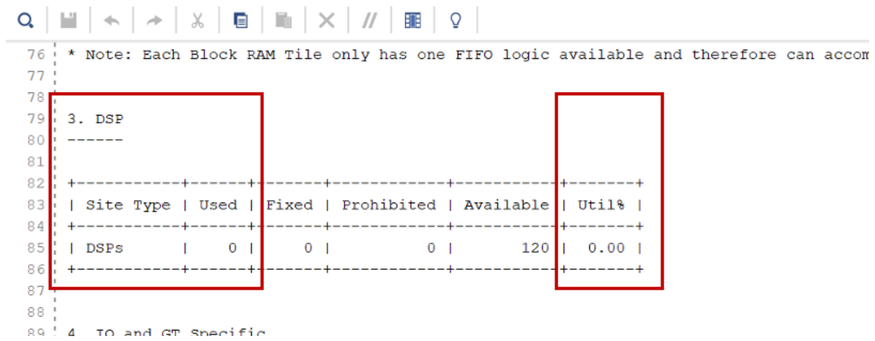


```
(* use_dsp = "no" *)

module multiplier(
    a,    //1st operand
    b,    //2nd operand
    out    //output
    );

    parameter DSIZE = 64;

    input [DSIZE-1:0] a, b;
    output [DSIZE-1:0] out;

assign out = a*b;

endmodule
```

In Vivado, inserting the following code snippet at the top line of a Verilog file can avoid the usage of DSP.

```
(* use_dsp = "no" *)
```

You may also double check the usage of DSP after you synthesize the code. Click on “Utilization – Synth Design” in the “Reports” tab at the bottom section and navigate to the “DSP” section, make sure the DSP usage is 0.



```
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accom


3. DSP
------

+-----------+------+-------+------------+-----------+-------+
| Site Type | Used | Fixed | Prohibited | Available | Util% |
+-----------+------+-------+------------+-----------+-------+
| DSPs      |    0 |     0 |          0 |       120 |  0.00 |
+-----------+------+-------+------------+-----------+-------+


4. IO and GT Specific
```

5) Continue with the rest of the steps as earlier.

Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8			
16	16			
32	32			
64	64			

Table 2: Slices and delay for multiplier

(**Please make sure to implement all the experiments and fill out Table 2 in the last page of this manual. You’ll need to submit the outcomes at the end of this lab.**)

PART II: ARITHMETIC LOGIC UNIT(ALU)

ARITHMETIC LOGIC UNIT (ALU) SPECIFICATIONS

In this part of the Lab, we consider a simple ALU that performs the computation for eight arithmetic and logical operation. The seven operations are: ADD, SUB, AND, XOR, COM, MUL and ADDI as described in Table 1.

Table 1 - Description of ALU Operations

Instruction	Equation	Operation	Description
ADD	A+B	Addition	Addition of A and B, where both A and B are in 2's complement representation
SUB	A-B	Subtraction	Subtraction of A and B, where both A and B are in 2's complement representation
AND	A&B	Logical AND	Bit-wise AND of A, B
XOR	A^B	Logical XOR	Bit-wise XOR of A, B
ORR	A|B	Logical OR	Bit-wise OR of A, B
PassB	B	Only allow B to pass to output	Used for conditional branch; whether to check whether the content is equal to zero or not

A and B are the data input ports of the ALU to feed maximum of two operands to the ALU. The ALU has 3-bit control input to perform one out of the 6 possible instructions which the ALU can perform. The information is also listed in Table 2. The encoding of the 6 instructions is listed in Table 3.

Table 2 - Port List Specification. The bit-width to be varied from 8 bit to 64 bit.

Port Name	Port Direction	Description
A	Input	First operand
B	Input	Second operand
op	Input	Indicates the ALU about the operation to be performed
Out	Output	Output of the operation

Table 3 - ALU operation encoding

Operation	'op' value
ADD	000
SUB	001
AND	010
XOR	011
ORR	100
PassB	101

ALU IMPLEMENTATION, TESTING AND ANALYSIS

For this assignment,

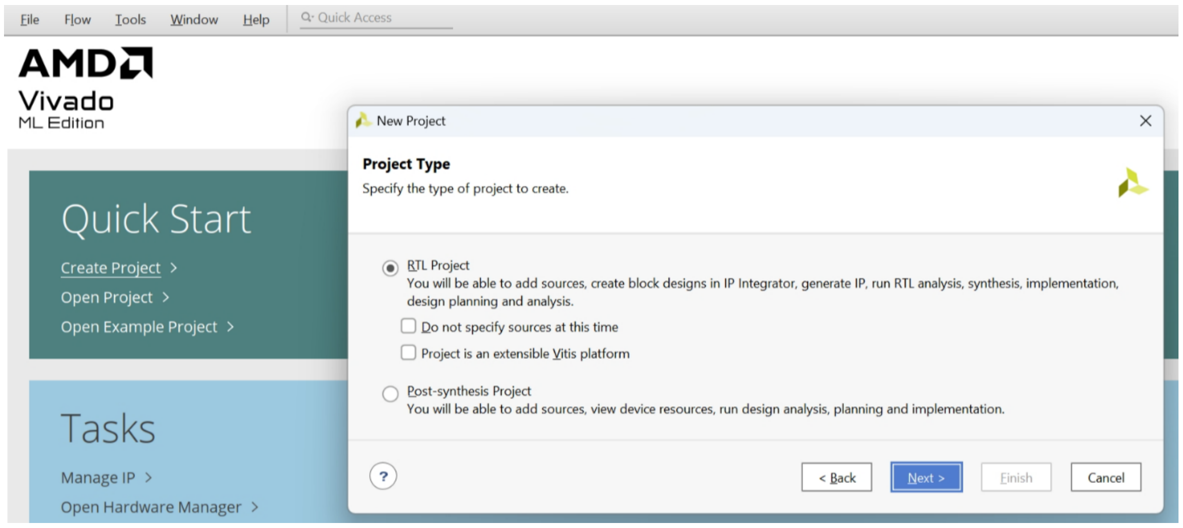
a) You are given the Verilog code as well as the test bench for ALU. You have to test whether the ALU gives correct results.
b) You need to set the input bit-width to 8, 16, 32 and 64 (one by one) and find out the number of slices used and the maximum combinational path delay in each case. You need to plot area (vs) bit-width as well as delay (vs) bit-width for the ALU module. In FPGA you cannot find area directly so instead of area you can take number of slices, which would be considered proportional to the area.

DETAILS

Section A: ALU Vivado Project

The given ALU code is parametrizable but initialized to 8 bits. To find out whether the design is functioning correctly you need to synthesis code and to see if that gives the correct results. For that you can proceed as follows.

1) Open Vivado to create a project by clicking "Create Project" and name the project as "SC3050_lab1_alu". Keep the default Project Type as below:



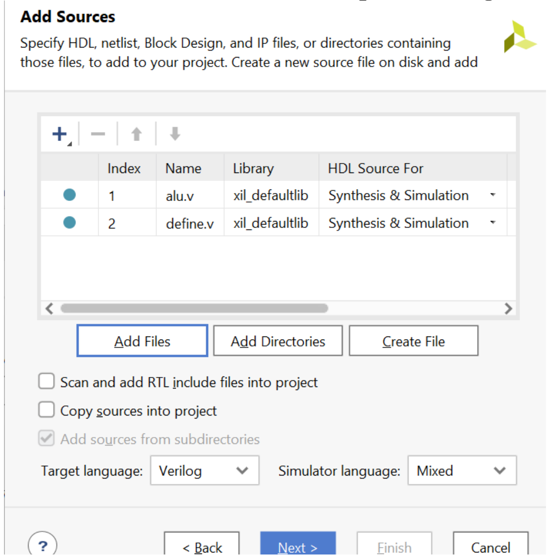
Project Type

Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time
☐ Project is an extensible Vitis platform

( ) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.

2) This will create a folder by the name "SC3050_lab1_alu". You may add the files 'alu.v', 'define.v' to this folder in the "Add Source" window for now. (Here, you are highly suggested to *copy all the project files to the folder*, including the Verilog files (.v) and text files (.txt) if needed) The description of all the files are: The "alu.v" file has the code for ALU and the "define.v" file has the user variables in "alu.v". The "alutest.v" provides the input test vectors listed in "input.txt" to test the code in "alu.v". The output is being written to "output.txt".



Add Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add

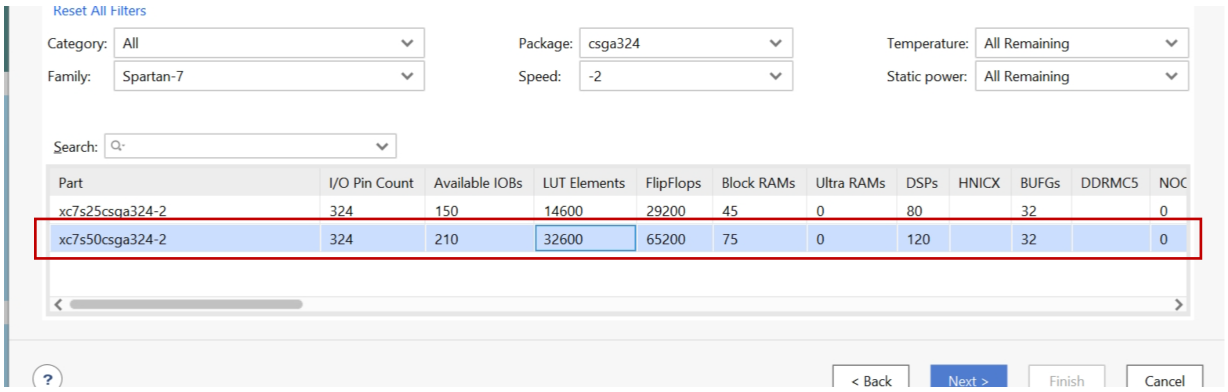
	Index	Name	Library	HDL Source For
●	1	alu.v	xil_defaultlib	Synthesis & Simulation
●	2	define.v	xil_defaultlib	Synthesis & Simulation

☐ Scan and add RTL include files into project
☐ Copy sources into project
☑ Add sources from subdirectories

Target language: Verilog   Simulator language: Mixed

3) In the next Add Constraints (optional) window, you may skip it by clicking on Next.

4) The project settings are as below:
   - In the field Family: select > Spartan7
   - In the field Package: select > csga324
   - In the field Speed: select > -2
   - Click > Next to move to the page New Project Summary
   - Click > Finish in the page New Project Summary



Reset All Filters

Category: All
Family: Spartan-7
Package: csga324
Speed: -2
Temperature: All Remaining
Static power: All Remaining

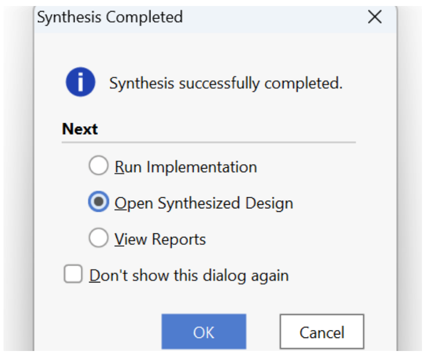
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	HNICX	BUFGs	DDRMC5	NOC
xc7s25csga324-2	324	150	14600	29200	45	0	80		32		0
xc7s50csga324-2	324	210	32600	65200	75	0	120		32		0

5) Once the project is open, double click the alu module and go through your "alu.v" code to understand the functionalities provided by the given ALU and how it is achieved. Before synthesizing the code, make sure the following code is already added at the top line to avoid the usage of DSP.

```
(* use_dsp = "no" *)
```

6) To synthesize the "alu.v" code and to see the schematic diagram of it, make it "top module" and click on to the "Run Synthesis" tab under "SYNTHESIS" in the left most "Flow Navigator" column. This process will initiate the synthesis of the code as can be seen in synthesis report (number of Look Up Table (LUT) slices which is a measure of area and delay of the circuit) and also will give the schematic of the code. When the synthesis is completed, select "Open Synthesized Design" to view the designs.



Synthesis Completed

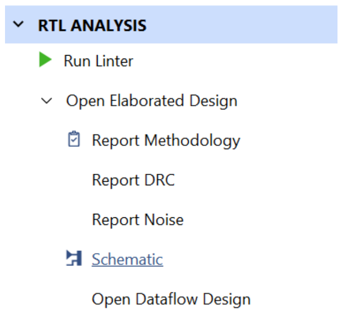
Synthesis successfully completed.

Next

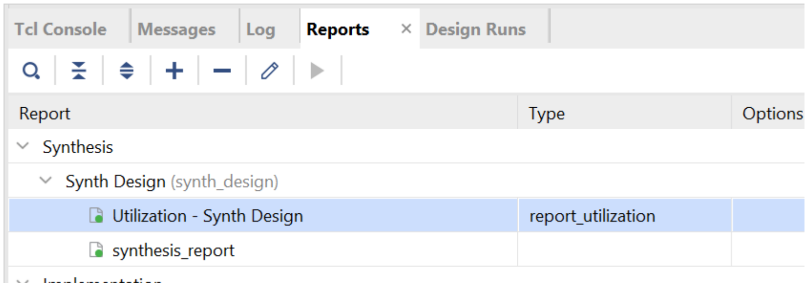
( ) Run Implementation
(●) Open Synthesized Design
( ) View Reports

☐ Don't show this dialog again

7) To view the RTL schematic, click on "Schematic" under "RTL ANALYSIS" in the left most "Flow Navigator" column.

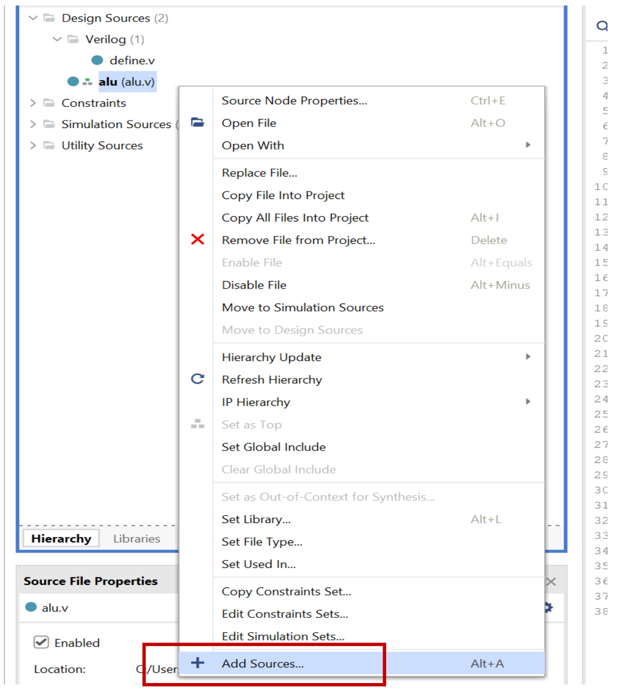


8) The schematic diagram can be seen. Click into the main module diagram to go into the circuit diagram of multiplier. You can re-verify the functionality by checking the circuit diagram. You can also know the amount of hardware (LUTs) used by clicking "Schematic" tab under "SYNTHESIS" to view the technology schematic.
9) The synthesis result can be found by clicking on "Utilization – Synth Design" in the "Reports" tab at the bottom section.



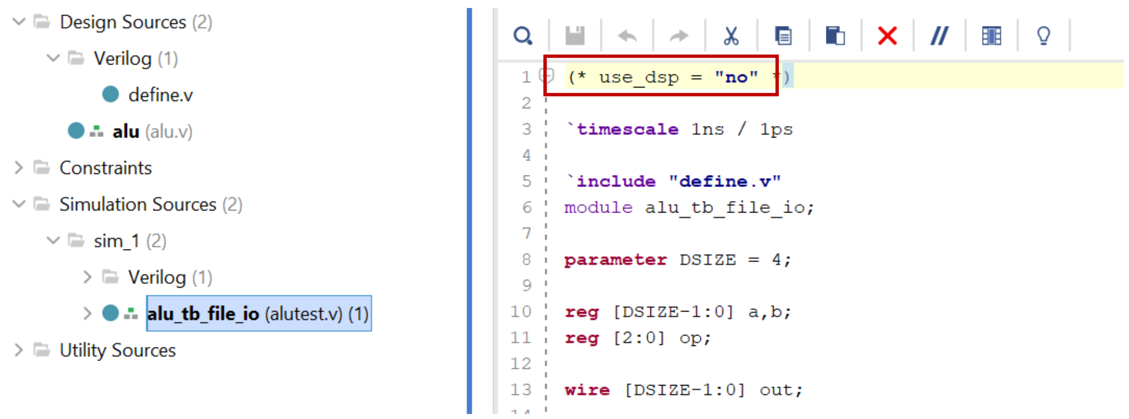
Report	Type	Options
Synthesis		
Synth Design (synth_design)		
Utilization - Synth Design	report_utilization	
synthesis_report		

10) To do simulation of the code we need a test bench. We can add the test bench to alu.v by right clicking the file and select "Add Sources".



11) In the “Add Sources” window, select “Add or create simulation sources”. In the next window, select “Add Files” to add the given “alutest.v” file to your folder.

12) After the test bench is added, click on “Simulation Sources” and double click on “alu_tb_file_io(alutest.v)” to read the code and make sure the DSP usage is disabled.



```
(* use_dsp = "no" *)

`timescale 1ns / 1ps

`include "define.v"
module alu_tb_file_io;

parameter DSIZE = 4;

reg [DSIZE-1:0] a,b;
reg [2:0] op;

wire [DSIZE-1:0] out;
```

13) To do simulation and test the ALU, click “Run Simulation” under the “SIMULATION” tab and select “Run Behavioral Simulation”. **The ‘alutest.v’ is initialized to bit-width 4, but can be changed to different bit-widths. Note that you need to change the values in ‘define.v’ accordingly. As the DSIZE of test bench is 4, the DSIZE in “define.v” should be 4.**

14) Copy the “input.txt” and paste the same into “ ... \SC3050_lab1_alu.sim\sim_1\behav\xsim”, under SC3050_lab1_alu folder. The simulator will open, but as we had given the input as text file and has programmed the ‘alutest.v’ to give the output as text file, you can go and check the ‘output.txt’ file in the folder “SC3050_lab1_alu\SC3050_lab1_alu.sim\sim_1\behav\xsim”. You can see the results of the inputs that you have given.

<u>Section B: Experiments on ALU</u>

The ALU code is parametrizable, so we can change the bit-width of the ALU by changing the value for parameter DSIZE: If parameter DSIZE=8, then the bit-width of the operands of ALU is 8 bit. If parameter DSIZE=8, then the bit-width of the operands of ALU is 8 bit.

1) Change the parameter DSIZE from 8, 16, 32 and 64 (one by one) and note the number of slices occupied and delay in terms of (ns).

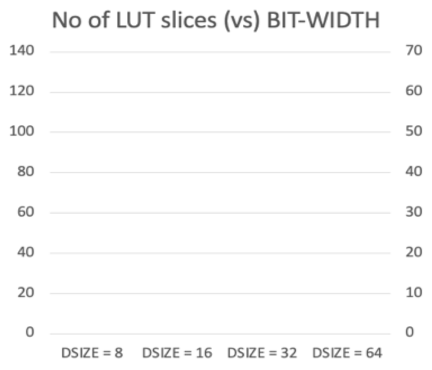
Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8			
16	16			
32	32			
64	64			

Table 3: Slices and delay for ALU

(**Please make sure to implement all the experiments and fill out Table 3 in the last page of this manual. You'll need to submit the outcomes at the end of this lab**.)

CHALLENGE (Optional)

1) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the adder module for DSIZE=8, 16, 32 and 64.
2) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the multiplier module for DSIZE =8, 16, 32 and 64.
3) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the ALU module for DSIZE=8, 16, 32 and 64.



Delay (vs) BIT-WIDTH
8
7
6
5
4
3
2
1
0
70
60
50
40
30
20
10
0
DSIZE = 8
DSIZE = 16
DSIZE = 32
DSIZE = 64

Lab Report for Lab 1 (Name: Ng Tze Kean, Matric: U2121193J, Group: TEL2)

EVALUATION

Please fill the three tables based on the experiments.

Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8	8	0.02	5.440
16	16	16	0.05	5.636
32	32	32	0.10	6.028
64	64	64	0.20	6.812

Table 1: Slices and delay for adder

Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8	31	0.10	7.168
16	16	146	0.45	9.128
32	32	634	1.94	10.223
64	64	2162	6.63	12.643

Table 2: Slices and delay for multiplier

Parameter: **DSIZE**	BIT-WIDTH	No of LUT slices	LUT Util %	Delay in ns
8	8			
16	16			
32	32			
64	64			

Table 3: Slices and delay for ALU

CHALLENGE (Optional, no marks)

4) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the adder module for DSIZE=8, 16, 32 and 64.
5) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the multiplier module for DSIZE =8, 16, 32 and 64.
6) Plot a graph on the number of LUT slices (vs) bit-width as well as delay (vs) bit-width for the ALU module for DSIZE=8, 16, 32 and 64.