DIGITAL SYSTEM DESIGN (UE20EC313)

Cadence Lab manual and Additional Hands-on Projects (AHP) reference

(QR code to the read-only version of the Cadence tools commands)

Index

1. Cadence tools command reference

Link to this document: *http://bit.ly/cadencelabpesu*

1.1 Getting Started on Linux for Cadence Lab

> create a folder in the desktop, with your srn/name
> open the folder
> right-click and create files for design and testbench,
eg. db_fsm.v and db_tb.v
> right-click on the files and open them using *gedit*, save the design and testbench codes in the respective files
> right-click inside the folder and select *open in terminal*
> enter the following commands in the terminal

csh

Enters the C-Shell

source /home/<install location>/cshrc

Navigates to the Cadence Tools install path and starts the tool

Note: You can use the upper arrow in the terminal to navigate quickly to the already used paths/commands and use tab-key to auto-complete commands.

> A new window appears that welcomes the user to the Cadence Design Suite, the following tools can be invoked in this window.

1.2 Simulation Tool (*ncverilog*)

>To start reading the design and testbench files, to obtain a waveform in the Graphical User Interface (*simvision),* enter the following commands.
Note: No space between +access and +rw, but mandatory space between +rw and +gui. (make sure to follow all similar spacing patterns given in the tool reference)

ncverilog <design> <testbench> +access+rw +gui

eg. ncverilog db_fsm.v db_tb.v +access+rw +gui
Note: the +gui starts up the *ncverilog* GUI window.

> navigate through the design hierarchy and select the signals you want to analyze in the design browser (hold down ctrl-key while selecting), right-click and select *send to waveform*
> in the *simvision* window, select the play button, followed by the pause button to start and stop the simulation. The simulation will end automatically if the $finish statement is executed in the HDL.
> select the '=' symbol at the top right corner of the window, to fit the waveform's entirety in the same frame.
> drag the red marker to the beginning of the waveform and select on the '+' symbol on the top right corner, to magnify until the waveform pulses are visible for verifying the functionality of the design.

1.3 Checking Code Coverage in ncverilog using IMC (*Incisive Metrics Center*)

```
ncverilog design.v tb.v +access+rw +gui +nccoverage+all
```

> Check for the path of the file "cov_work" generated in the terminal then type:
(Invoke Incisive Metrics Center)
>enter the command 'imc' in the terminal which will launch the IMC GUI.

```
imc
```

> In he IMC's Graphical User Interface, you can navigate and select the file to check the Code Coverage (block, branch, expression, toggle) and FSM Coverage, represented in percentages.

1.4 Synthesis Tool (*genus)*

genus -gui

Opens the genus tool with gui, alternatively you can show and hide gui using command gui_show and gui_hide

read_libs <*path of .lib file*>

Reads library file for synthesis, from the specified path. Eg. saed90nm_typ.lib the 90 nanometer typical library

read_hdl <*path of design file*>

Reads design file to be synthesized, written in HDL (eg. verilog, systemverilog)

elaborate

Elaborates the design in the tool, and can be viewed in the GUI by selecting Hier Cell > Schematic View(Module) > in New.

For Synthesis with constraints

read_sdc <path of .sdc constraints file>

syn_generic

Synthesizes the design to the G Tech cells (default cells for the Cadence Tool)

syn_map

maps the synthesized cells to the library specified earlier in read_libs command

syn_opt -incremental

Incrementally optimizes the synthesized design

report_timing > (path for .rpt file to save timing report)

Reports timing Time Borrowed, Uncertainty, Required Time, Launch Clock, Data Path and Slack.

report_area > (path for .rpt file to save area report)

Reports area of the synthesized design in micro-meters-square

report_power > (path for .rpt file to save power report)

Reports power in nano Watts (nW)

write_hdl > (path for .v file for netlist to be written)

Writes the netlist in HDL format in the path specified

quit

Exits the genus tool

1.5 Syntax for writing the Constraints file (.sdc)

Creating Clock “clk”

```
set EXTCLK "clk" ;
set_units -time <units eg.1.0ns> ;
set EXTCLK_PERIOD <time period eg. 20.0>;
create_clock -name "$EXTCLK" -period  "$EXTCLK_PERIOD"  -waveform "0
[expr $EXTCLK_PERIOD/2]" [get_ports clk]
```

Setting clock skew

```
set SKEW <skew value eg. 0.200>
set_clock_uncertainty $SKEW [get_clocks $EXTCLK]
```

Setting source rise and fall latency

```
set SRLATENCY <source rise latency eg. 0.80>
set SFLATENCY <source fall latency eg. 0.75>
```

Setting Rise times and Fall Times

```
set MINRISE <minimum rise time eg. 0.20>
set MAXRISE <maximum rise time eg. 0.25>
set MINFALL <minimum fall time eg. 0.15>
set MAXFALL <maximum fall time eg. 0.10>
set_clock_transition -rise -min  $MINRISE [get_clocks $EXTCLK]
set_clock_transition -rise -max  $MAXRISE [get_clocks $EXTCLK]
set_clock_transition -fall -min  $MINFALL [get_clocks $EXTCLK]
set_clock_transition -fall -max  $MAXFALL [get_clocks $EXTCLK]
```

Setting Input and Output Delays for ports in the design

```
set INPUT_DELAY <input delay value eg. 0.5>
set OUTPUT_DELAY <output delay value eg. 0.5>

set_input_delay -clock [get_clocks $EXTCLK] -add_delay 0.3 [get_ports <input
port name>]
set_output_delay -clock [get_clocks $EXTCLK] -add_delay 0.3 [get_ports
<output port name>]
```

Note: the port names must correlate with the names used in the modules, and can be validated by using the report_timing command while synthesizing with constraints in the Cadence Genus tool

---

1.6 Equivalence Checking (*conformal LEC*)

Opening the Cadence Conformal Logical Equivalence Checking tool

```
lec -LPGXL
```

Note: The GUI can be used alternative to entering the commands

Reading Library

GUI: file -> Read library -> choose the library's .v file
Command Line: read library -Both -Replace -sensitive -Verilog technology_file.v -nooptimize

Reading Verilog file (GOLDEN)

GUI: file -> Read design for RTL(Verilog code) -Golden -> choose the RTL verilog code
Command Line: read design design.v -Verilog -Golden -sensitive -continuousassignment Bidirectional -nokeep_unreach -nosupply

Reading Netlist file (REVISED)

GUI: file -> Read design for RTL netlist or other code we want to compare with -Revised
Command Line: read design design2_or_netlist.v -Verilog -Revised -sensitive -continuousassignment Bidirectional -nokeep_unreach -nosupply

Set mode:

```
set system mode lec
```

Add points at which equivalence should be checked(Here we check all points):

```
add compared points -all
```

Run the comparison:

```
compare
```

The Equivalent and Non-Equivalent points are displayed in the GUI window, and can Non Equivalent modules/ sub-modules can be viewed by clicking on the '*Non-Equivalent*' label

---

1.7 Linting (*irun)*

To run linting on the design file

```
irun -superLint <path of the HDL file>
```

The rtlchecks.log and modelchecks.log files are generated in your present working directory.

The tool uses automated structural analysis on the design so that it complies with design coding rules that prevent synthesis issues & functional bugs and enforce coding styles for readability & re-use. These rtlchecks.log and modelchecks.log files present the *N - notes, *W - warnings and *E - errors for the design.

1.8 Static Timing Analysis (*Tempus)*

Invoke the Tempus tool and open GUI

```
tempus
```

Alternatively: Invoke Tempus without GUI

```
tempus -no_gui
```

Read library used for synthesis:

```
read_lib library_file.lib
```

Read synthesized gate level netlist:

```
read_verilog netlist.v
```

Set the top module in your design:

```
set_top_module top
```

Read the constraints file:

```
read_sdc constraints.sdc
```

Setting things to check: Signal integrity check

```
set_delay_cal_mode -siAware true
set_si_mode -enable_delay_report true
set_si_mode -enable_glitch_report true
set_si_mode -enable_glitch_propagation true
```

Update the settings to report all timing parameters

```
update_timing -full
```

Report timing: (general report with parameters like slack and critical path)

```
report_timing
```

Report slack alone:

```
report_slack
```

```
report_timing -path_type summary_slack_only
```

Report clocks:

```
report_clocks
```

Analysis report generation:

```
report_analysis_coverage
```

1.9 Additional References

Changing power engine from joules to legacy in Genus Tool

```
set_db power_engine legacy
```

Sending files from one system to another in cadence lab

```
scp <filename> <sender's username>@<ipaddress of reciever>:<~ or receiving path
>

Eg. scp /home/cmos/Desktop/DSD_lab.zip @10.3.32.69:</home/cmos/Desktop>
```

Incisive Formal Verifier(IFV):

IFV tool can either read scripts(f file) or directly read from .sv file. Optionally a bind file can be created to link separate design(.sv) file and assertions file(.sva).

Run basic:
ifv example.sv
Will display in the command prompt if the assertions failed or passed.
ifv example.sv +gui
-> Click on the run button on the top right corner of the tool bar.
-> Right click on "Pass" or "Failed" to look for traces where it passed or failed.
Will open simvision to display the traces.
Exit

2. AHP Project A – Bitonic Sorting Network

2.1. Week 1 – CAE block

Designing a two-input compare and exchange (CAE) block, with the given specifications.

DATA_WIDTH 32 bits

Description: Design a basic CAE (Compare and Exchange) block which compares two inputs A(i0) and B(i1) based on the direction (dir) and provides the outputs o1 and o2. The design must include a clock, asynchronous reset and should operate only when the enable signal is set high.

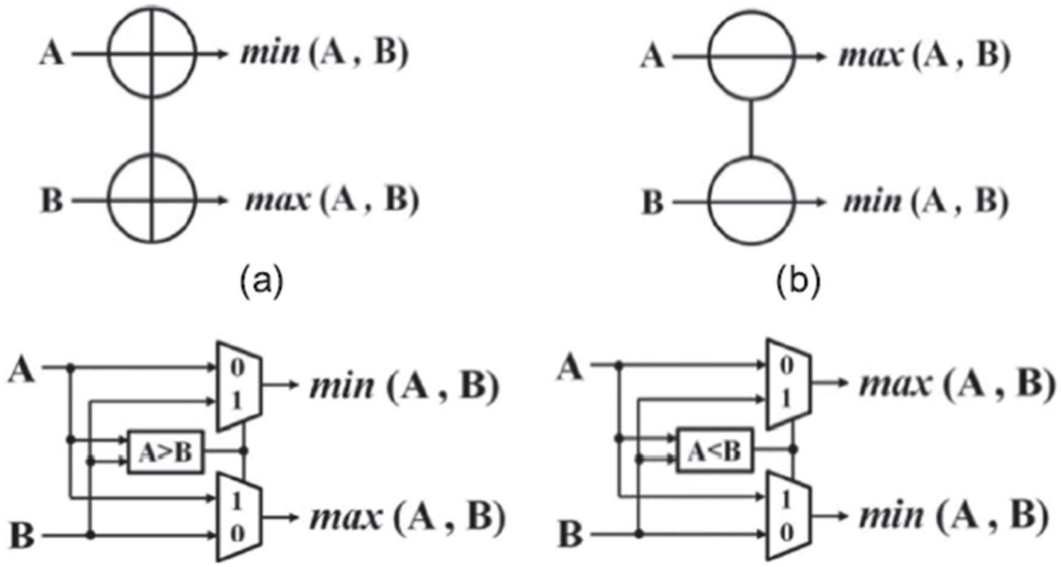


figure (a) represents the CAE block for direction 0 – ascending order sorting, and figure (b) represents the direction 1 – descending order sorting.

Design ports specification:

Signal Name	Direction	Bit Width	Description
i1	input	32	Input data 1 to be compared
i2	input	32	Input data 2 to be compared
dir	input	1	Direction bit 0- specifies ascending order of sorting, 1 specifies descending order
clk	input	1	Clock input to the design
reset	input	1	Reset signal sets the output to zero
enable	input	1	Data is fed in the design only when the input enable signal is high
o1	output	32	Output data 1 of the CAE
o2	output	32	Output data 2 of the CAE

2.2. Week 2 – Design a 4 input Bitonic Sorting unit (BM4) using CAE blocks

Design a 4 input Bitonic Sorting unit (BM4) using CAE blocks

DATA_WIDTH 32 bits

Description: Design a four input bitonic unit with inputs in0, in1, in2, in3 based on the direction (dir) and provides the outputs o1, o2, o3, o4.
The design must include a clock, asynchronous reset and should operate only when the enable signal is set high.

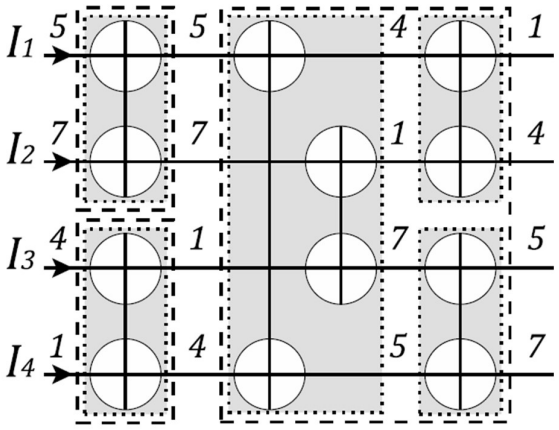


figure above represents the bitonic BM4 unit for ascending order sorting taken for inputs 5,7,4,1 sorted finally in ascending order 1,4,5,7

Design ports specification:

Signal Name	Direction	Bit Width	Description
in1	input	32	Input data 1 to be compared
in2	input	32	Input data 2 to be compared
in3	input	32	Input data 3 to be compared
in4	input	32	Input data 4 to be compared
dir	input	1	Direction bit 0- specifies ascending order of sorting, 1 specifies descending order
clk	input	1	Clock input to the design
reset	input	1	Reset signal sets the output to zero
enable	input	1	Data is fed in the design only when the input enable signal is high
o1	output	32	Output data 1 of the BM4
o2	output	32	Output data 2 of the BM4
o3	output	32	Output data 3 of the BM4
o4	output	32	Output data 4 of the BM4

2.3. Week 3 - Design a BM8 unit for fully sorting two sets of four pre-sorted inputs

Design a BM8 unit for fully sorting two sets of four pre-sorted inputs

DATA_WIDTH 32 bits

Description: Design an eight input BM8 unit, which sorts two sets of four pre-sorted inputs into a sequence of 8 sorted records.
Note: in the given diagram below, the two sets of ascending sorted inputs are {1,4,5,7} and {0,2,3,6} which are pre sorted pairs, and will be sorted ascendingly at the end of the BM8 block.

The design must include a clock, asynchronous reset and should operate only when the enable signal is set high.

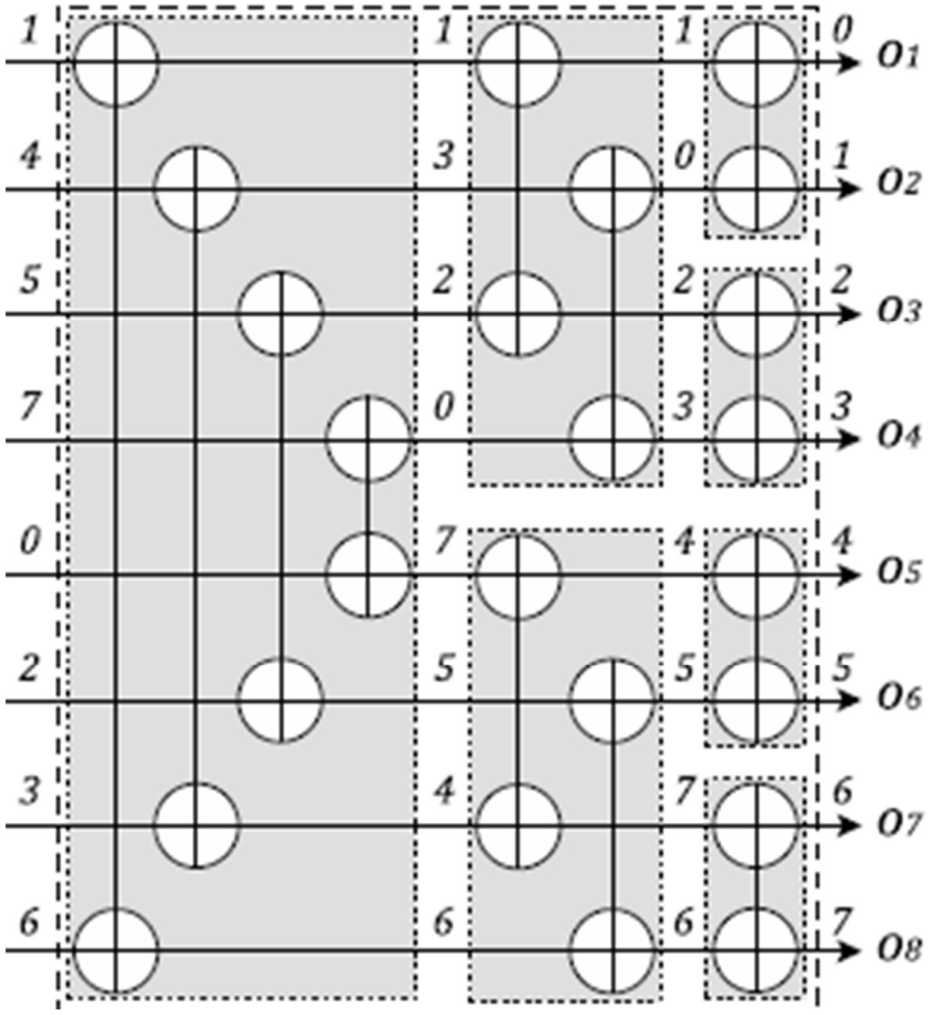


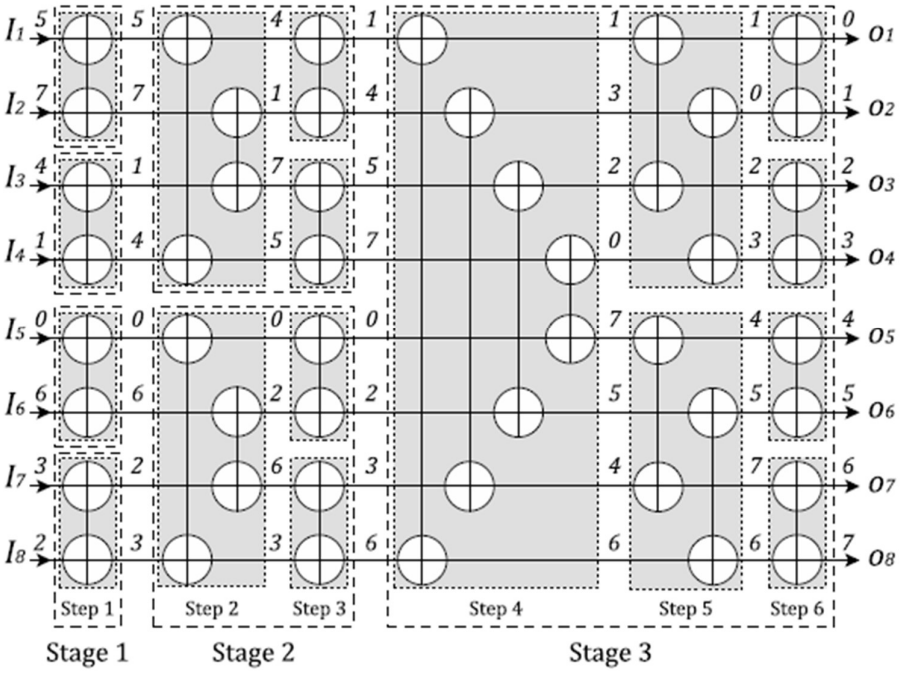
figure above represents the bitonic BM8 unit for ascending order sorting

Design ports specification:

Signal Name	Direction	Bit Width	Description
in1	input	32	Input data 1 to be compared
in2	input	32	Input data 2 to be compared
in3	input	32	Input data 3 to be compared
in4	input	32	Input data 4 to be compared
in5	input	32	Input data 5 to be compared
in6	input	32	Input data 6 to be compared
in7	input	32	Input data 7 to be compared
in8	input	32	Input data 8 to be compared
clk	input	1	Clock of the design
enable	input	1	Data is fed in the design only when the input enable signal is high
reset	input	1	Reset signal of the Design, asynchronously set
o1	output	32	Output data 1 of the BM8
o2	output	32	Output data 2 of the BM8
o3	output	32	Output data 3 of the BM8
o4	output	32	Output data 4 of the BM8
o5	output	32	Output data 5 of the BM8
o6	output	32	Output data 6 of the BM8
o7	output	32	Output data 7 of the BM8
o8	output	32	Output data 8 of the BM8

2.4. Week 4 (Final) - Eight Input Bitonic Sorter Design

Eight Input Bitonic Sorter Design



Design ports specification:

Signal Name	Direction	Bit Width	Description
in1	input	32	Input data 1 to be compared
in2	input	32	Input data 2 to be compared
in3	input	32	Input data 3 to be compared
in4	input	32	Input data 4 to be compared
in5	input	32	Input data 5 to be compared
in6	input	32	Input data 6 to be compared
in7	input	32	Input data 7 to be compared
in8	input	32	Input data 8 to be compared
clk	input	1	Clock of the design
enable	input	1	Data is fed in the design only when the input enable signal is high
reset	input	1	Reset signal of the Design, asynchronously set
o1	output	32	Output data 1 of the sorter
o2	output	32	Output data 2 of the sorter
o3	output	32	Output data 3 of the sorter
o4	output	32	Output data 4 of the sorter
o5	output	32	Output data 5 of the sorter
o6	output	32	Output data 6 of the sorter
o7	output	32	Output data 7 of the sorter
o8	output	32	Output data 8 of the sorter

3. AHP Project B – RISC CPU

3.1. Week 1 – ALU unit

DATA WIDTH- 8 bits

Description: Use Verilog operators while describing a parameterized-width arithmetic/logic unit (ALU). The arithmetic/logic unit, depending upon the operation encoded within the instruction, selects between the inputs or operates upon both inputs. It also outputs the truth state of the in_a input, that is, That is, a_is_zero is true when in_a is zero.

The operation value, instruction, operation and output are as follows:

0	HLT	PASS A	in_a
1	SKZ	PASS A	in_a
2	ADD	ADD	in_a + in_b
3	AND	AND	in_a & in_b
4	XOR	XOR	in_a ^ in_b
5	LDA	PASS B	in_b
6	STO	PASS A	in_a
7	JMP	PASS A	in_a

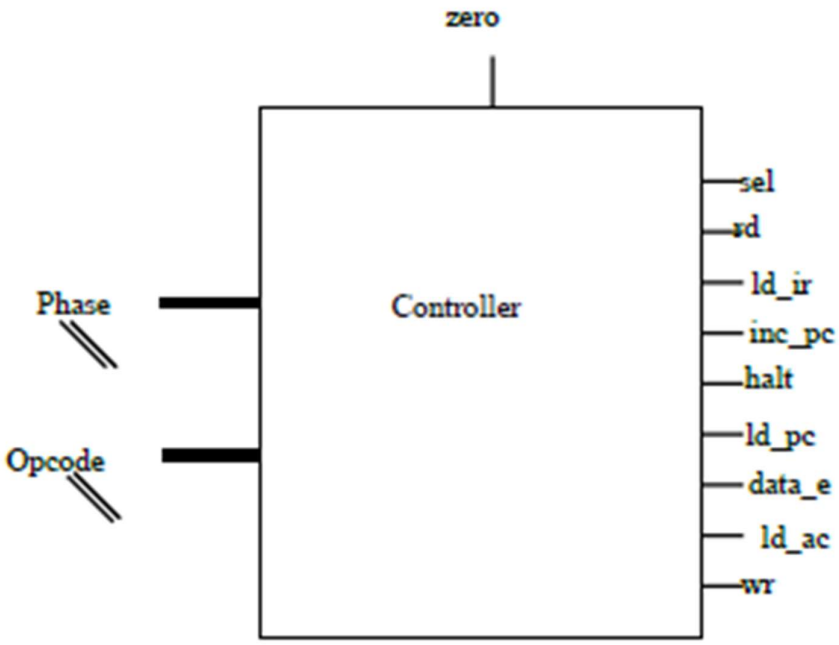
Design ports specification:

Signal Name	Direction	Bit Width	Description
opcode	input	3	Operational value of the ALU
in_a	input	8	Input A
in_b	input	8	Input B
a_is_zero	output	1	Truth state of the input A
alu_out	output	8	Output value of the ALU

3.2. Week 2 – Controller

Description: Design a controller such that the opcode, phase and zero signals determine the control signals. Use the provided testbench to verify your design.

The operation value, instruction, operation and output are as follows:



Design ports specification:

Input Signal Name	Direction	Bit Width	Description
opcode	input	3	Operational value of the ALU
phase	input	3	Phase value for controller
zero	input	1	Signal is set to high if value in accumulator is zero (*when integrated with complete design*)

Output Control Signals: sel, rd, ld_ir, inc_pc, halt, ld_pc, data_e, ld_ac, wr

The Outputs for each phase is determined by the table below:

phase	sel	rd	ld_ir	inc_pc	halt	ld_pc	data_e	ld_ac	wr
0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	1	1	1	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0
4	0	0	0	1	H	0	0	0	0
5	0	A	0	0	0	0	0	0	0
6	0	A	0	Z	0	J	S	0	0
7	0	A	0	0	0	J	S	A	S

H: High if instruction is HALT.

A: High if instruction involves accumulator, i.e. ADD, AND, XOR, LDA.

Z: High if instruction is SKZ and zero is true.

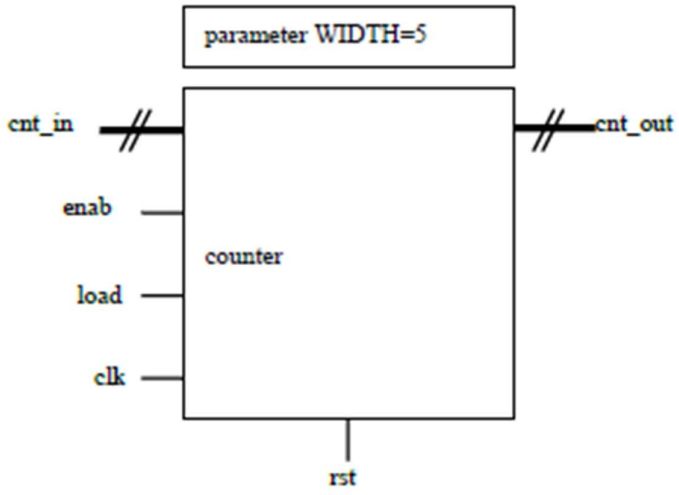
J: High if instruction is JMP.

S: High if instruction is STO.

3.3. Week 3 - Counter, Register and Memory Units

Design the following blocks: Counter, Register and Memory as per the given functional specification.

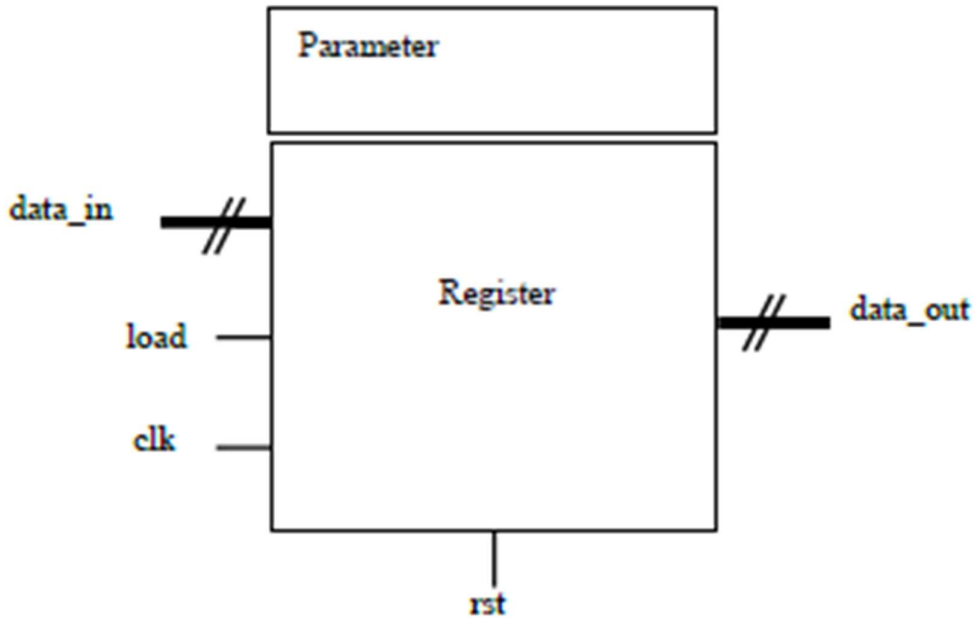
1. **Counter:**



Design ports specification:

Input Signal Name	Direction	Bit Width	Description
Clk	Input	1	Clock input
Rst	Input	1	Reset input
Load	Input	1	Load signal, if set to high will assign counter output to the assigned value cnt_in
Enab	Input	1	Will perform normal counting incrementally when Enable signal is set to high
Cnt_in	Input	5	Input value to the counter, assigned to cnt_out only when load is set high
Cnt_out	Output	5	Counter output

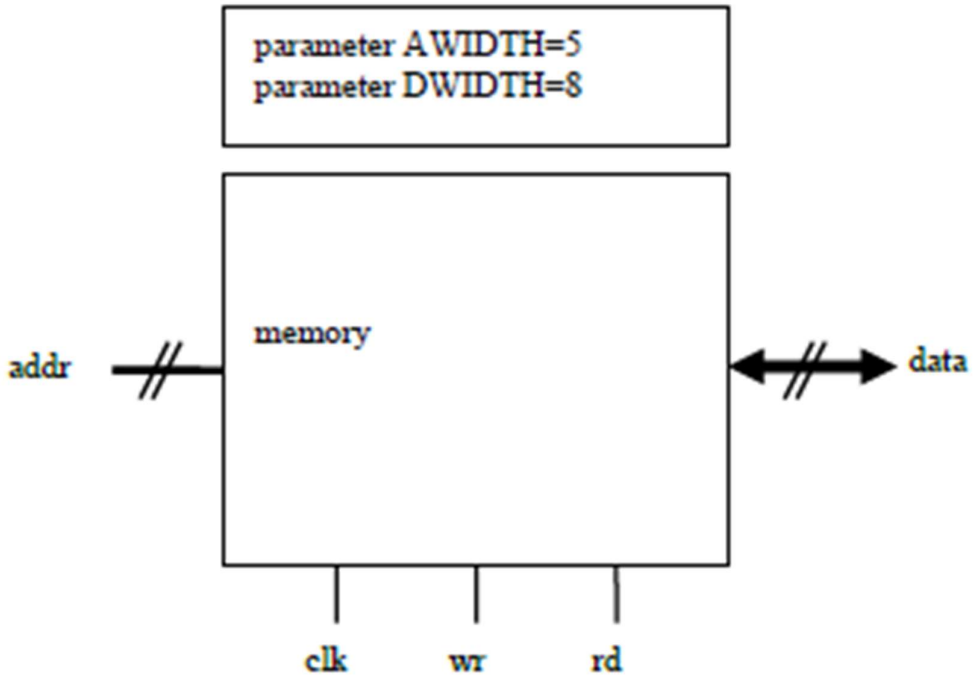
2. **Register:**



Design ports specification:

Input Signal Name	Direction	Bit Width	Description
Clk	Input	1	Clock input
Rst	Input	1	Reset input
Load	Input	1	Load signal used to control when the data input is assigned to the data_out
Data_in	Input	8	Input of the register
Data_out	Output	8	Output of the register

3. Memory



Design ports specification:

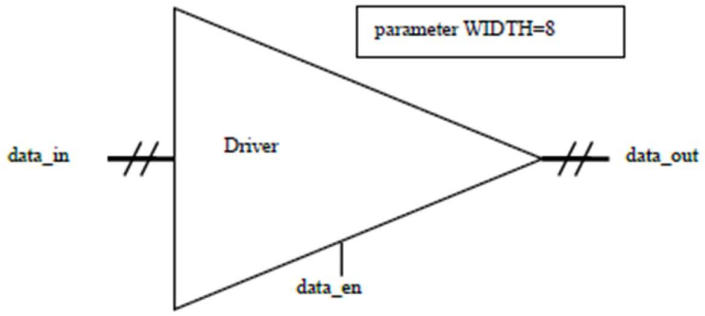
Input Signal Name	Direction	Bit Width	Description
Clk	Input	1	Clock input for the design
Wr	Input	1	Control signal which allows the data to be assigned in the memory address only when set high
Rd	Input	1	Used to assign the stored data at a given address into the data line if set to high
Addr	Input	5	Address value to locate the data in the memory
Data	Inout	8	Data written in, stored and read out of the memory

3.4. Week 4 (Final) – Driver, Mux and Final CPU integration

RISC CPU Design

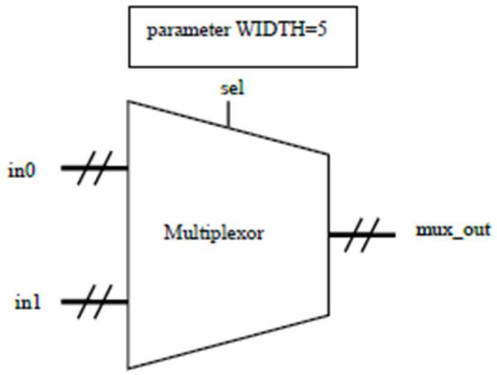
Additional Modules to be designed:

1. **Driver**



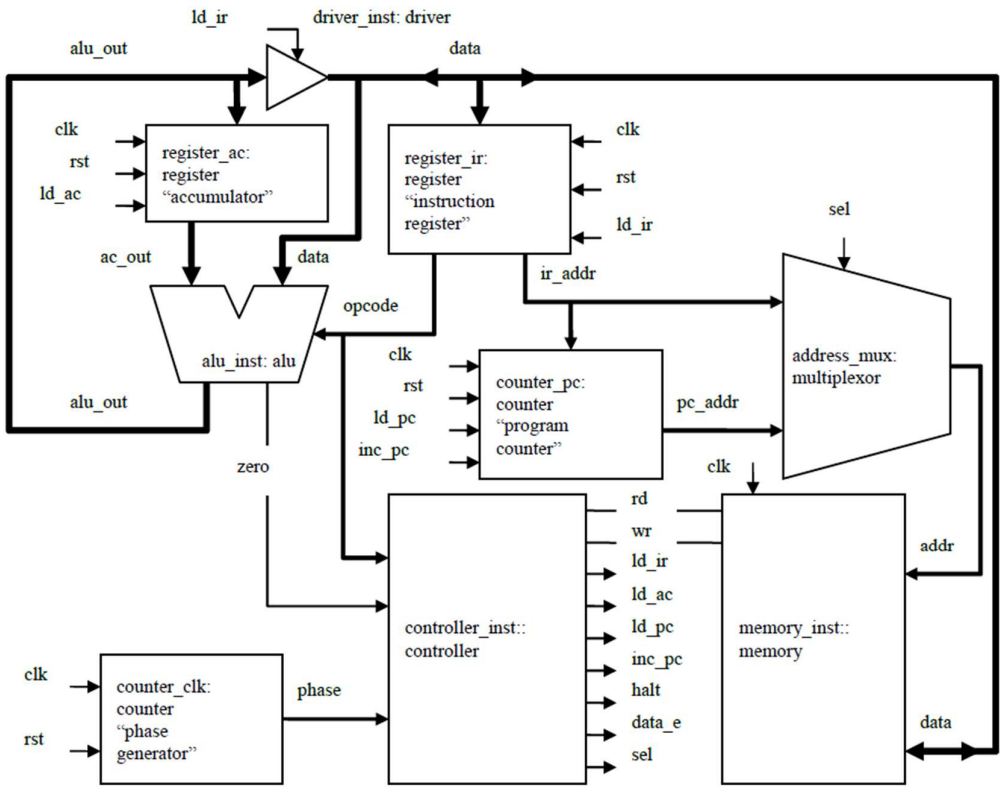
Port Name	Bit Width	Direction	Description
Data_in	**8**	**Input**	**Input of the driver**
Data_en	**1**	**Input**	**Enable Signal**
Data_out	**8**	**Output**	**Output of the driver**

2. **Multiplexor**



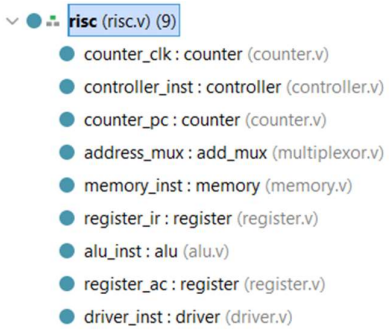
Port Name	Bit Width	Direction	Description
Sel	1	Input	Select line for the multiplexor
In0	5	Input	Multiplexor input
In1	5	Input	Multiplexor input
Mux_out	5	Output	Multiplexor output

Final RISC CPU design:



Instructions for Integrating the design:

1. **Use the given risc.v file (as a top module) include all the submodules**



2. **Use the risc_test.v file as your testbench for the top module and include the following files: CPUtest1.txt, CPUtest2.txt and CPUtest3.txt in your present working directory / include them in sources (under All files).** ***Self learning component: including files using $readmem***

3. **Run Simulation and the console must perform tests in order, ultimately providing 'TEST PASSED' command.**
   In case the Test Fails at a particular sub-module, verify the design and its integration with the top module, and restart simulation.

4. AHP Project C – Approximate Adder Tree

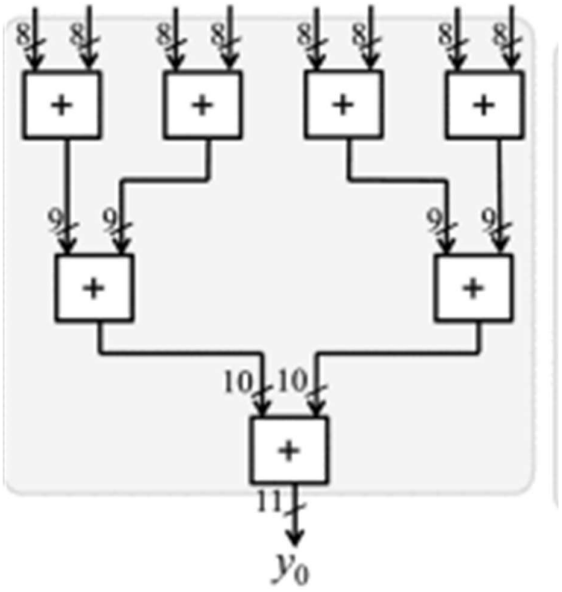
4.1. Week 1 – Full Width Adder Tree

Full Width Adder Tree

DATA WIDTH- 8 inputs with 8 bit data width

Description:
In the first stage of the adder tree, consist of four two input adders (8 bit inputs) and provides output of 9 bits (carry). These four outputs are the inputs for stage 2 of the adder tree, where we use two adders (9bit input) to provide outputs of 10bits(carry) and finally this is summed in a 10 bit adder to provide a 11 bit output

This design is the basis for DSP applications.



Design ports specification for a single adder block:

Signal Name	Direction	Bit Width	Description
A	input	8/9/10	Adder input 1
B	input	8/9/10	Adder input 2
Sum	output	9/10/11	Adder output

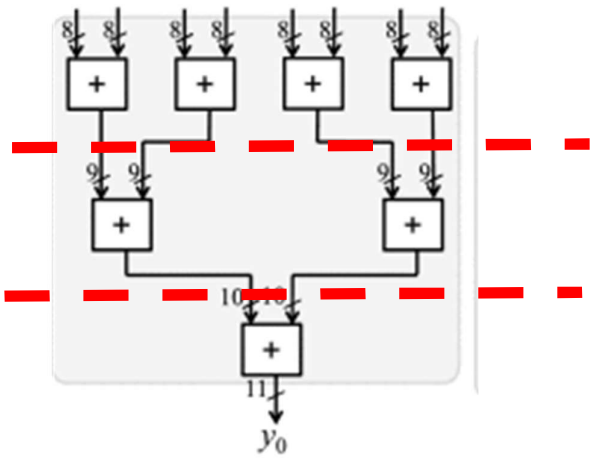
4.2. Week 2 - Pipelined Full Width Adder Tree

Week 1 Project C Pipelined Full Width Adder Tree

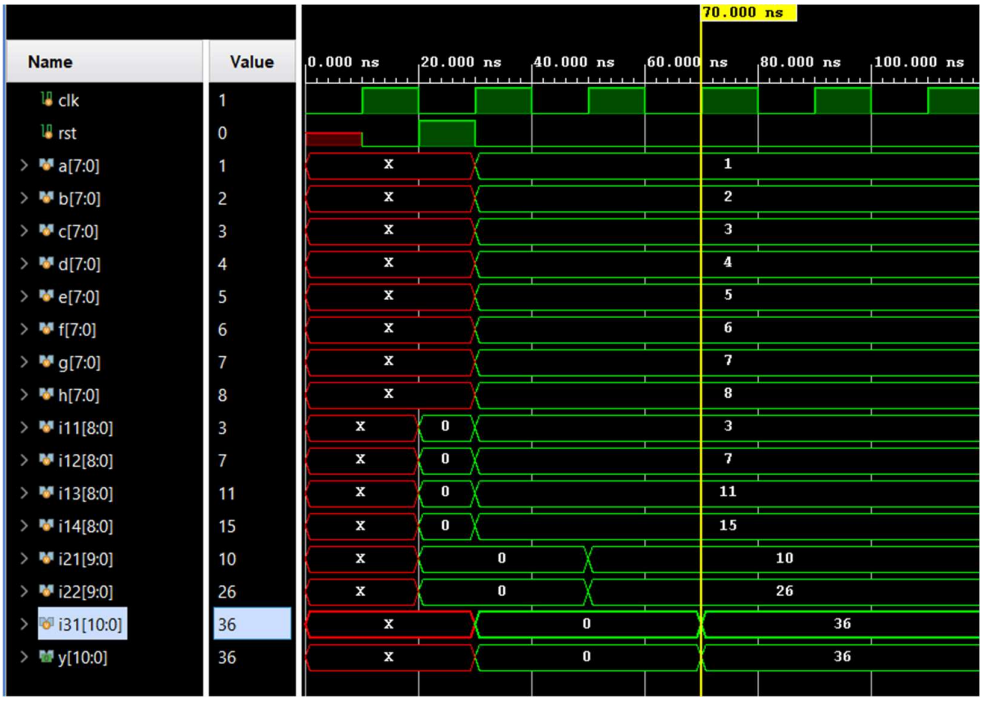
DATA WIDTH- 8 inputs with 8 bit data width

Description:

Insert pipeline stages in the previous full width adder tree design in the regions indicated by the dotted lines. The data from the previous stage enters the next only at the positive edge of the clock, when enable signal is high. The data also resets to zero if the reset signal is pulled high.



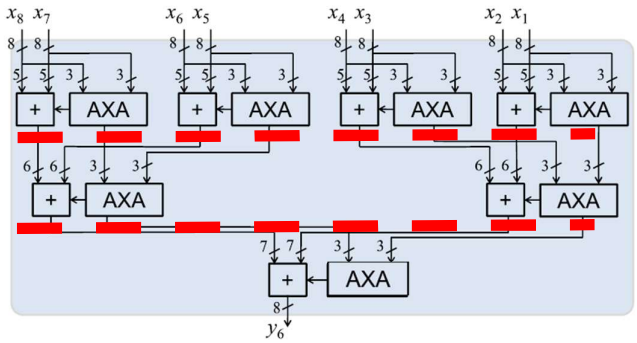
Note: Use two segment Verilog coding and ensure that the output is obtained after two clock cycles as per waveform given below for inputs a,b,c,d,e,f,g,h and intermediary registers i11, i12, i13, i14, i21, i22



4.3. Week 3 - Approximate OR and XOR modified

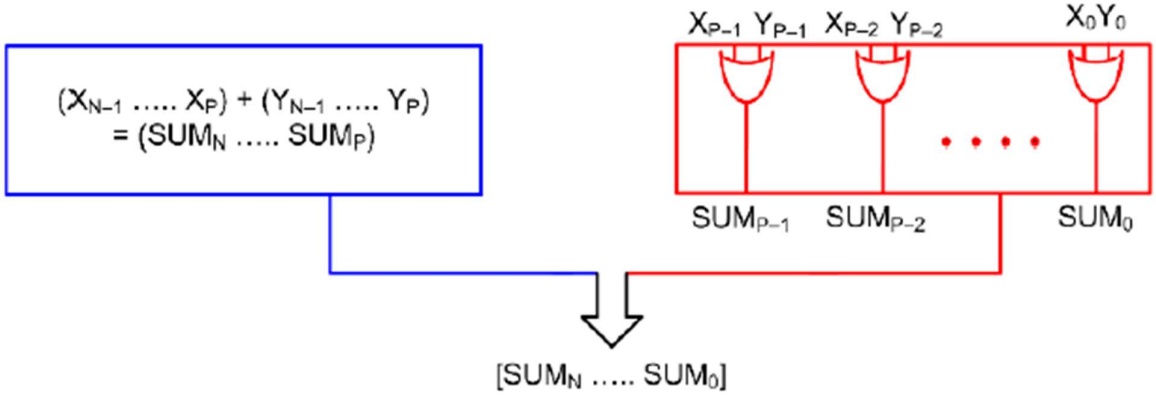
Description:

For the Pipelined Adder tree design, include a new approximator block AXA for the last 3 bits of the input's LSB to design two different tree designs, each performing
(i) bitwise or operation in AXA block
(ii) bitwise xor operations in AXA block
as depicted in the diagram below.



Note: Use two segment Verilog coding and ensure that the output is obtained after two clock cycles as per waveform given below for inputs a,b,c,d,e,f,g,h and intermediary registers i11, i12, i13, i14, i21, i22

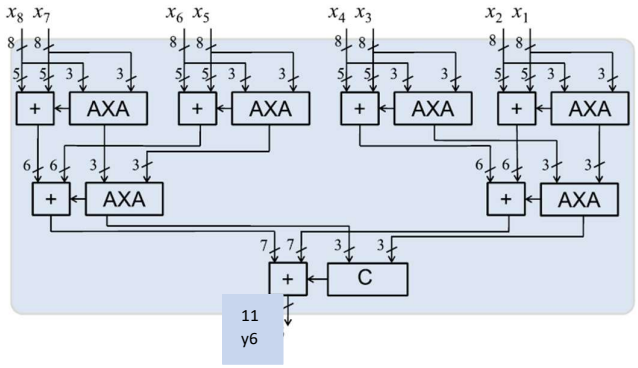
For example, given two inputs X and Y in the adder tree, The AXA block for bitwise or is depicted in red as follows, while normal addition operation is performed in the blue box.



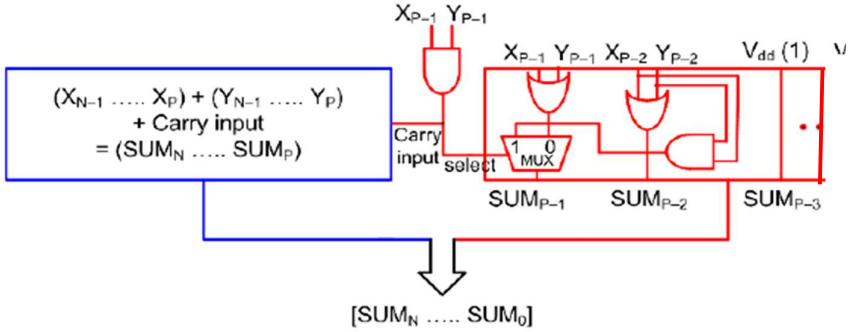
Repeat the same procedure to design an AXA block for **bitwise xor**.
Note: For testing, consider the inputs {1,2,3,4,5,6,7,8} in the same sequence for both the designs (Adder tree with AXA-bitwise-or and Adder tree with AXA-bitwise-xor)

4.4. Week 4 (Final) – Perform Approximation D1 and D2 for the Pipelined Adder tree design

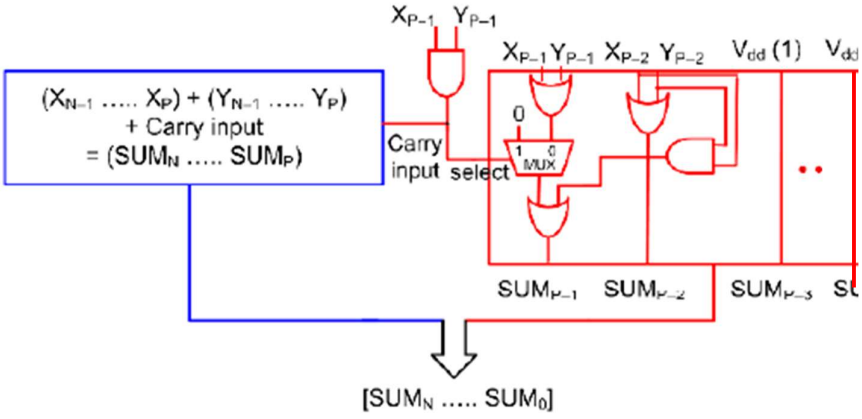
Perform Approximation D1 and D2 for the Pipelined Adder tree design



D1 approximation in the AXA block:



D2 approximation in the AXA block:



Note: Provide inputs {1,2,3,4,5,6,7,8) for the 8 input Approximate Adder tree modifications D1 and D2 and show the final output in the simulation.

Also find the worst case error due to approximation in both the designs.

5. AHP Project D - MAC unit

5.1. Week 1 – MAC unit 1x1

DATA WIDTH- 32

Description: MAC (Multiply and Accumulate) is used to multiply and add the subsequent data in the subsequent clock cycles.

Design ports specification:

Signal Name	Direction	Bit Width	Description
clk	input	1	Clock signal
Rst	Input	1	Reset signal
In_a	input	32	Input 1 for the MAC
In_b	input	32	Input 2 for the MAC
Out_mac	output	64	Mac output

Expected functioning of the MAC circuit

in_A	In_b	Clock cycle number	Out_mac
A0	B0	1	A0 * B0 + 0
A1	B1	2	A1 * B1 + A0 * B0
A2	B2	3	A2 * B2 + A1 * B1 + A0 * B0

5.2. Week 2 – Mac Unit 2x1

DATA WIDTH- 32

Description: MAC (Multiply and Accumulate) is used to multiply and add the subsequent data in the subsequent clock cycles.

Design ports specification:

Signal Name	Direction	Bit Width	Description
clk	input	1	Clock signal
Rst	Input	1	Reset signal
In_a	input	32	Input 1 for the MAC
In_b	input	32	Input 2 for the MAC
Out_mac	output	64	Mac output

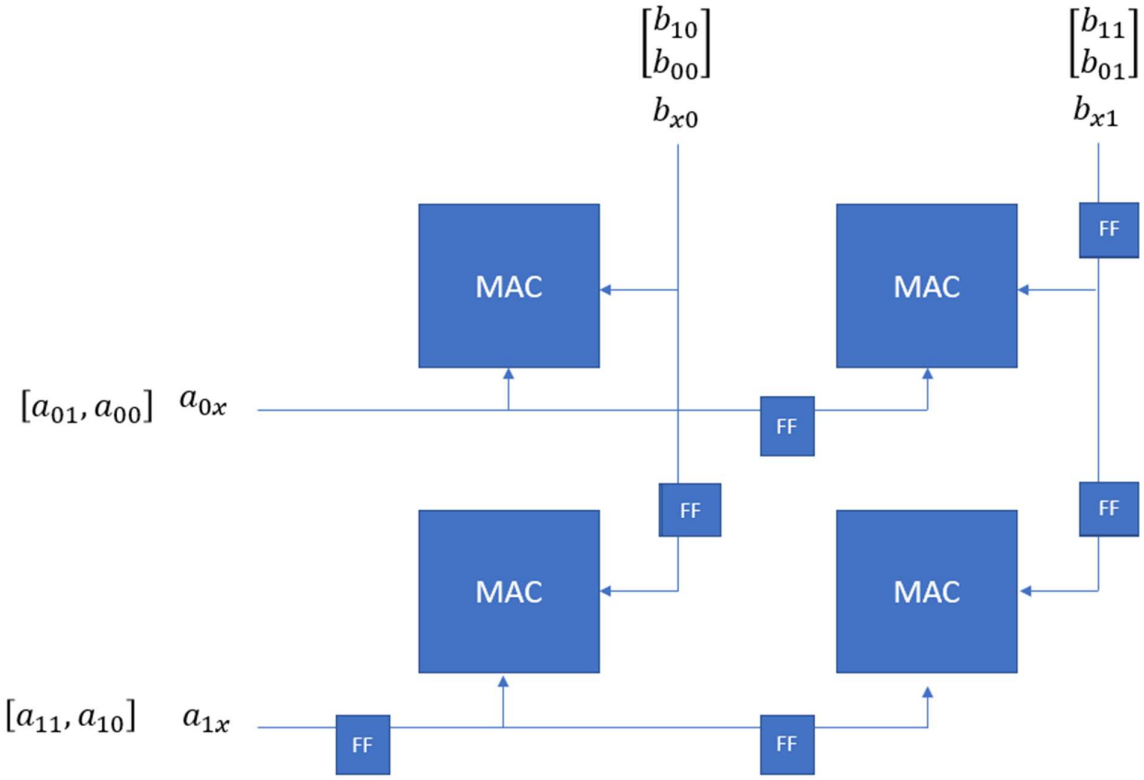
Expected functioning of the MAC circuit

in_A	In_b	Clock cycle number	Out_mac
A0	B0	1	A0 * B0 + 0
A1	B1	2	A1 * B1 + A0 * B0
A2	B2	3	A2 * B2 + A1 * B1 + A0 * B0

5.3. Week 3 – Mac Unit 2x2

Description:
Use the previously designed MAC units in the structure shown below to form a systolic array matrix multiplier. Insert flipflops as shown in order to get the necessary delays between the MAC stages. The output of each MAC represents one of the elements of the product matrix.

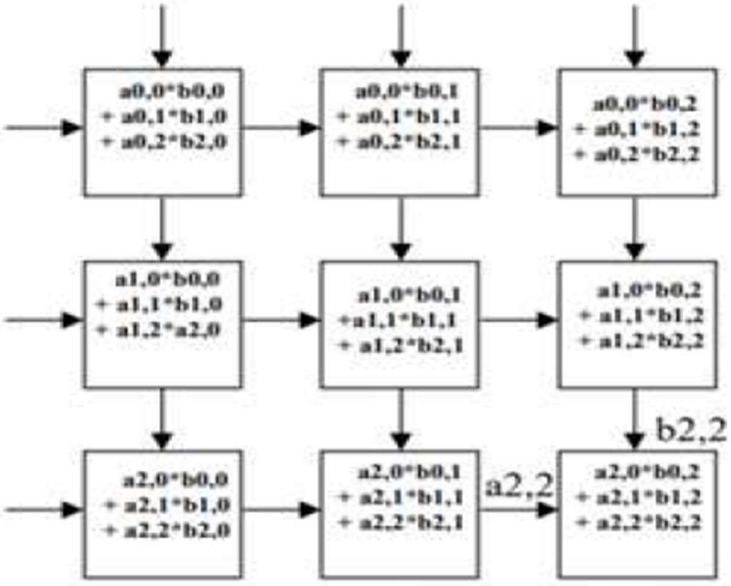
$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} a_{00} \times b_{00} + a_{01} \times b_{10} & a_{00} \times b_{01} + a_{01} \times b_{11} \\ a_{10} \times b_{00} + a_{11} \times b_{10} & a_{10} \times b_{01} + a_{11} \times b_{11} \end{bmatrix}$$



5.4. Week 4 (Final) - Mac Unit 3x3

Description:

Use the previously designed MAC units in the structure shown below to form a 3x3 systolic array matrix multiplier. Insert flipflops as shown in order to get the necessary delays between the MAC stages. The output of each MAC represents one of the elements of the product matrix.



Reference: week3's 2x2 martix mac