ASSIGNMENT-2-STAGE-5

cs1200840

March 2022

1 FILES/FOLDERS/NEW ENTITIES

- 7 new .vhd files one with each entity are added as compared to previous stage.(apart from testbenches)
- 5 entities are shift1, shift2, shift3, shift4, shift5 which will do one of LSR,ASR,ROR by 1,2,4,8,16 bits respectively.
- There is an entity for bit_reversal in bit_reversal.vhd, which will be used for instruction like LSL, where the input is reversed LSR is performed and the output is also reversed
- The file shifter.vhd has instantiations of shift1,shift2,shift3,shift4,shift5 and 2 bit_reversal components, and is the main entity responsible for shifting
- The file testbench_shifter.vhd contains testcases for LSL, LSR, ASR, ROR instructions of shifter, and the output on each instruction is matched with the expected output and error is raised if any.
- The file decoder.vhd and TYPES.vhd has been added with more outputs and types as follows - DT_operand_src(which is the source of offset for DT instruction(imm,reg)), DP_shifttype(LSL - "00",LSR- "01",ASR-"10",ROR-"11"), DP_shiftamount(whether shiftamount for dp instruction is constant or it comes from a register), DT_shifttype (LSL,LSR,ASR,ROR)
- In the FSM.vhd file some more input-output signals are added which are explained later.
- In datapath.vhd file changes are made to incorporate the shifter and the new states of FSM

2 FSM CHANGES

-
- Two new states DP_DT_Shift and getShiftAmount are added in the FSM.

- The state DP_DT_Shift is used by dp instructions to provide the appropriate shift/rotate to the third operand, and by DT instructions to shift/rotate the offset (when it comes from a register) and the output from shifter is used by ALU(The output data of shifter is stored is in register SR and the carry_out is stored in SCR).

- In case of DP instructions where the offset also comes from a register a new state getShiftAmount is added during which the amount of shift is fetched from the register and is given as input to shifter.

- For DT instructions with offset coming from register, no extra state is required as the offset is already fetched in the second common stage in register B

3 TESTCASES

- testbench_processor1.vhd contains sufficient number of number of clock cycles to check for each of the 4 testcases based on shift and rotate operations.

- The corresponding .s and .txt(containing machine code) files for each testcase are given and the memory has to be loaded with the testcases from .txt files.
  4 TESTCASES -

- dp_lsl_lsr_with4op.s - There are 2 move instructions followed by an add instruction with left shift and shift amount specified as a constant. The result upto these 3 instruction is that r4 should contain 13 (0xD). Then and is taken with right shift and shift amount specified as constant. The result of this instruction would cause r4 to store 0 . Then 2 move instructions are there followed by xor/eor of with left shift coming from a register r4. This would finally result in register r4 storing the value 13(0xD)

- constant_rotate_dt_shift.s - The first instruction mov a constant 1020 into register r3(Since 1020 is greater than 255, rotate has to be done by shifter). Then there are 4 simple move instructions. The second last instruction is str instruction with offset(byte) coming from register r4 and shift specified. This would result in storing contents of register r5(2) into mem(15) (60 byte address). From the same address contents of memory are loaded back into r0 through a ldr instruction with offset coming from register and constant shift. r0 would be storing 2 at the end

- asr_branch.s - First instruction is add instruction with constant second operand. Then a mov instruction with asr of 1 unit is specified. This would result in r1 having the value 2. Next instruction mov -9(0xFFFFFFF7) into r2. The next instruction is movs instruction with constant specified arithematic right shift.This would cause r3 to store -2(0xFFFFFFFE).

Since the instruction is movs and carry out from shifter would be 1, carry flag would be set to 1. In the next instruction adc, r5 is updated by its previous value + 1 + carry(which would be 1), resulting in r5 to have a value 1. r4 is added with contents of r3 and r1 (-2+2) resulting in 0.
These instructions are followed by two branch instructions and compare instructions to automate testing. If the output are not as expected(r5 is not equal to 2 to r4 is not equal to 0), then r4 is loaded with 1 to indicate error.

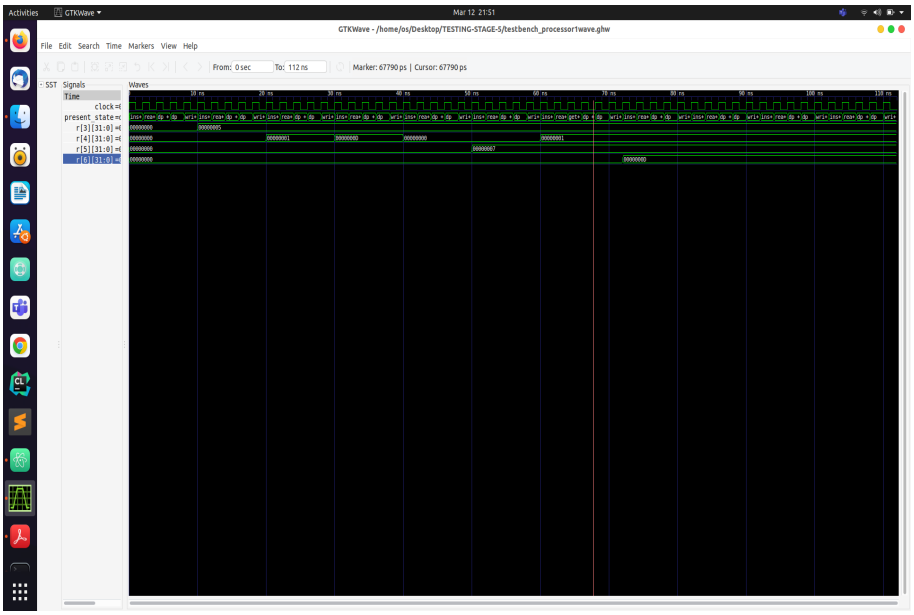
- ror_branch.s - r0 is loaded with -3 , followed by a mov instruction with rotate right by 30 bits. This would cause r1 to store the value the value -9(0xFFFFFFF7). Then eor instruction is there with a lsl of 1 unit. This would cause r2 to store 13((-9)xor (-6)) .
cmp and branch instructions are at end to automate testing if r2 is storing 13 then r4 will store 0. In case of error(r2 not equal to 13) r4 will store 1

4 SIMULATION

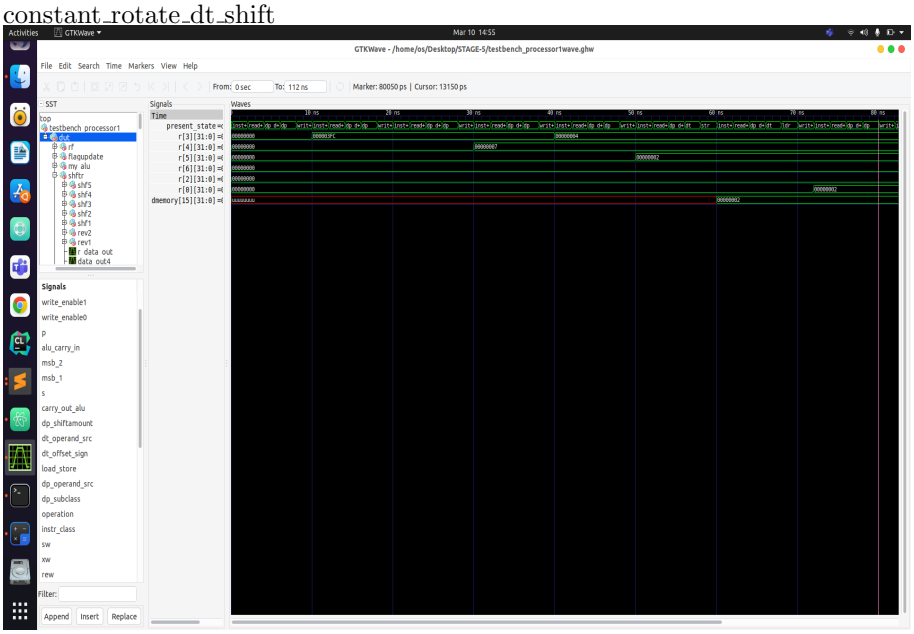
- Testbench_shifter

TESTCASES FOR COMPLETE PROCESSOR

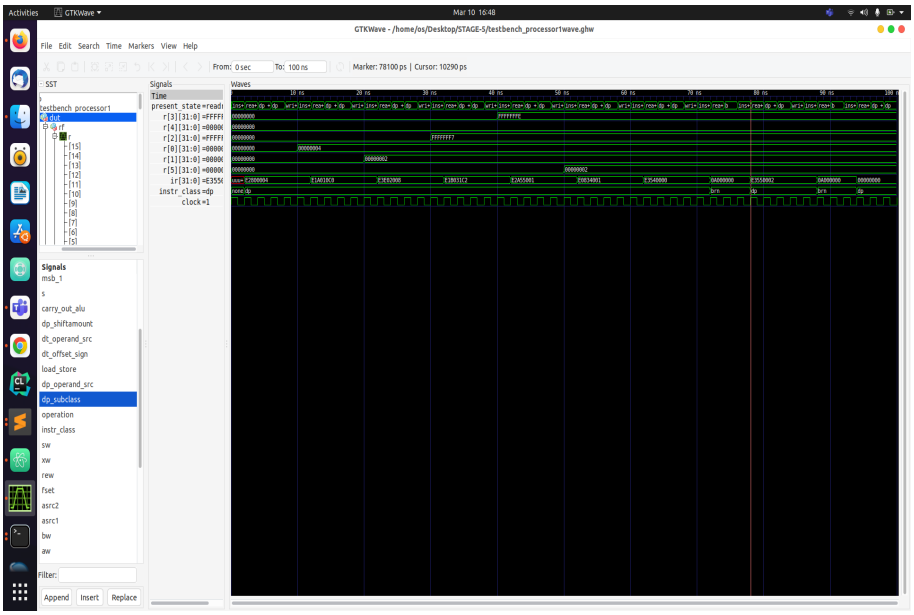
- dp_lsl_lsr_with4op



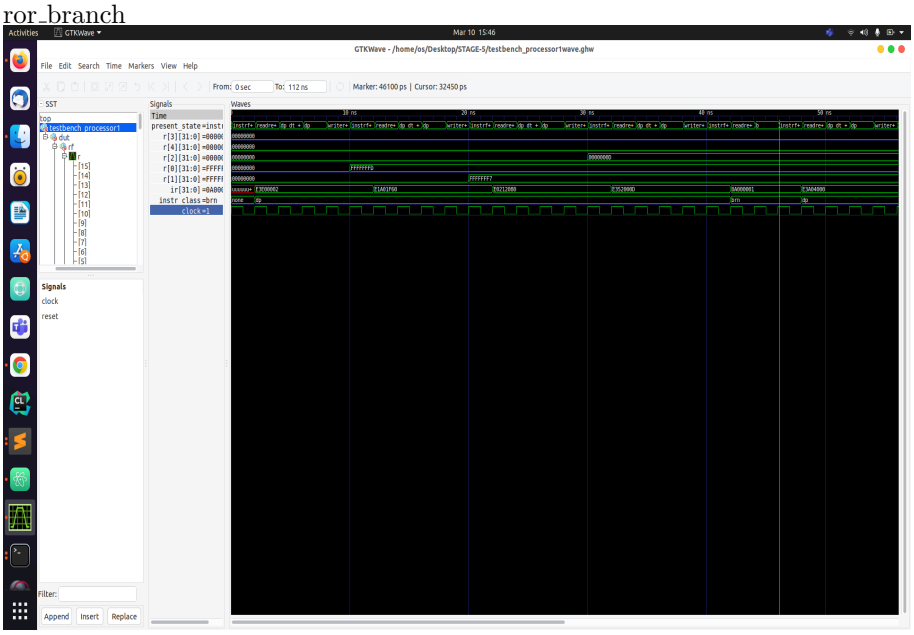
- constant_rotate_dt_shift



- asr_branch

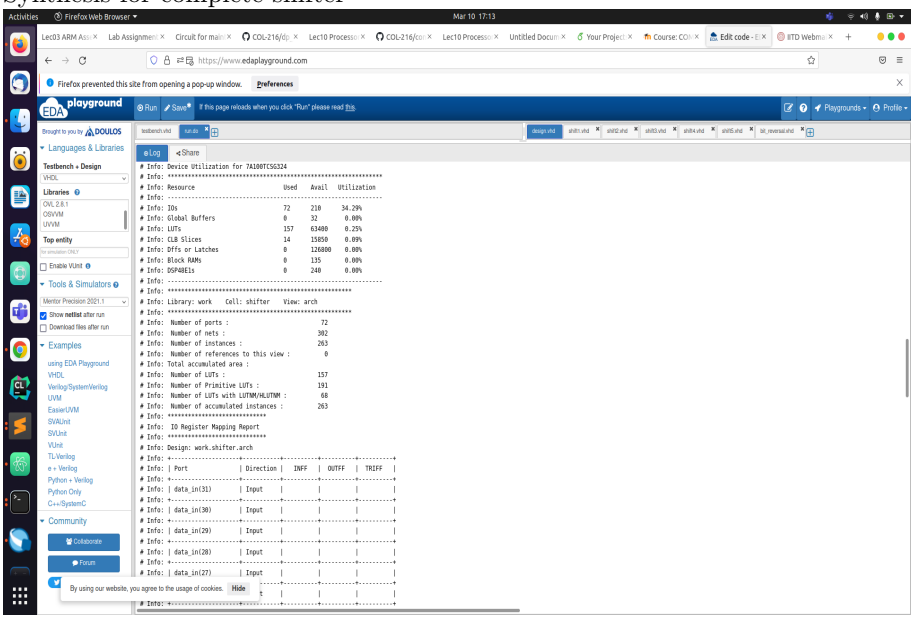

Activities
GTKWave
Mar 10 16:48
GTKWave - /home/os/Desktop/STAGE-5/testbench_processor1wave.ghw
File Edit Search Time Markers View Help
From: 0 sec
To: 100 ns
Marker: 78100 ps | Cursor: 10290 ps
SST
testbench_processor1
dut
rf
Signals
msb_1
s
carry_out_alu
dp_shiftamount
dt_operand_src
dt_offset_sign
load_store
dp_operand_src
dp_subclass
operation
instr_class
sw
xw
rew
fset
asrc2
asrc1
bw
aw
Filter:
Append
Insert
Replace
Signals
Waves
Time
present_state=readi
r[3][31:0] =FFFFI
r[4][31:0] =0000(
r[2][31:0] =FFFFI
r[0][31:0] =0000(
r[1][31:0] =0000(
r[5][31:0] =0000(
ir[31:0] =E355(
instr_class =dp
clock =1

- ror_branch

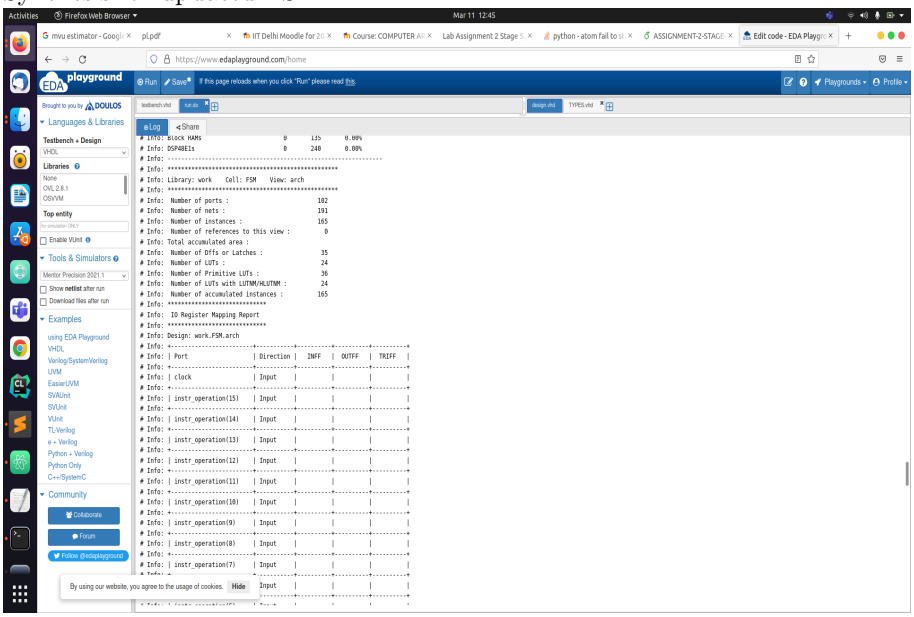

Activities
GTKWave
Mar 10 15:46
GTKWave - /home/os/Desktop/STAGE-5/testbench_processor1wave.ghw
File Edit Search Time Markers View Help
From: 0 sec
To: 112 ns
Marker: 46100 ps | Cursor: 32450 ps
SST
top
testbench_processor1
dut
rf
Signals
clock
reset
Filter:
Append
Insert
Replace
Signals
Waves
Time
present_state=instr
r[3][31:0] =0000(
r[4][31:0] =0000(
r[2][31:0] =0000(
r[0][31:0] =FFFFI
r[1][31:0] =FFFFI
ir[31:0] =0A00(
instr_class =brn
clock =1

5 SYNTHESIS

- Synthesis for complete shifter

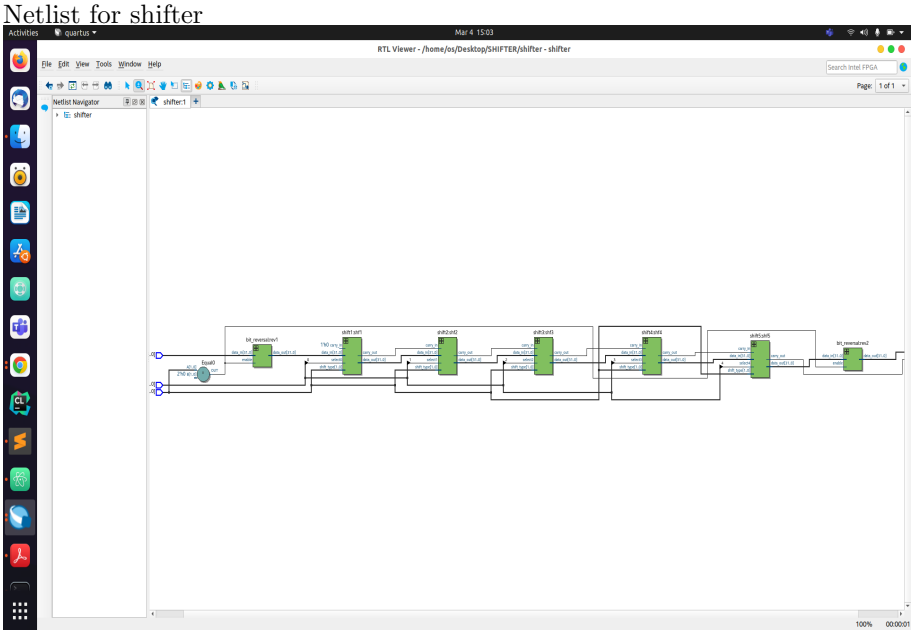


- Synthesis for updated FSM

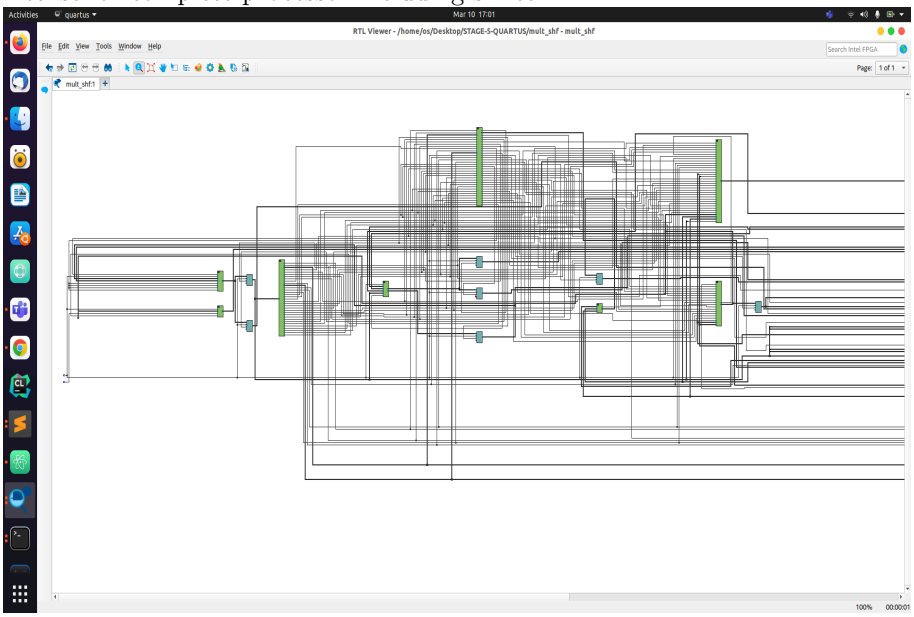


6 NETLIST (GRAPHICAL GENERATED VIA QUARTUS)

- Netlist for shifter



- Netlist for complete processor including shifter



7 NOTE

- I have added synthesis report of shifter(which is a new component), and FSM (which has some updates)
- Synthesis of complete processor/datapath on eda and quartus gave only two input ports and no other resources.(This problem also happended with many others in stage-2 that is why prof. anshul sir said on piazza at that time to include only simulation result for it) . So the resource utilization for that is not attached (however netlist generated by quartus is a)
- Other files like run.do, Makefile for ghdl are also present
- Images for synthesis and simulation are also present in the folder submitted apart from the report
- Testcases used (with .txt and .s files) are in folder TESTCASES.
- SIMULATION folder contains simulation results and SYNTHESIS folder contain synthesis report of FSM and shifter and netlist.
- Small changes were made in alu to make it synthesisable with quartus(although with mentor precision synthesis was possible without this)