SAN JOSE STATE UNIVERSITY
Charles W. Davidson College of Engineering
DEPARTMENT OF ELECTRICAL ENGINEERING

EE 271 – Advanced Digital System Design and Synthesis

Fall 2014 Final Project Report

A Simple 8-bit Scalar Processor

Name	MOHIT LALIT GOGRI	**Email**	mohit.gogri@sjsu.edu
SJSUID	009994530	**Phone**	669-221-9605

Executive Summary (50 to 80 words)

The project is intended to analyze the design of 8 bit scalar processor whichch is available to do several operations and understand the relationship between timing, power, area and operand sizes.

Register bank : 5 user visible registers are provided AR,DR,GR,PR,FR on which the processor has to work its instructions.

Alu: Alu in this design has only 2 operations i.e. Addition and Subtraction and it has to work this operations on the registers from the register bank.

Memory: The memory here is 8bit wide and 256 bit long . From 0 to 127 is the code memory and 128 to 255 is data memory.The processor gets the value from the memory and registers are initialise with this memory values only.Memory is connected to the data bus and address bus of processor as well as write and read signal.

Theoretically, the Processor consumes a greater area due to presence of alu which give rise to more power consumption on less clock period.But the design gives the understanding of relation between time,area and power.

I. General Project Information

Table I.1: List of EDA Tools Used

Name	Company	Used for	Free? (Y/N)	Software Documents
Vcs	Synopsys	Simulation & test	No	
Design Vision	Synopsys	Synthesis	No	

Table I.2: List of Libraries Used

Library file name	Company	Used with (EDA tool)	The libraries are at (directories on eecad systems)
tc240c.db_BCCOM25	Toshiba	Synopsys Design vision	/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_BCCOM25
tc240c.db_WCCOM25	Toshiba	Synopsys Design Vision	/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_WCCOM25

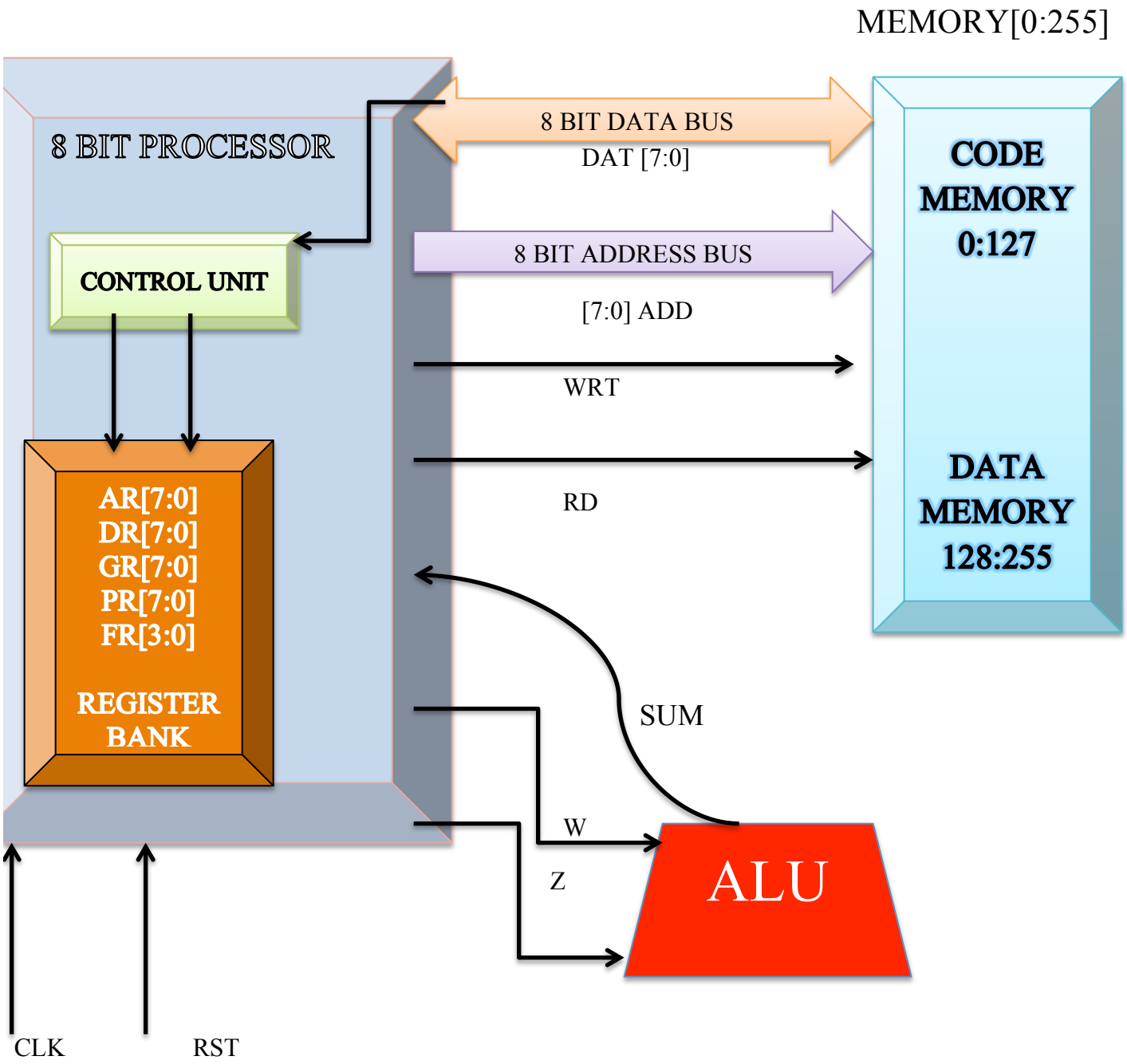
Table I.3: List of Verilog Modules (both design and test modules)

Module Name	Ports	Short Description
processor8bit	• [7:0]add • [7:0] dat • wrt • rd • clk • rst	Output Address bus which carries address to memory Bidirectional Data bus to carry Data . Output Write signal to write value into memory. Output Read signal to read data from the memory. Input Clock signal on which the processor works. Input Reset signal which resets all the registers of the processor.
alu	• [7:0] sum • cout • w[7:0] • z[7:0] • c_in • en • clk	Output Sum which is the result of the alu module . Output cout is the result of the carry generated from alu Input register as a Source register to alu. Input register as a Destination register to alu. Input carry signal. Input enable/mode signal. Input clock signal .
tb_processor	• [7:0] add • [7:0] dat • wrt • rd • clk • rst	Address bus to locate the memory. Data bus for data flow. Write signal to write in memory. Read signal to read from memory. Clock signal to provide input to processor. Reset signal to reset the registers.

II. The Design Overview

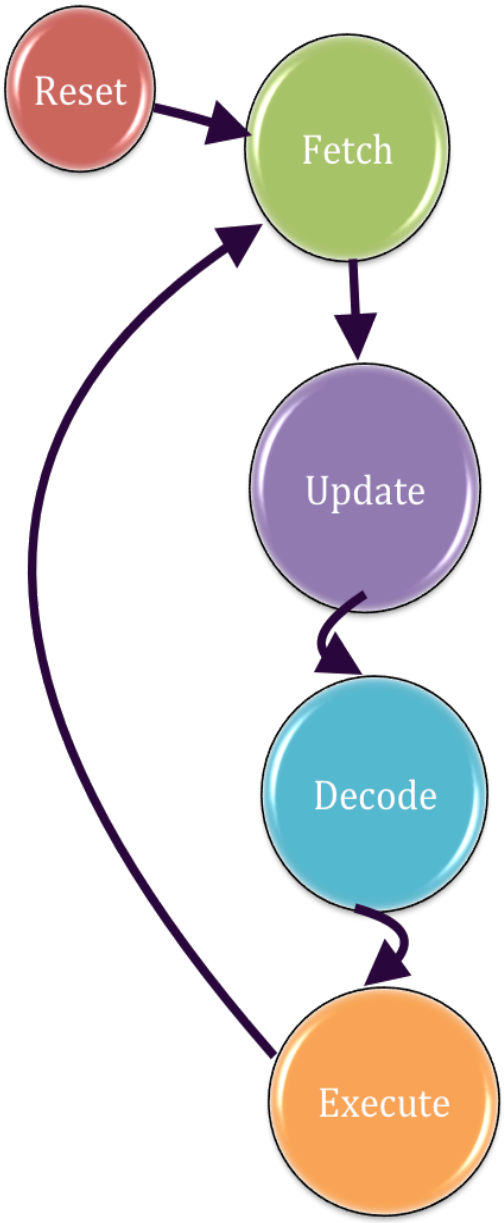
The Microprocessor has totally 4 main blocks ALU, Control Unit, Memory, and Register bank The Microprocessor has the following ports namely Clock, Reset, Data, Address, read, write. The Processor is designed based on state transitions. Every instruction takes min 2 clock cycles and maximum 4 clock cycles to complete the operation. The flow or the architecture of the microprocessor is like the address in the program counter is sent to the Address Register in the first cycle and data is fetched. In next cycle data is assigned to opcode if incoming data is not to load registers. If data is opcode it is given to decoder unit for decode and control unit send signals to different blocks as per need of instruction, finally program counter is executed .Further in next cycle instruction get executed. Hence total 2 clock cycles for Instruction Nop, Jmp, Cpr, Cfr, Lls, Lms is used and 3 clock for RDM, WRM, ADD, SUB is used.

Figure II.1: Overall Processor Block Diagram



DAT: 8 BIT DATA BUS
ADD : 8 BIT ADDRESS BUS
WRT: WRITE SIGNAL
RD: READ SIGNAL
CLK : CLOCK SIGNAL
RST : RESET SIGNAL
W,Z ,AR,DR,PR,GR,FR: REGISTERS
SUM: RESULT OF ALU

Figure II.2: Overall State Transition Diagram(s)



State Diagram :

RESET : when rst is high all the registers are set to zero.Next state is fetch state.

FETCH : It fetches the instruction from the memory.Next state is Update .

UPDATE: It updates the PR to fetch the next instruction. Next state is Decode.

DECODE : It decodes the instruction for the final execution. Next state is Execution.

EXECUTION : It executes the instruction function and gives out the result and then again the process is continued from FETCH state.

Table II.1: Number of Clock Cycles Required for Each Instruction

Instruction	# of Clock Cycles	Instruction	# of Clock Cycles	Instruction	# of Clock Cycles
NOP	1	LMS	1	ADD	2
JMP	1	CFR	1	SUB	2
CPR	1	RDM	2		
LLS	1	WRM	2		

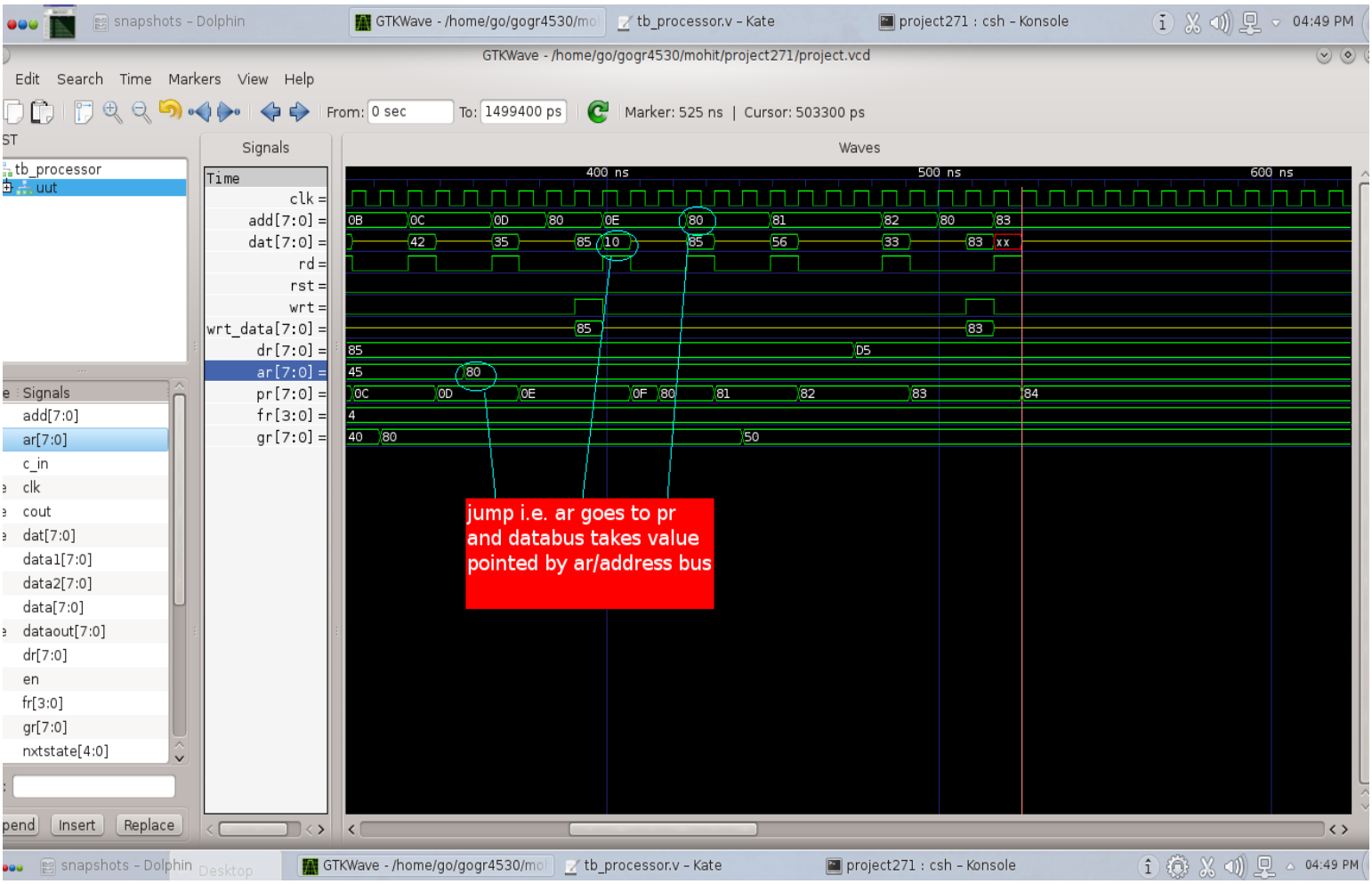
***Describe** the experiments and analysis that you have done to get the optimum number of clock cycles for memory instructions RDM, WRM and for arithmetic instructions ADD, SUB (**less than 50 words**)*

At First the cycle count for NOP and JMP were more than 2 due to increase in number of states and some syntax error.For other instruction too the no. of cycles were more than desired.But after decreasing the state i was able to get the optimum solution for the number of cycles for all instruction.In between i also experiemented with reduction in syntax but it changed the design flow.RDM,WRM,ADD and SUB were taking 4 cycles as the data was taking 1 extra cycle to update.But then by making certain changes in syntax I got the desired no of cycles.

III. RTL-Level Simulations/Tests for Each Instruction

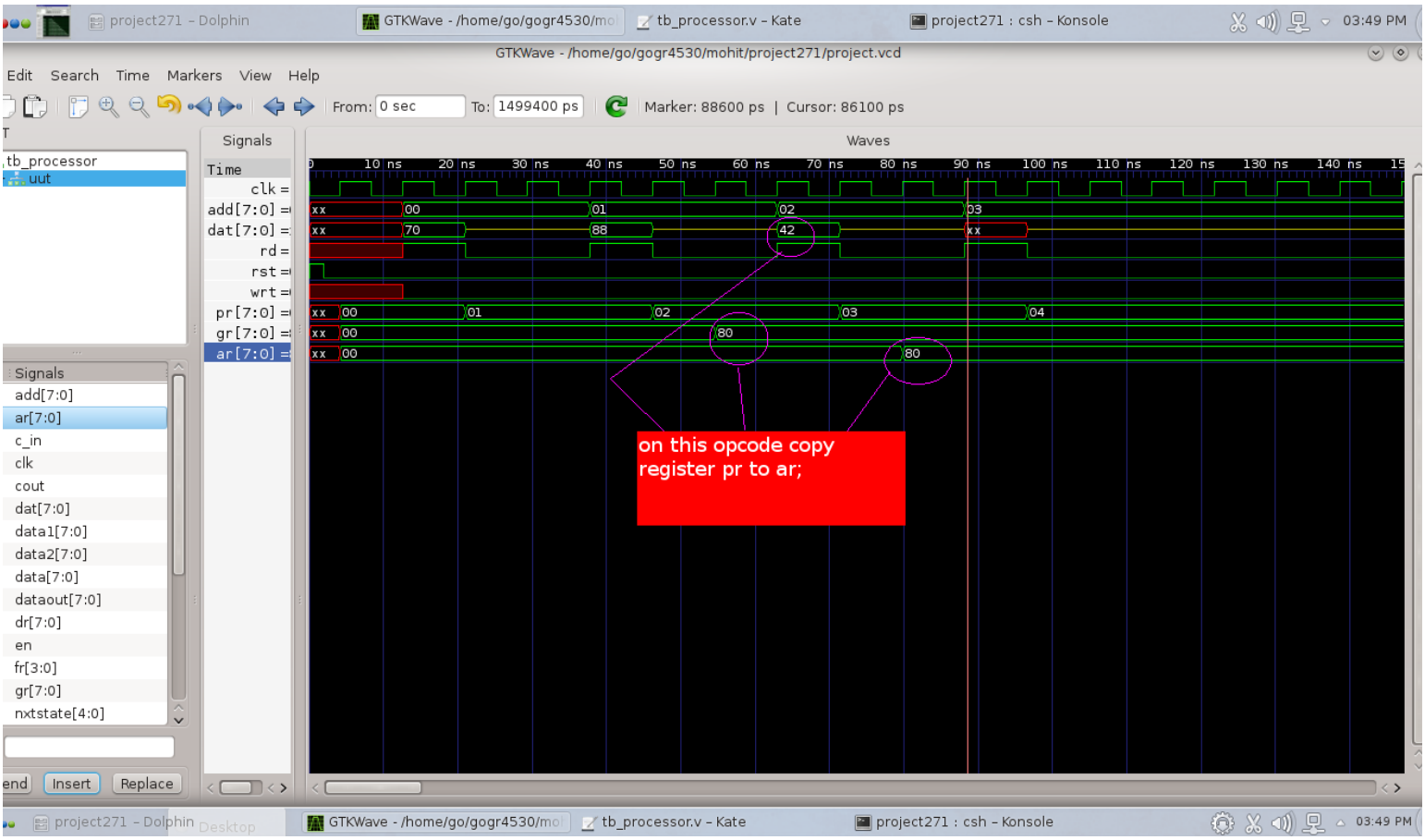
Show simulation/test results for each instruction by providing figures below. Results can be in forms of waveforms and/or text-outputs from system functions (such as `$display` function, etc.). You can do lots of testing but choose the set of data such that you can comprehensively represent your test results within each figure

Figure III.1: Test Result for **`JMP`** Instruction



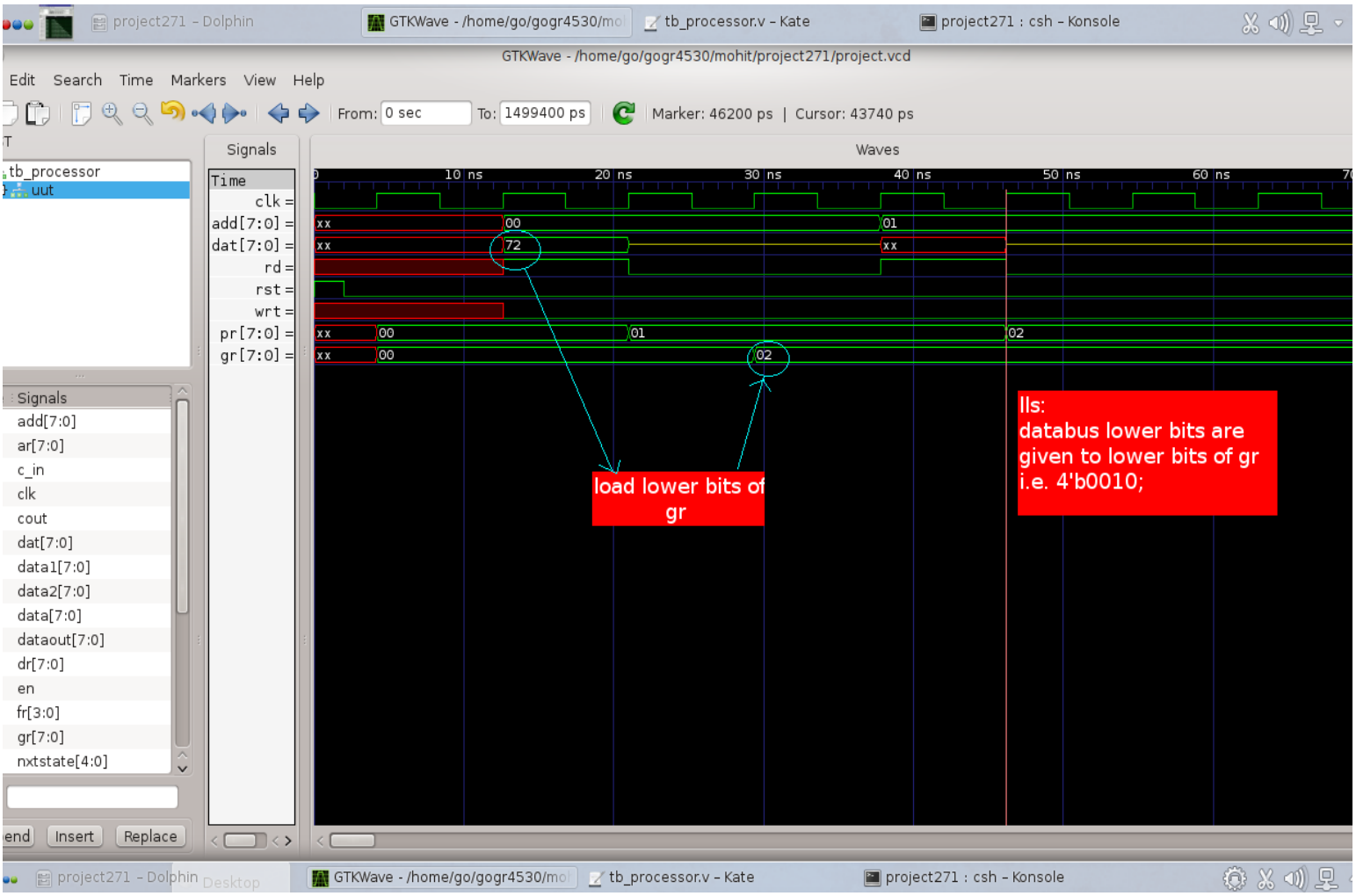
Description :Value in data bus Is 10h
Opcode =10h
Operation JMP i.e. Jump to instruction pointed by AR
Source = AR;
Destination=PR;
Adderess bus has value of AR i.e. 80h and Data bus has value stored at 80h

Figure III.2: Test Result for **CPR** Instruction



Description :Value in data bus Is 42h
Opcode =42h
Operation CPR i.e. copy register
Source = PR;
Destination=AR;
AR has value of PR i.e. 80h;

Figure III.3: Test Result for **`LLS`** Instruction



Description:Value in data bus Is 72h

Opcode =72h

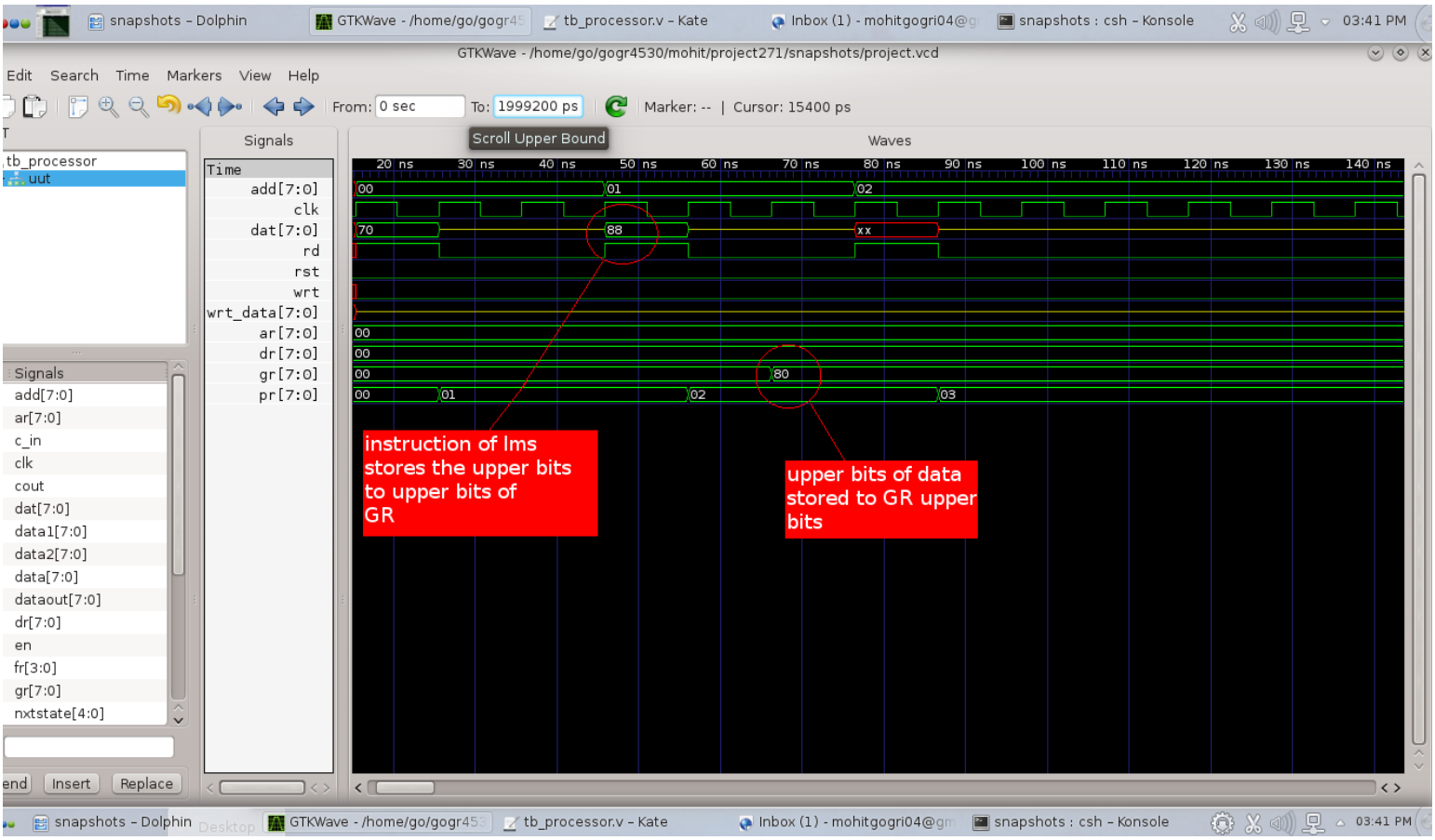
Operation LLS i.e. Load least significant bits of GR

Source = Data

Destination=GR;

LSB of GR has value 2h from Databus.

Figure III.4: Test Result for **LMS** Instruction



Description:Value in data bus Is 88h

Opcode =88h

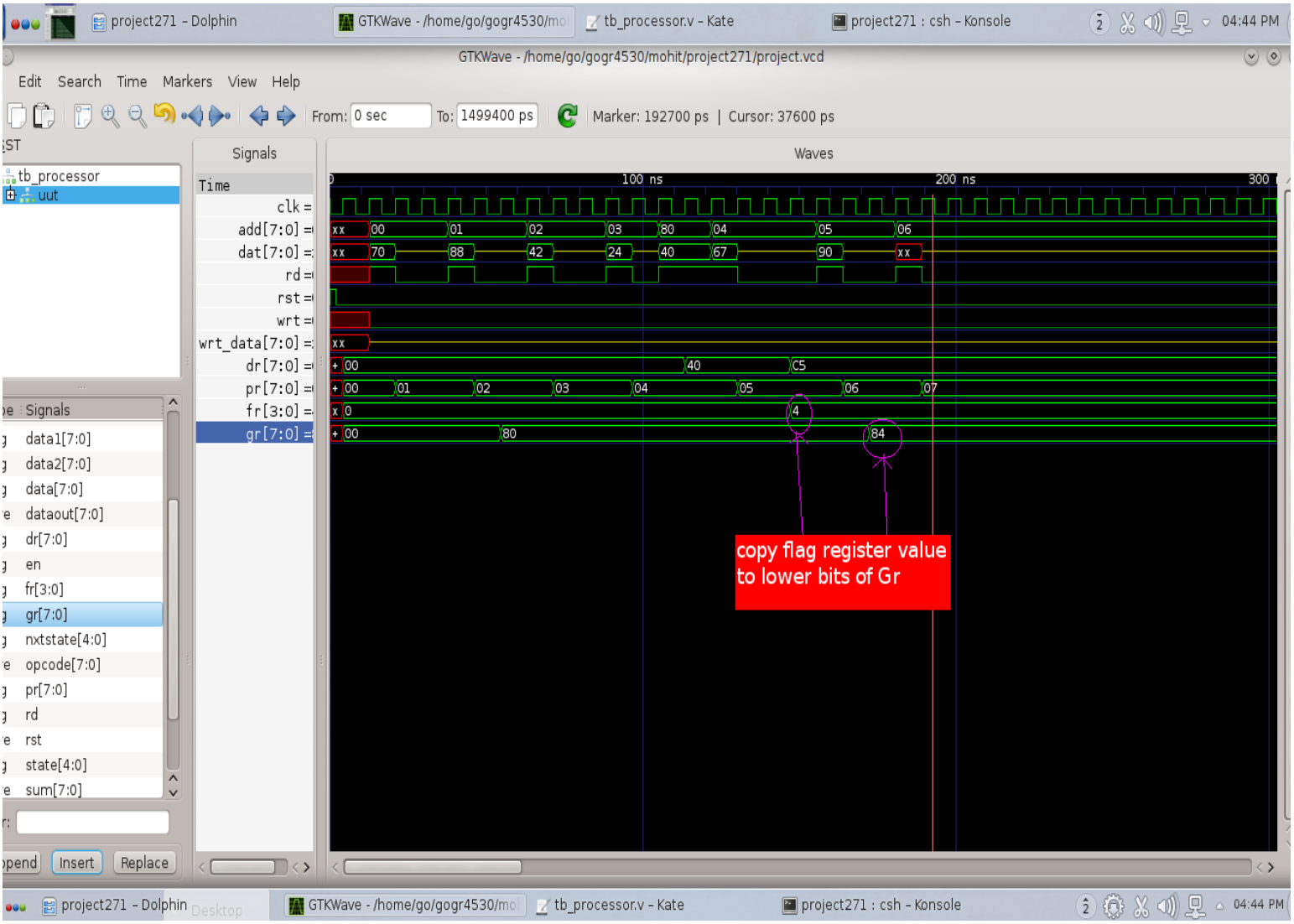
Operation LMS i.e. Load most significant bits of GR

Source = Data

Destination=GR;

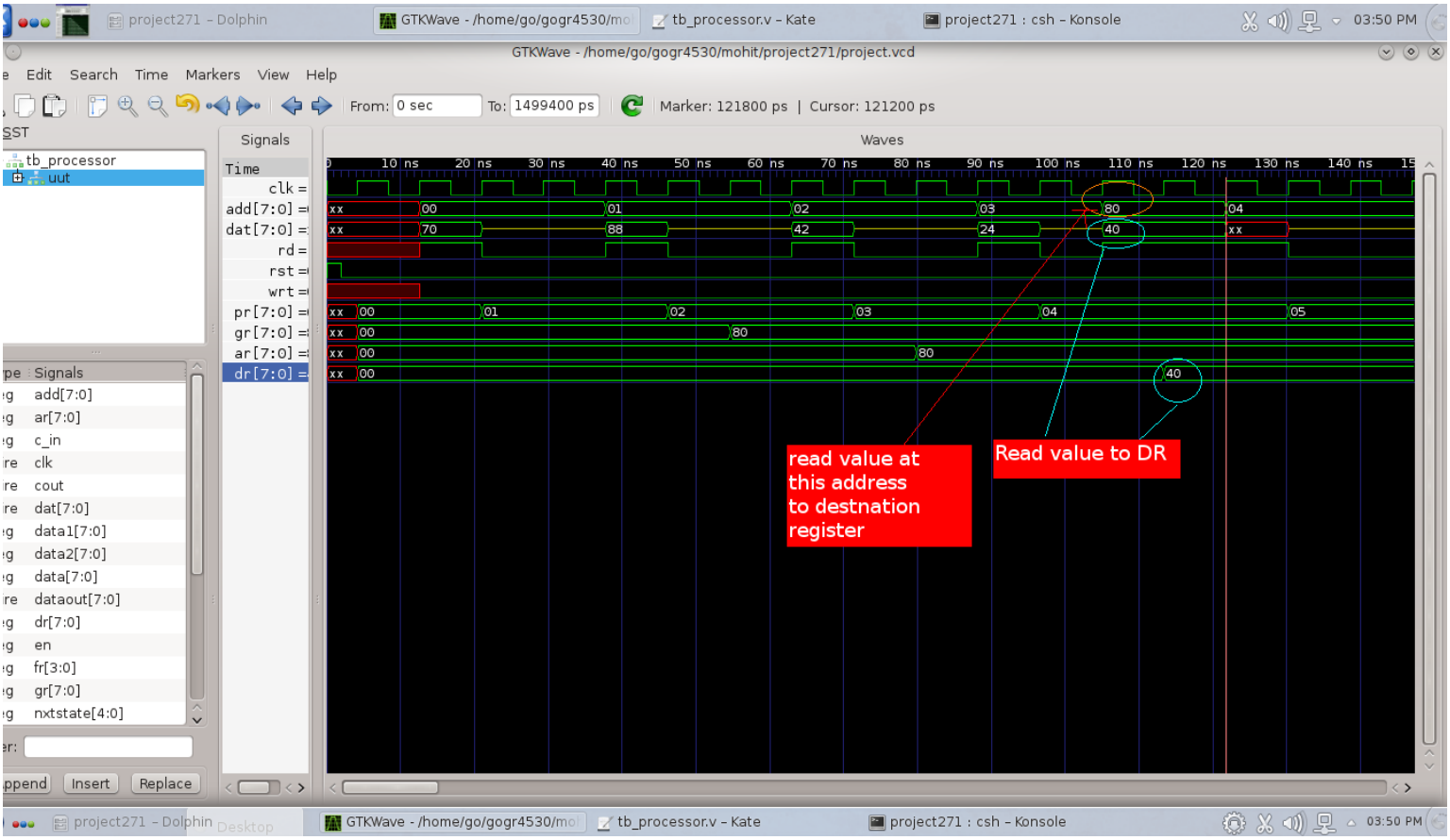
MSB of GR has value 8h from Databus.

Figure III.5: Test Result for **CFR** Instruction



Description:Value in data bus Is 90h
Opcode =90h
Operation CFR i.e. Load least significant bits of GR from FR;
Source = FR
Destination=GR;
LSB of GR has value 4h from FR

Figure III.6: Test Result for **RDM** Instruction



Description:Value in data bus Is 24h

Opcode =24h

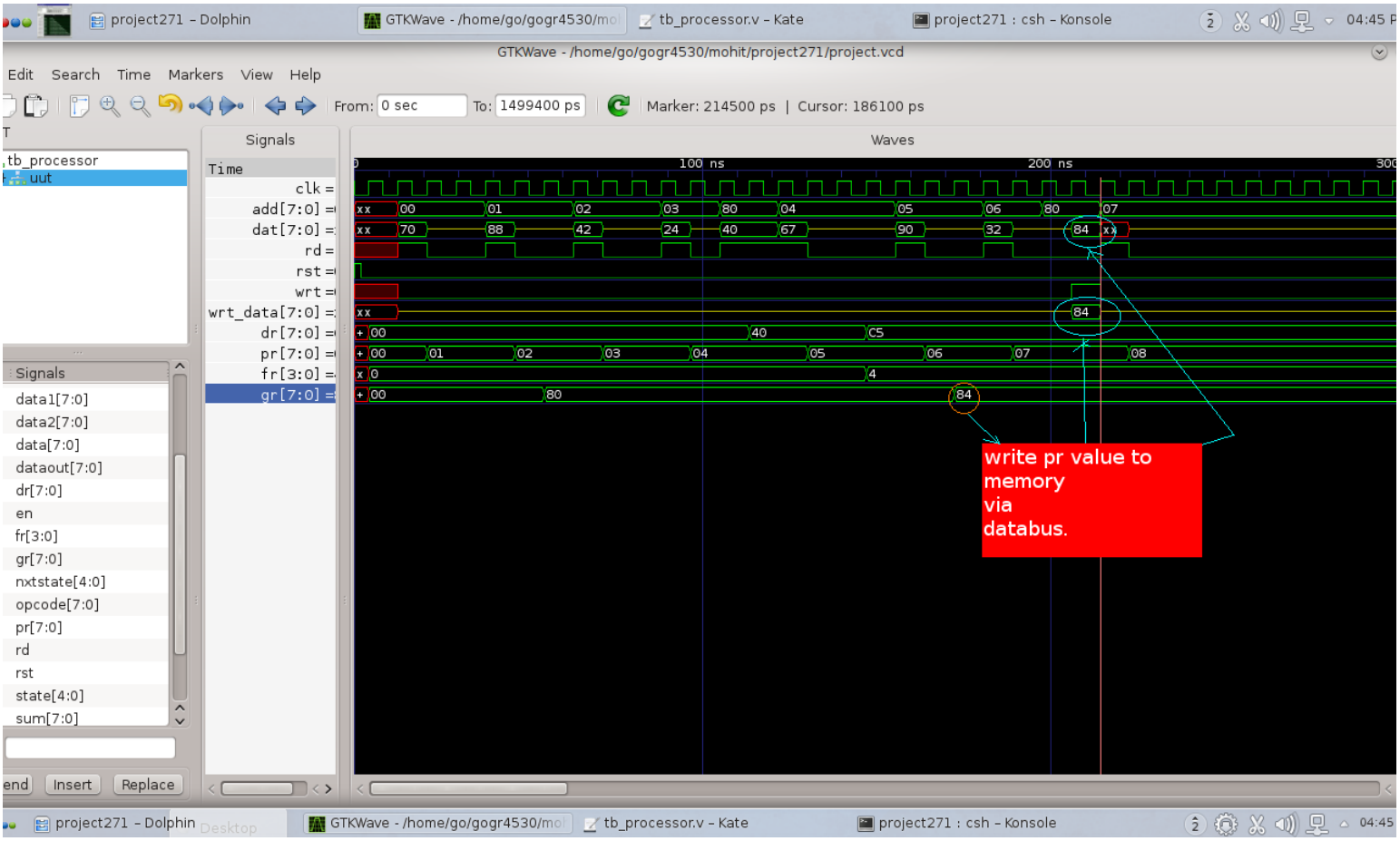
Operation RDM i.e. Read Data from Memory pointed by AR into DR

Source = Data

Destination=DR

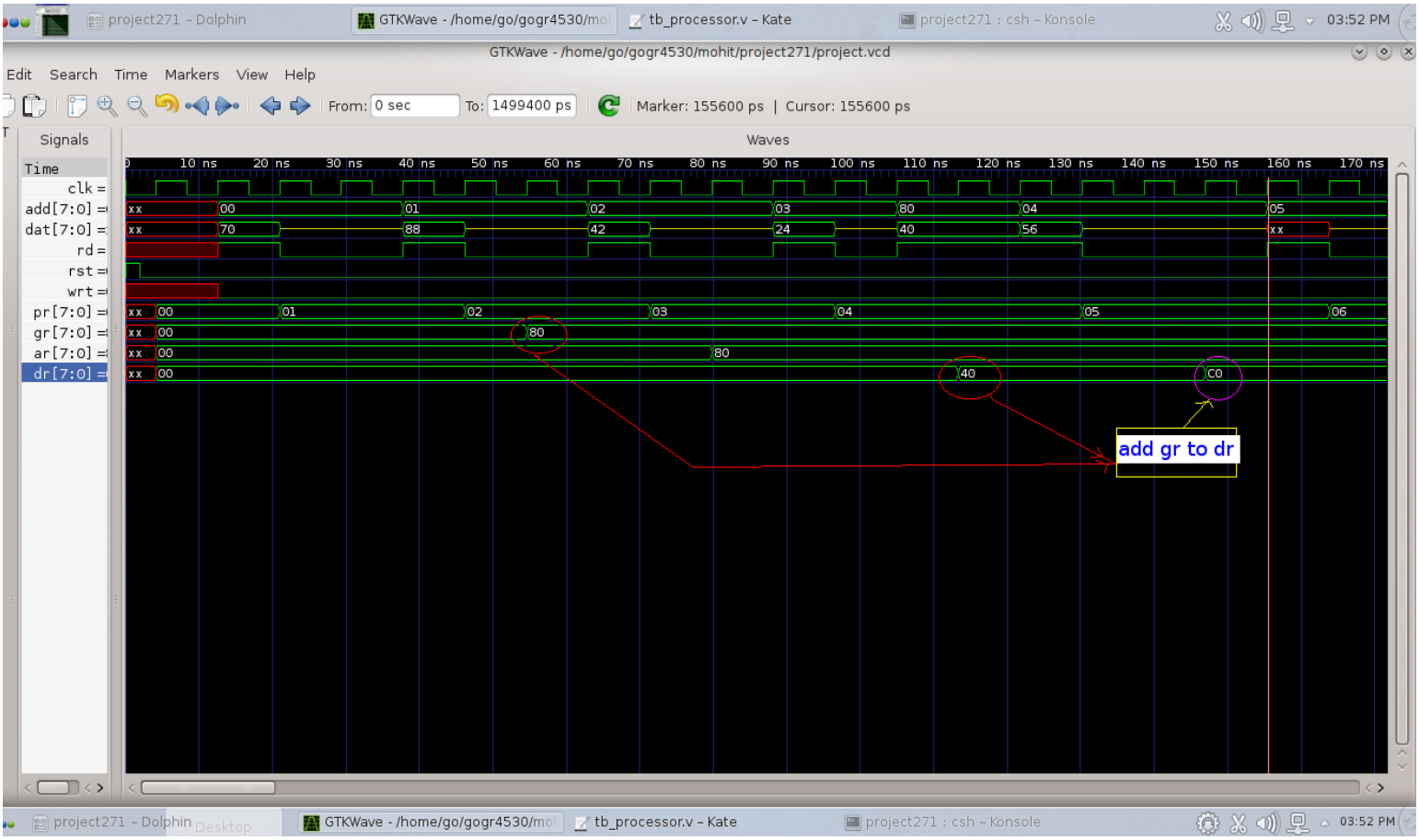
DR has value stored at 80h from Databus i.e 40h

Figure III.7: Test Result for **WRM** Instruction



Description:Value in data bus Is 32h
Opcode =32h
Operation WRM i.e. Write data from Register into Memory pointed by AR
Source = GR
Destination=DataBus
Value of GR is copied to memory address pointed by AR and so databus has 40h.

Figure III.8: Test Result for **ADD** Instruction



Description:Value in data bus Is 56h

Opcode =56h

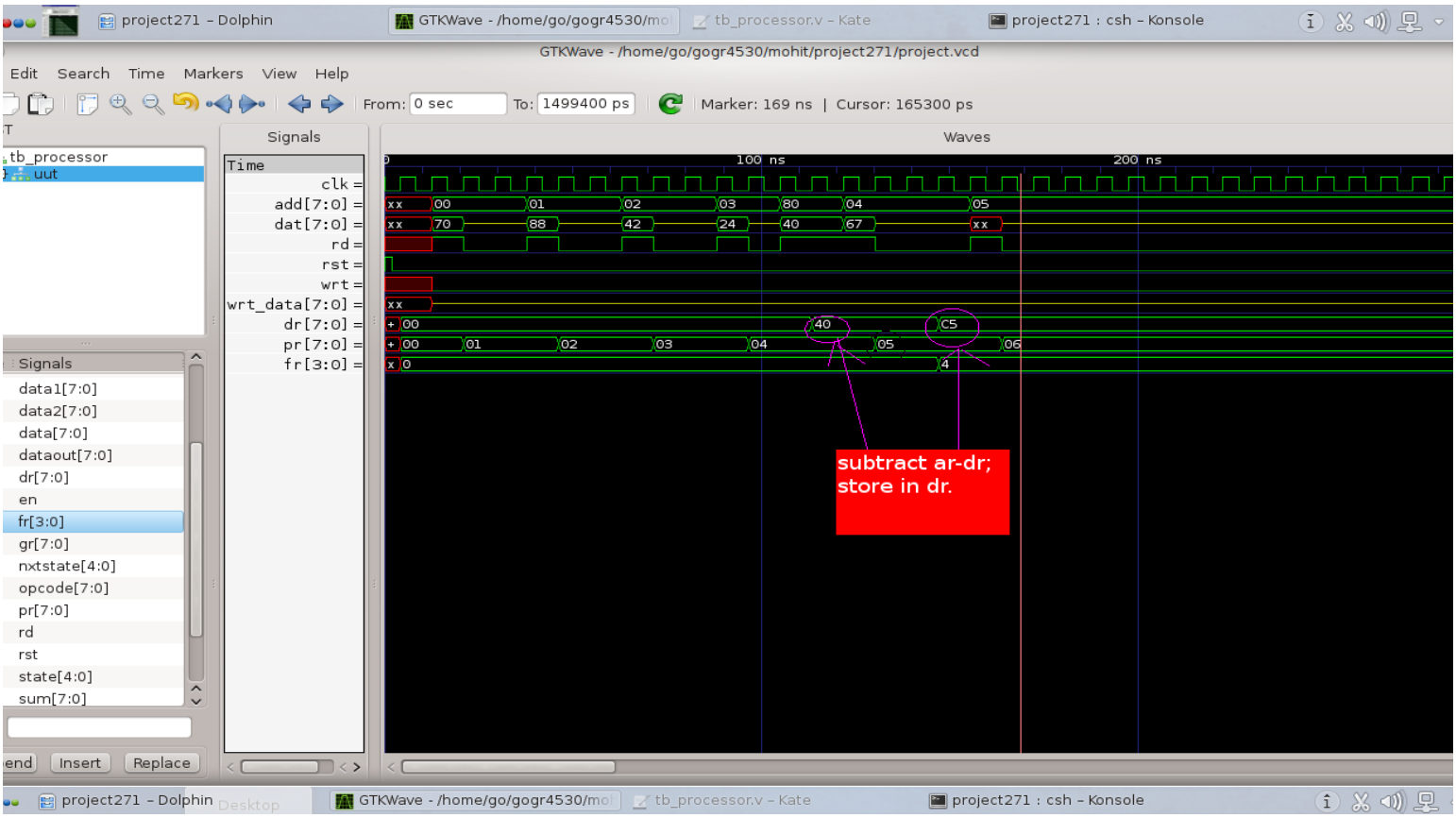
Operation ADD i.e. ADD GR to DR

Source = GR

Destination=DR;

DR has value of addition of DR and GR;

Figure III.9: Test Result for **SUB** Instruction



Description:Value in data bus Is 67h

Opcode =67h

Operation SUB i.e. Subtract 1 register from other.

Source = PR

Destination=DR;

Value of PR is subtracted from DR and value is stored in DR.

IV. RTL-Level Simulations/Tests of Whole Design (processor)

Based on the testbench and test results in testing the entire processor operation as described Section III.2 of the Final Design Project Assignment, complete the tables and show the output results as requested below:

IV.1 Test Program, Machine Codes, and Test Data

Table IV.1: Memory Code Address (in Hex), Test Instruction, and Machine Code (in Hex)

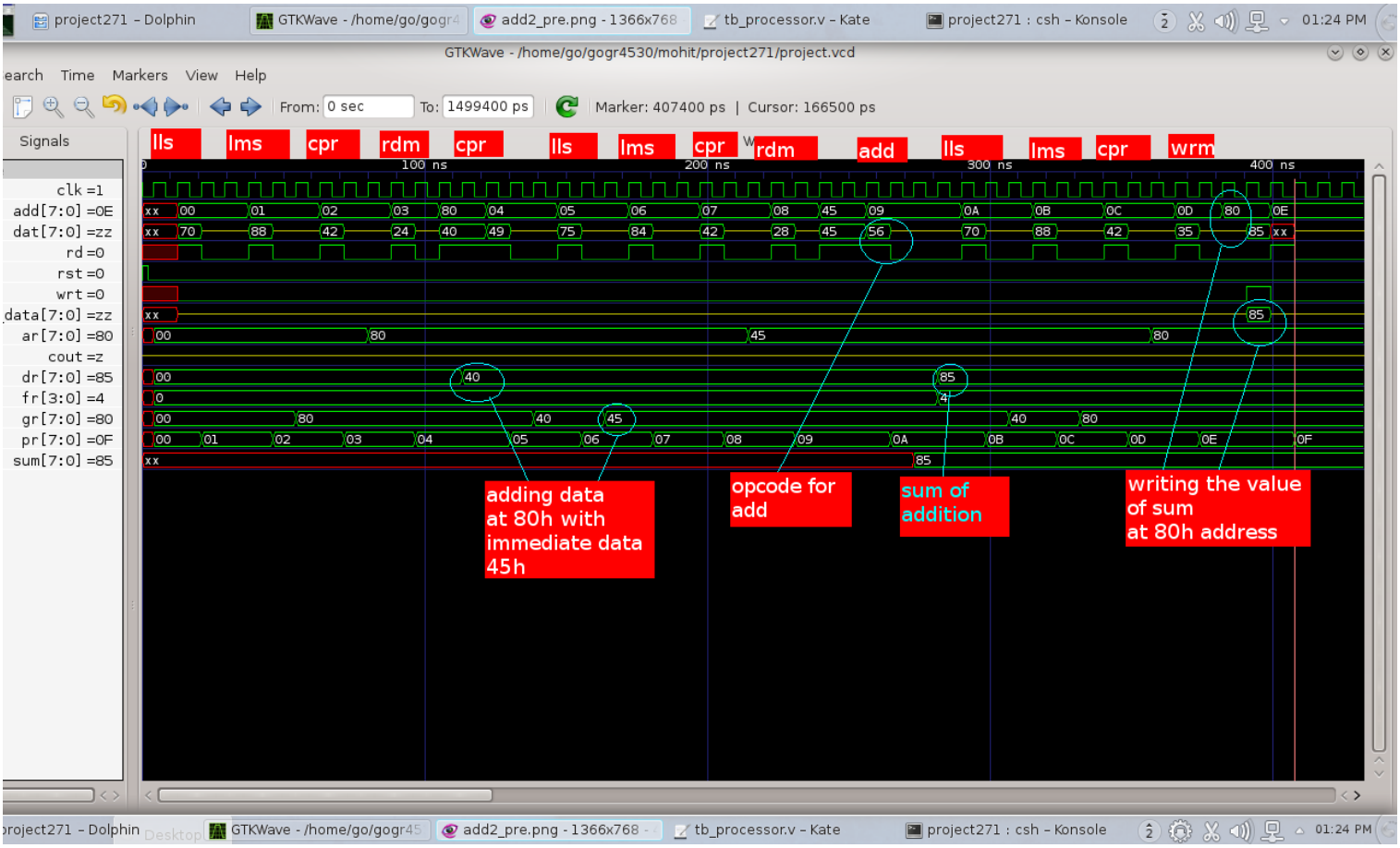
Memory Address	Instruction	Machine Code
00	LLS	70
01	LMS	88
02	CPR AR,GR	42
03	RDM DR	24
04	CPR GR,DR	49
05	LLS	75
06	LMS	84
07	CPR AR,GR	42
08	RDM GR	28
09	ADD DR,GR	56
0A	LLS	70
0B	LMS	88
0C	CPR AR,GR	42
0D	WRM DR	35
0E	LLS	71
0F	LMS	88
10	CPR AR,GR	42
11	RDM GR	24
12	CPR DR,GR	49
13	LLS	71
14	LMS	89
15	CPR AR,GR	42
16	RDM GR	28
17	ADD DR,GR	56
18	LLS	71
19	LMS	88
1A	CPR AR,GR	42
1B	WRM DR	35
1C	LLS	72
1D	LMS	89
1E	CPR AR,GR	42

1F	RDM DR	24
20	LLS	72
21	LMS	88
22	CPR AR,GR	42
23	RDM GR	28
24	SUB DR,GR	66
25	LLS	72
26	LMS	88
27	CPR AR,GR	42
28	WRM DR	35
29	LLS	70
2A	LMS	8F
2B	CPR AR,GR	42
2C	CFR	99
2D	WRM GR	3A
2E	NOP	00
2F	NOP	00
30	LLS	70
31	LMS	80
32	CPR AR,GR	42
33	JMP	10

Table IV.2: Initial Test Data Stored at Memory Addresses (in Hex)

Memory Address	Data	Memory Address	Data
45	45	82	33
80	40	91	66
81	56	92	75

IV.2 Test Results



Description: Value in data bus Is 56h

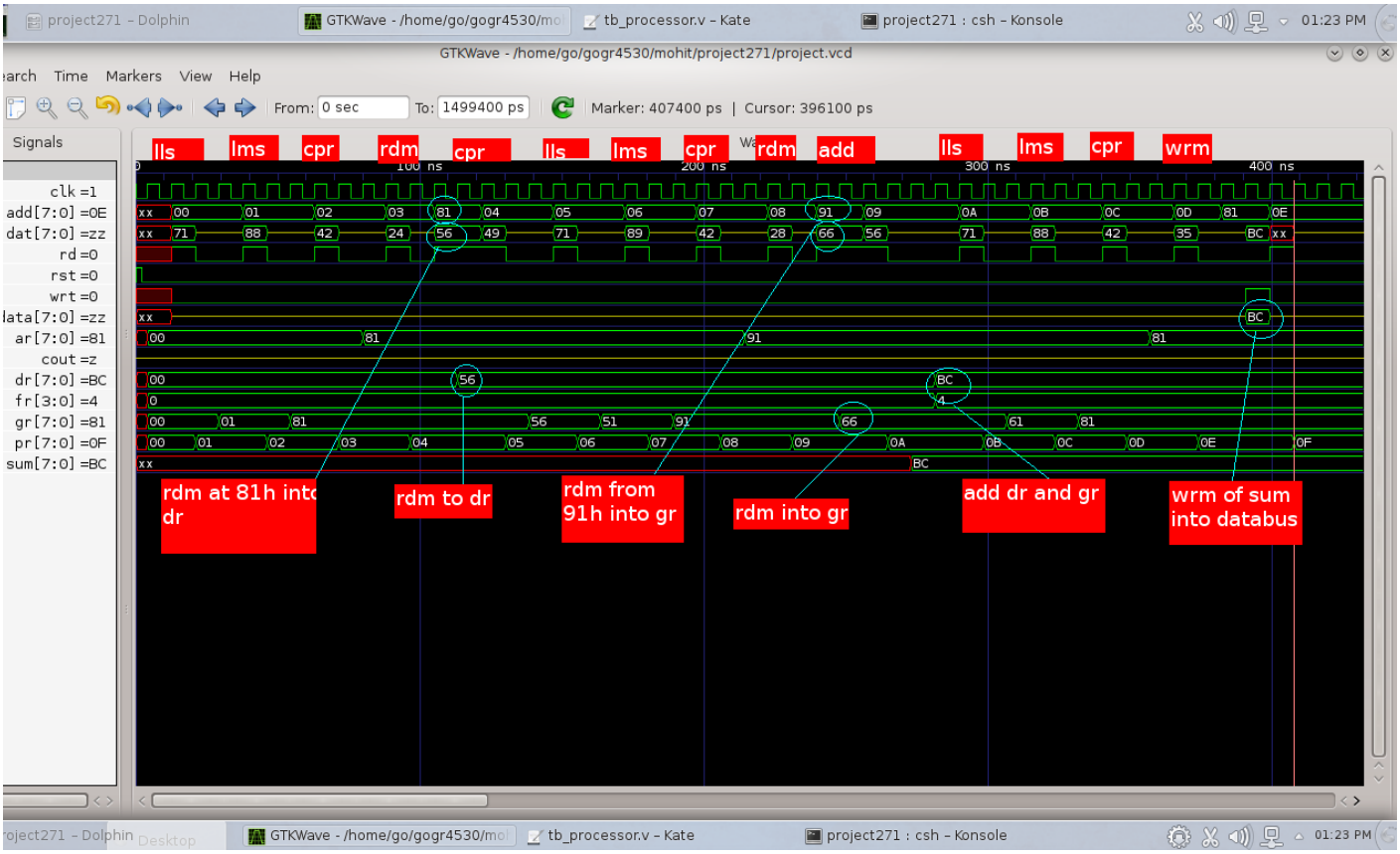
Opcode =56

Operation ADD i.e. ADD the value at 80h to immediate data 45h and store to 80h.

Source = GR

Destination=DR

DR has the sum of GR and DR .



Description:Value in data bus Is 56h

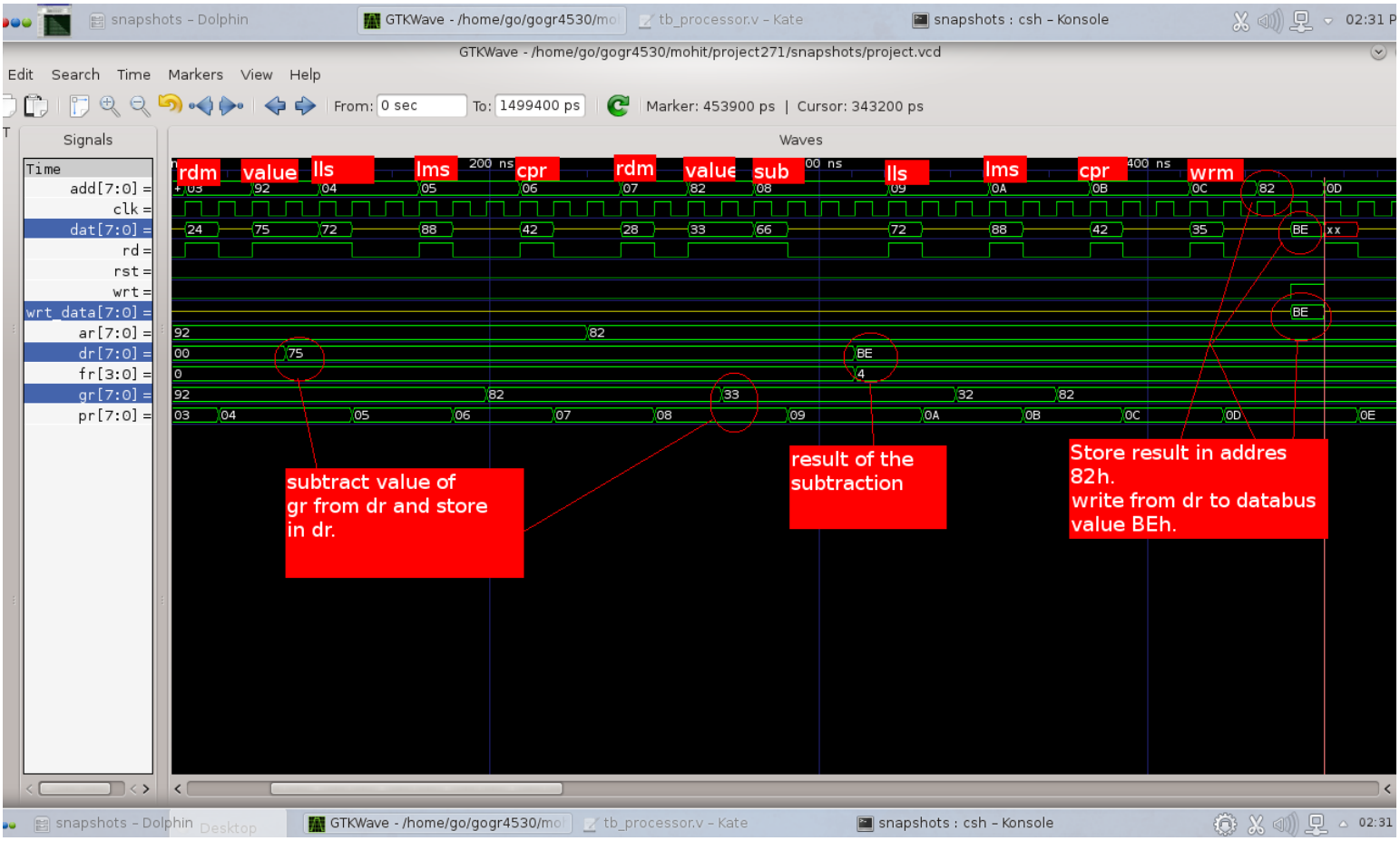
Opcode =56h

Operation ADD i.e. ADD value at 81h and 91h and store in 81h.

Source = GR

Destination=DR;

81h address has value of DR which is sum of DR and GR.



Description:Value in data bus Is 66h

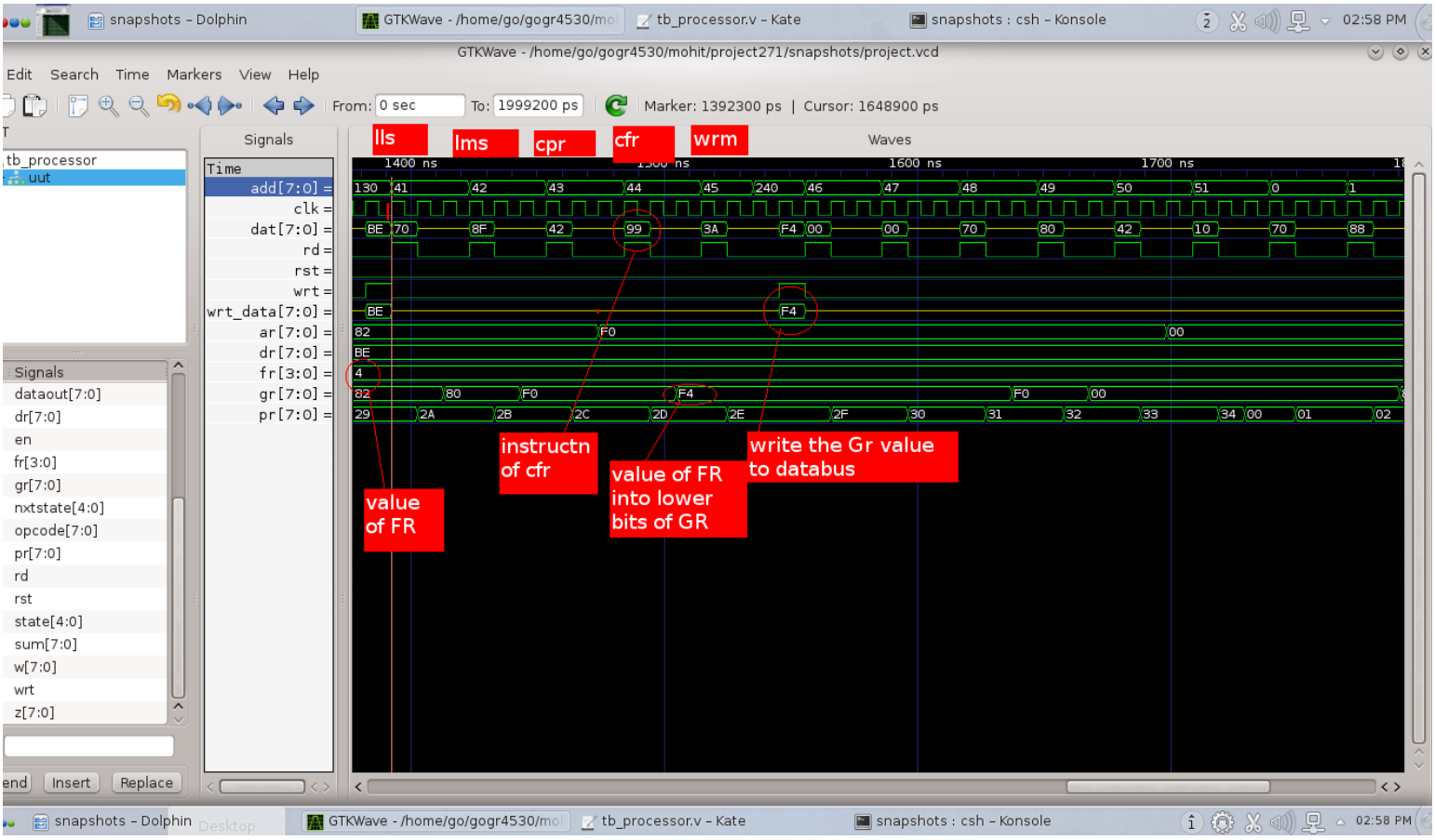
Opcode =66h

Operation SUB i.e. Subtract value at 82h and 92h and store in 82h.

Source = GR

Destination=DR;

DR has value of result of PR-DR and stored to 82h.



Description:Value in data bus Is 99h

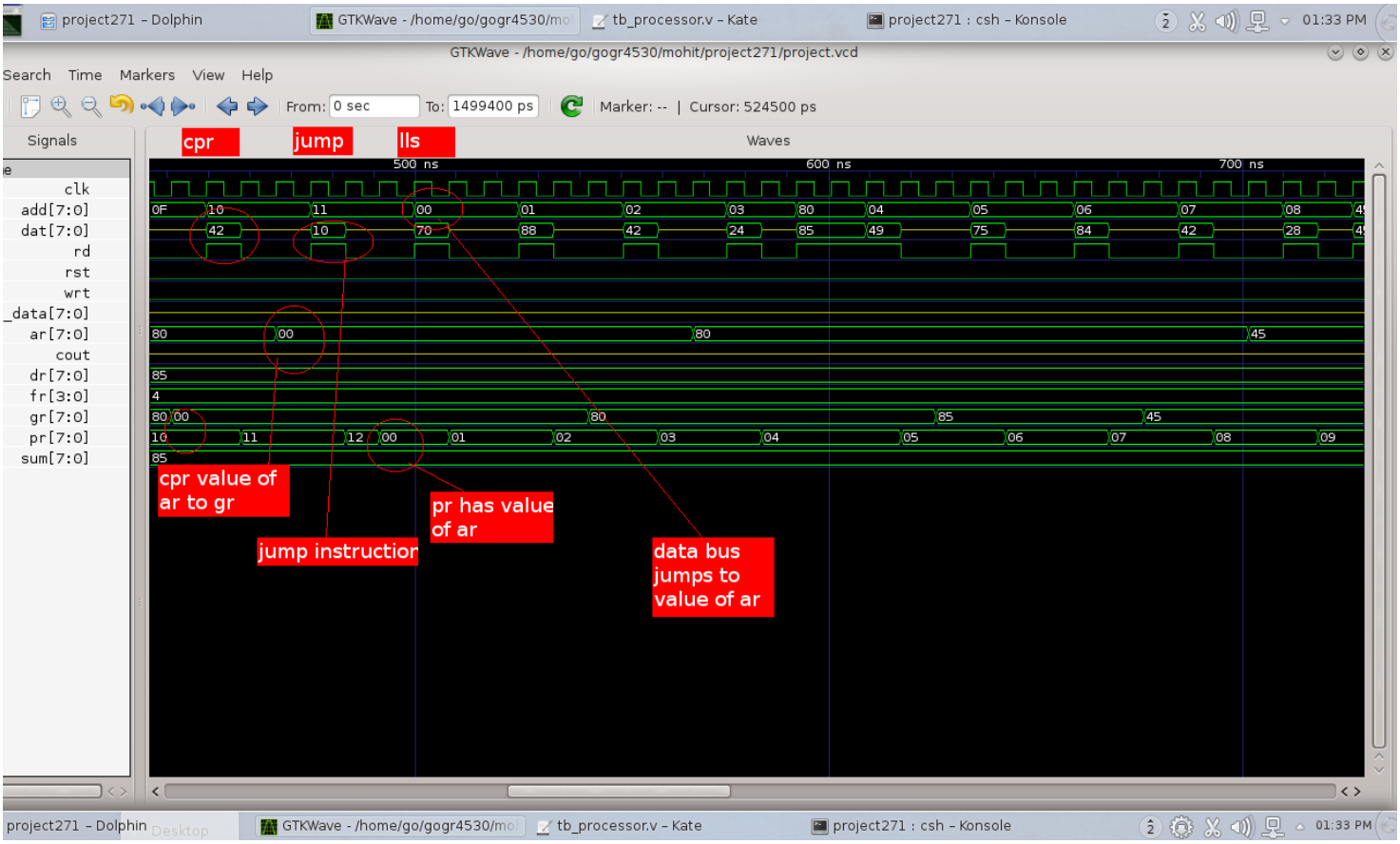
Opcode =99h

Operation CFR i.e. Copy the FR value to LSB of GR.

Source = FR

Destination=GR;

LSB of GR has value 4h from FR.



Description:Value in data bus Is 10h

Opcode =10h

Operation JMP i.e Jump to a value of AR and fetch instruction from that address.

Source = AR;

Destination=PR;

PR has value of AR and it points to that address and fetchs the instruction from memory.

V. Synthesis and Optimizations of Whole Design (processor)

In the table below, show 6 selected trials that you have gone through in synthesizing and optimizing your design. You may try tens of trials, but you need to select 6 to show in the table. Select the trials that can represent your synthesizing and optimizing efforts so that you can conclude that the last trial (trial #6) is the one you finally used as the most optimized processor.

Table V.1: Synthesis Constraints and Results

Trial #	**Constraint settings** (area, clock, delay, etc.)	**Synthesis results** (time slack, area, power, etc.)
1	• Clock=**4.2 ns** • I/p delay=**2 ns,** O/p Delay=**2 ns** • Area=**500**	• Slack=**0.0 (Met)** • Area=**1342** • Power=**3.7125 mW**
2	• Clock=**4.5 ns** • I/p delay=**1.8 ns**, O/p Delay=**1.5 ns** • Area=**500**	• Slack=**0.02 (Met)** • Area=**1383** • Power=**3.5722 mW**
3	• Clock=**4.75** • I/p delay=**1.5 ns**, O/p Delay=**1.5 ns** • Area=**500**	• Slack=**0.0 (Met)** • Area=**1464** • Power=**3.4757 mW**
4	• Clock=**4.2** • I/p delay=**1.5 ns**, O/p Delay=**1.5 ns** • Area=**500**	• Slack=**0.0 (Met)** • Area=**1506** • Power=**4.3011 mW**
5	• Clock=**4.0** • I/p delay=**2 ns**, O/p Delay=**2.5 ns** • Area=**500**	• Slack=**0.01 (Met)** • Area=**1397** • Power=**4.4597 mW**
6	• Clock=**5** • I/p delay=**1 ns**, O/p Delay=**1 ns** • Area=**500**	• Slack=**-0.15 (Violated)** • Area=**1828** • Power=**5.8521 mW**

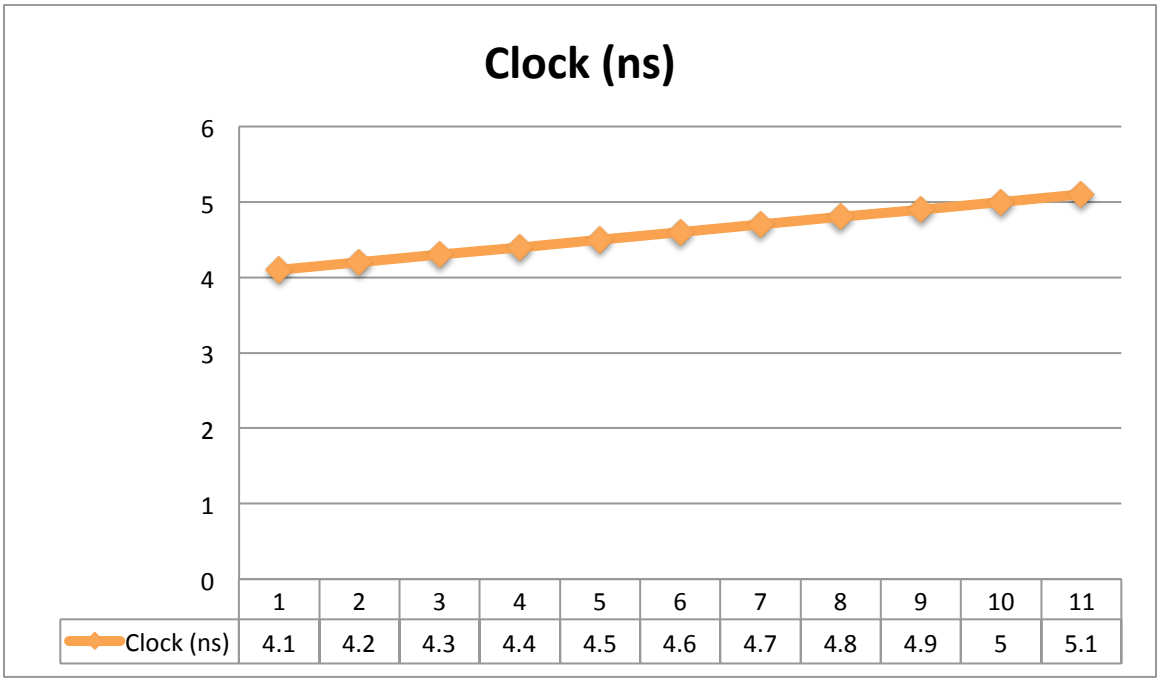
Figure V.1: Area versus Delay/Clock of the Final Design

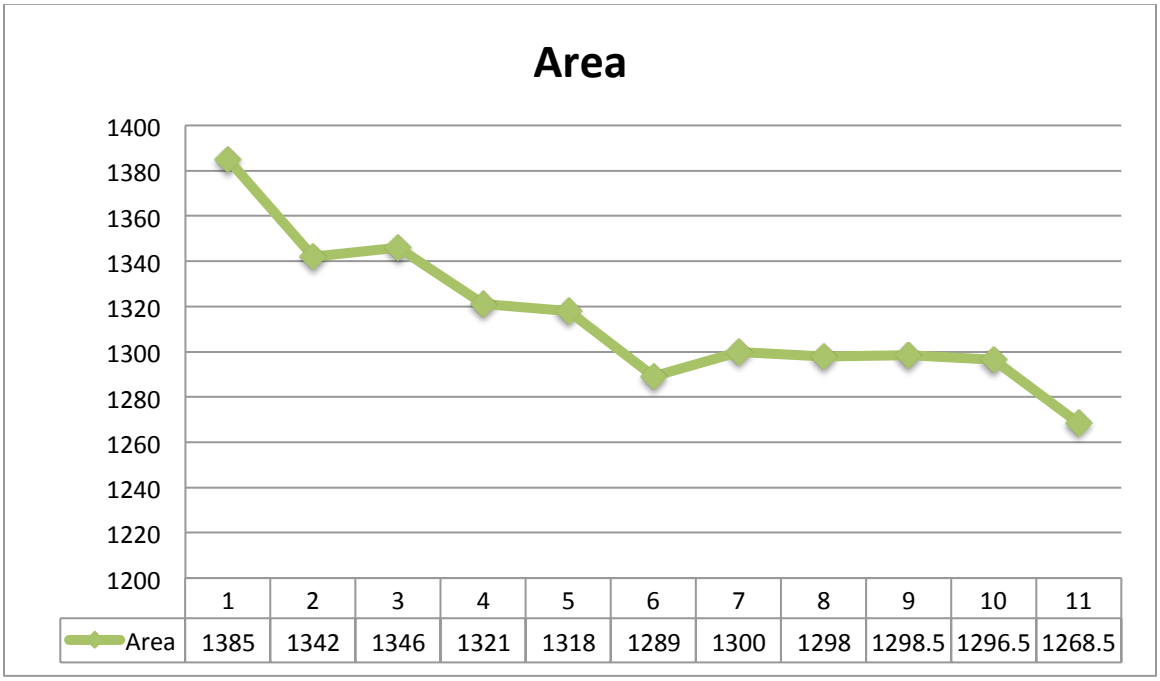
*Briefly discuss and analyze the relationships in timings, areas, other related constraints as well as final specifications of your optimized processor (**less than 50 words**)*

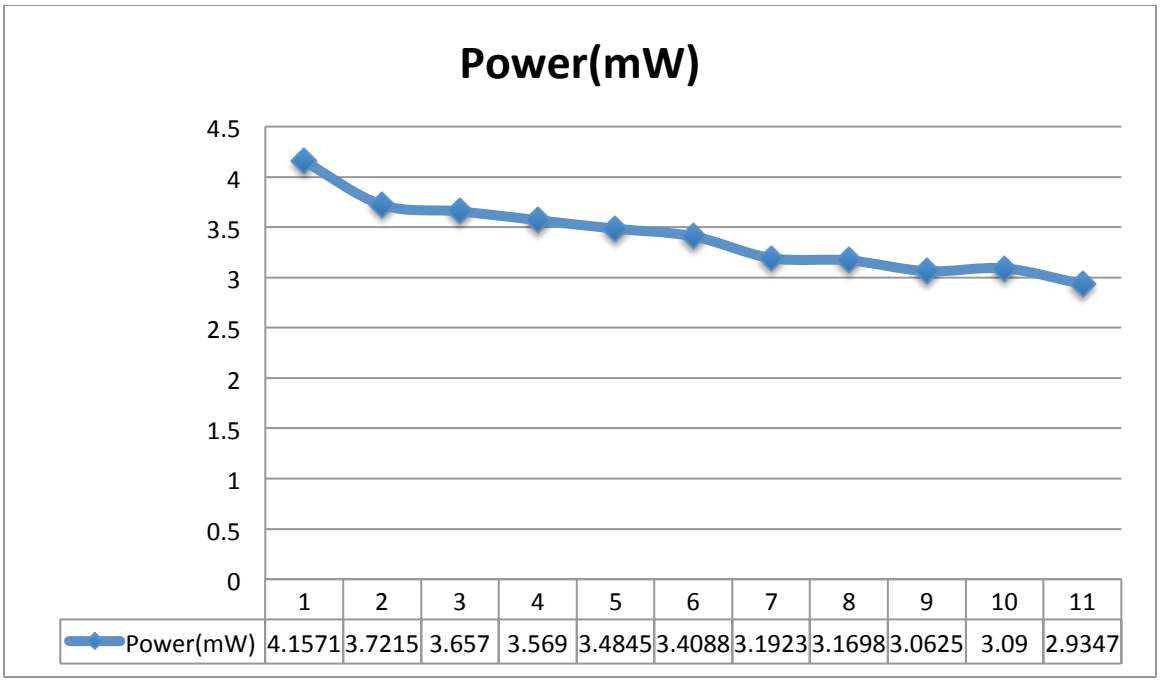
Various constraints were tested during synthesis.Area,clock,slack,power etc. were some of the constraints which were varying with change in value of delay or clock. Below is the graph of change in Area and Power due to change in Clock.Here Slack and Delay are kept constant.From the graph we can see that as the clock period is increase the area and power decreases.Power is related to the area of circuit so as area decreases power automatically goes down.So we can conclude that there is always a **trade off** between **Area** and **Clock.**Both are inversely proptional to each other.
As we can see from below Graph for higher clock area will remain almost same since compiler is not following deterministic approach hence there is some variation in area but this is very less. As we decrease clock time to run processor fast, area try to increase till slack reaches to zero.Various factor such as clock uncertainty, setup and hold time lead to increase clock time as in ideal case they hold some value. Below graph shows that:

- Slack increases as we increase clock timing.
- Area maximizes as slack approaches to zero or at very fast clock and then try to hold constant value as clock increases.
- Power increases as we clock frequency increases due to fast switching of gates .which increases average "r". This relation hold value as we increase Fclock.

$$P_{dynamic} = \sum_{i} \left[0.5 \times C_{Li} \times V_{DD}^{2} \times r_i \times f_{clock} \right]$$

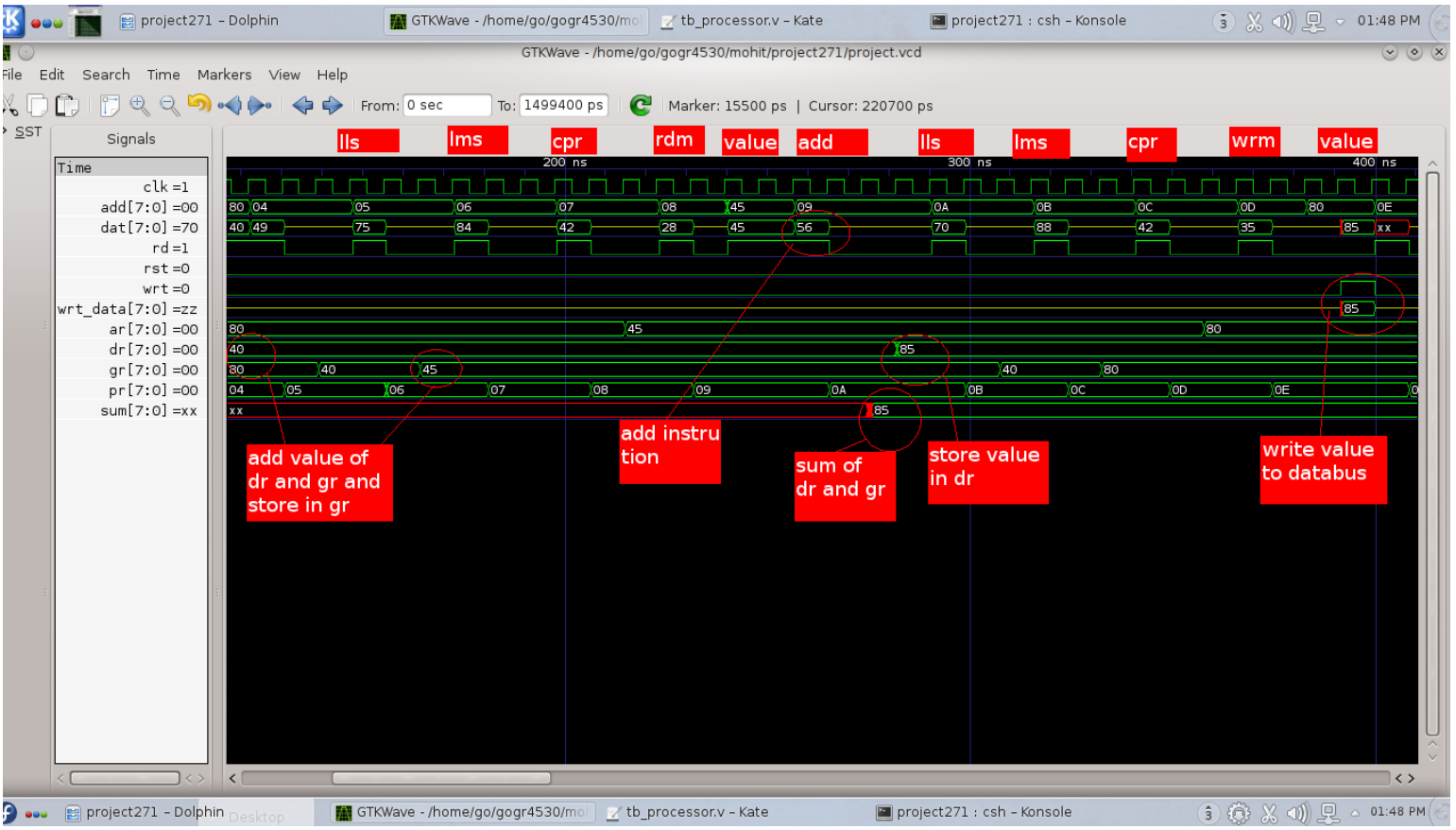

Clock (ns)
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8 9 10 11
Clock (ns) 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5 5.1


Area
1400
1380
1360
1340
1320
1300
1280
1260
1240
1220
1200
1 2 3 4 5 6 7 8 9 10 11
Area 1385 1342 1346 1321 1318 1289 1300 1298 1298.5 1296.5 1268.5

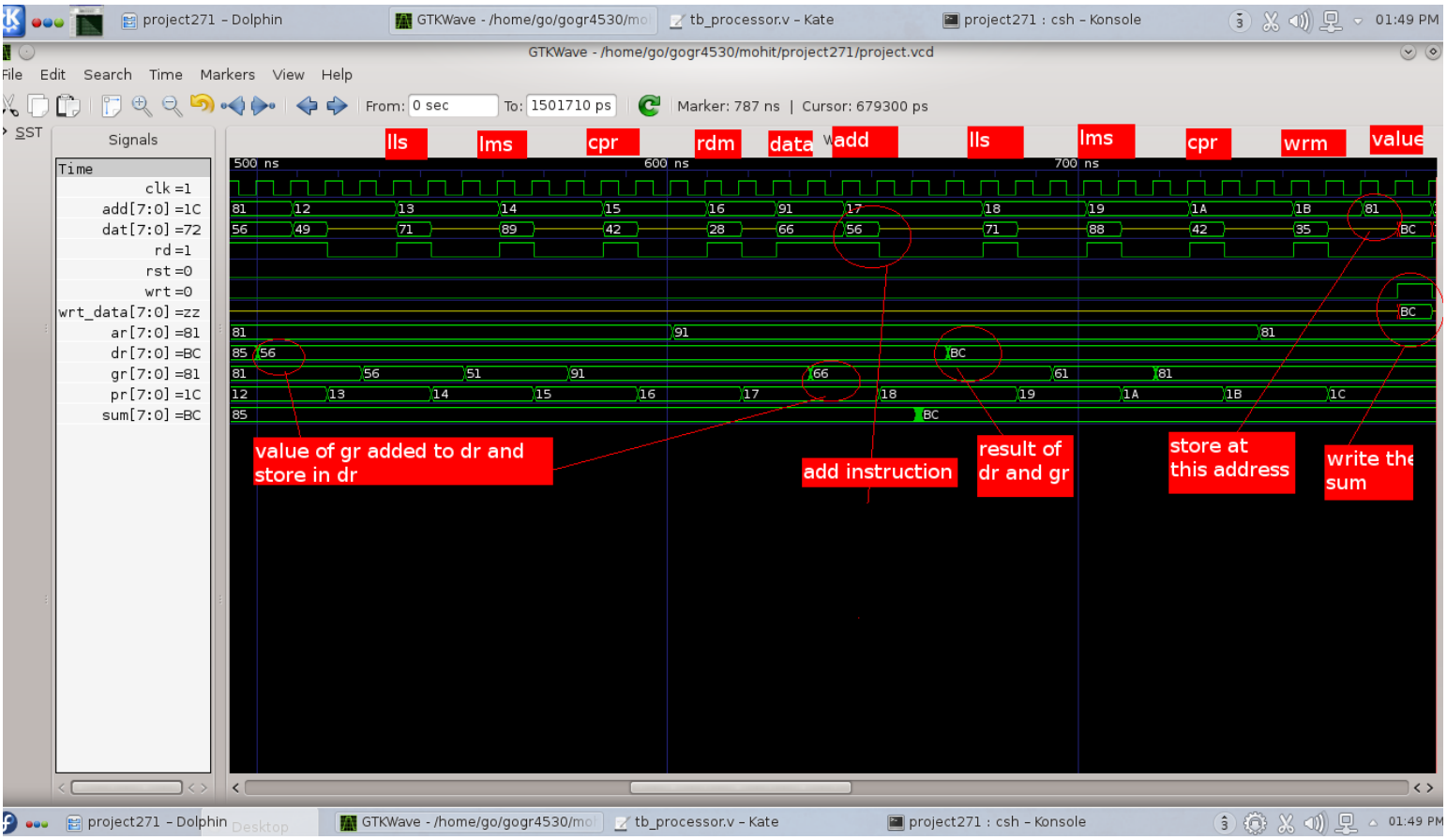

Power(mW)
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1 2 3 4 5 6 7 8 9 10 11
Power(mW) 4.1571 3.7215 3.657 3.569 3.4845 3.4088 3.1923 3.1698 3.0625 3.09 2.9347

VI. Gate-Level Simulations/Tests of Whole Design (processor)

With the same test strategies and test data in Table IV.1 and Table IV.2 above (and as described in Section III.3 of the Final Design Project Assignment,) ***show*** *the test outputs from the final comprehensive test case at the* ***gate-level (netlist).*** *Choose the way to display the results such that you can comprehensively and clearly represent your test results, both the correctness of the logic functions and the timing delays of the instructions at* ***gate-level.***



Description: Value in data bus Is 56h
Opcode =56
Operation ADD i.e. ADD the value at 80h to immediate data 45h and store to 80h.
Source = GR
Destination=DR
DR has the sum of GR and DR .



Description:Value in data bus Is 56h

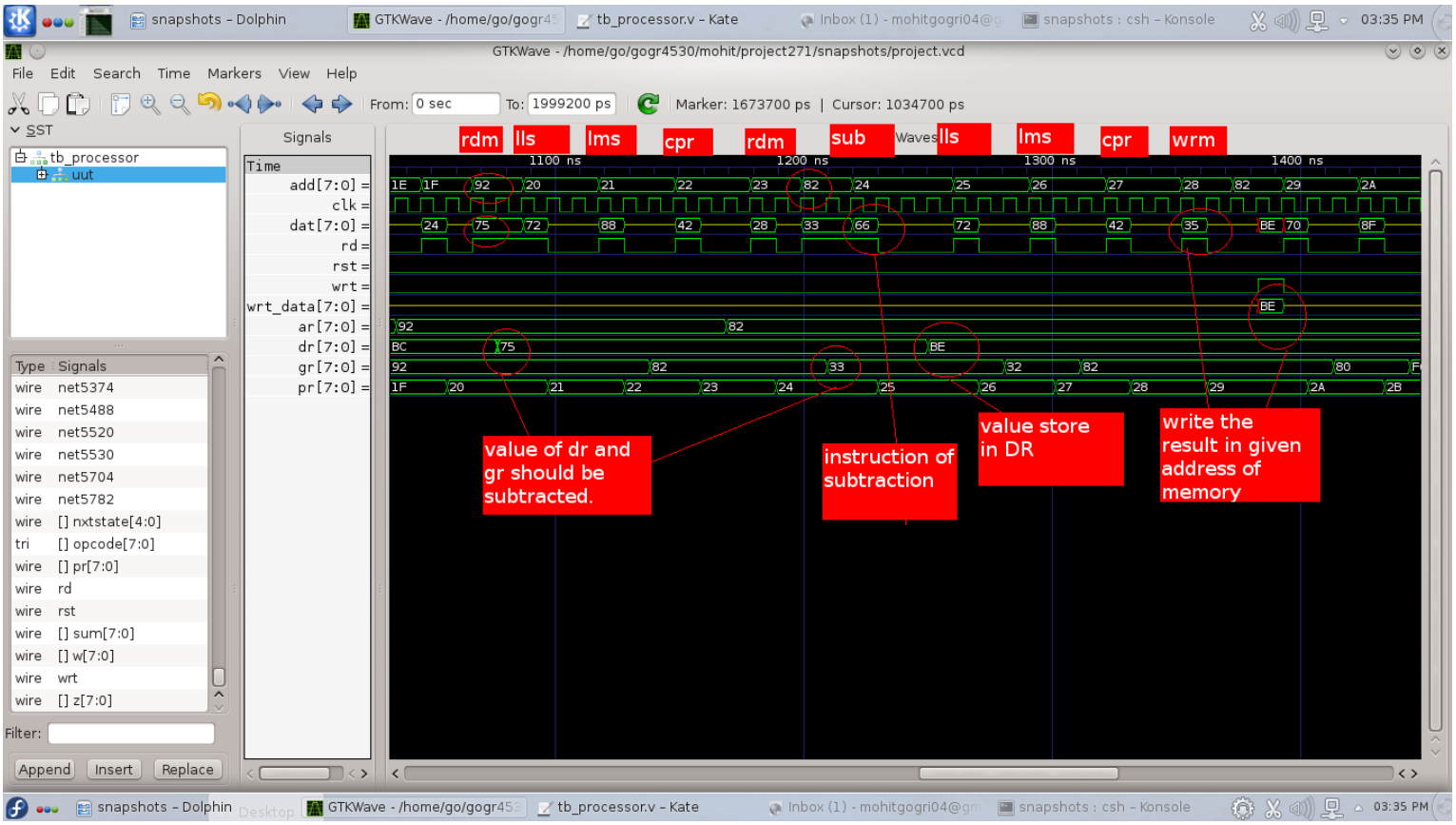
Opcode =56h

Operation ADD i.e. ADD value at 81h and 91h and store in 81h.

Source = GR

Destination=DR;

81h address has value of DR which is sum of DR and GR.



Description:Value in data bus Is 66h

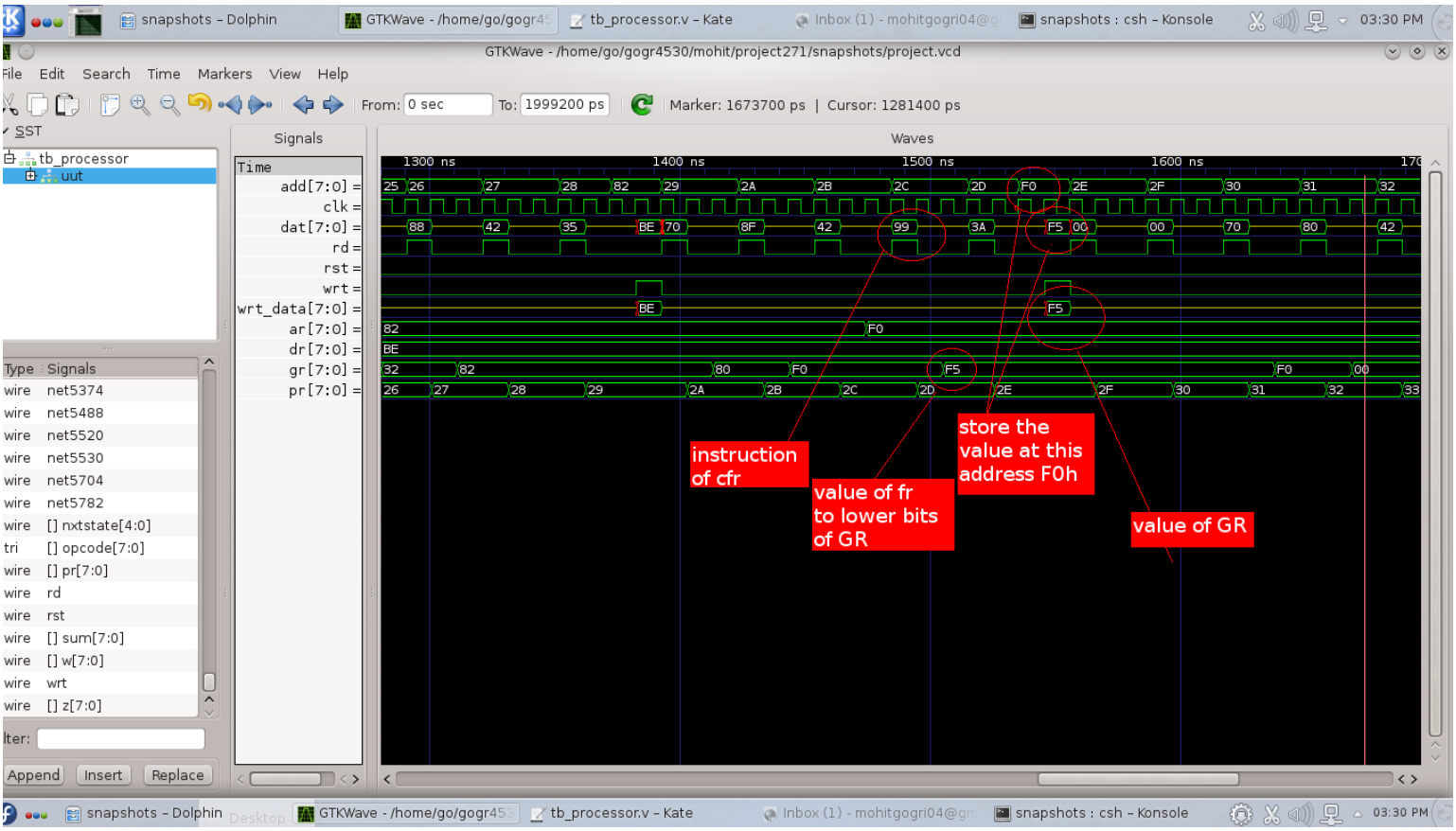
Opcode =66h

Operation SUB i.e. Subtract value at 82h and 92h and store in 82h.

Source = GR

Destination=DR;

DR has value of result of PR-DR and stored to 82h.



Description:Value in data bus Is 99h

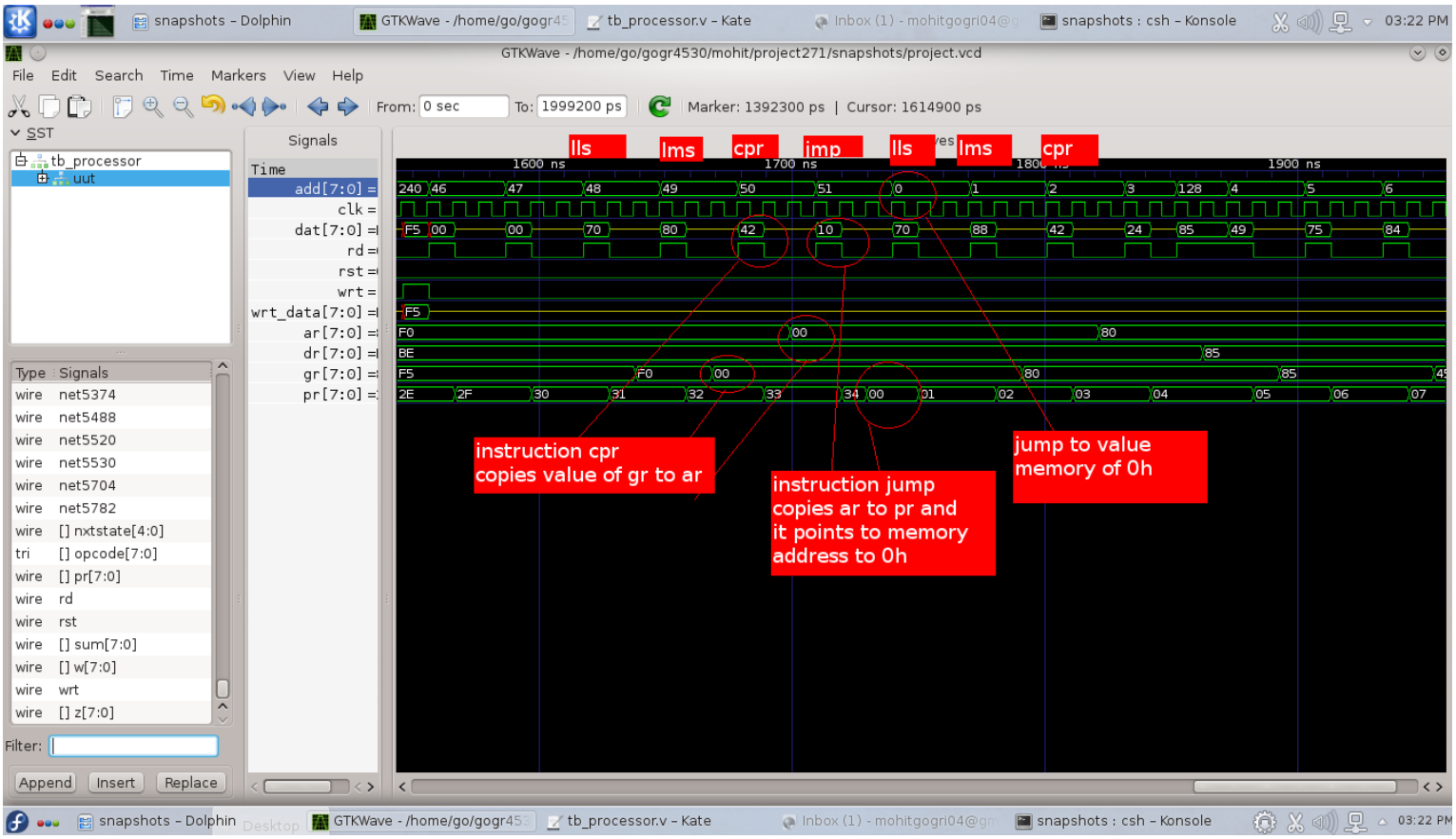
Opcode =99h

Operation CFR i.e. Copy the FR value to LSB of GR.

Source = FR

Destination=GR;

LSB of GR has value 5h from FR.



Description:Value in data bus Is 10h

Opcode =10h

Operation JMP i.e Jump to a value of AR and fetch instruction from that address.

Source = AR;

Destination=PR;

PR has value of AR and it points to that address and fetchs the instruction from memory.

VII. Conclusion

Give conclusion about the entire project (**less than 50 words**)

After designing 8 bit scalar microprocessor with various instructions such as read, write, add, sub, lls ,lms etc I have came across many new things and also I variation in many things which I already know . During working of project I lean about relationship with area , power and timing by keeping in mind various constraint such as hold time, set-up time , skew and jitter.I learn interfacing with different components such as memory etc Vcs tool help me during simulation and design-vision for synthesis the program. Working on Design Compiler helped me to understand the schematic and synthseis part.It was a good experience of learning the post synthesis debugging and verification of all the aspects.Some famous things which I came through during synthesis error removing are following:

1: When modeling sequential logic, use nonblocking assignments.
2: When modeling latches, use nonblocking assignments.
3: When modeling combinational logic with an always block, use blocking assignments
4: When modeling both sequential and combinational logic within the same always block, use non blocking assignments.
5: Do not mix blocking and nonblocking assignments in the same always block
6: Do not make assignments to the same variable from more than one always block.
7: Do not call module in case or if statements.instead task or function can be used.
8: If the variables are not defined properly then working of design will be ruined.

All these guidelines help me build project so that my simulated program can effectively be synthesize . Hence overall experience about this project is that I learn lot about digital design techniques , VCS tool, Design vision and Verilog under guidance of **Prof Thuy Le.**

Appendix A

A.1 Contents from EDA Tool Configurations and Setup Files

```
The sample setup file *//*for TOSHIBA Verilog Sign Off System*/
Please modify this file and include in your setup file. */
/*1 Add <DesignCompiler install path>/libraries/syn to   */
/*the search_path. */
/*2 Set <install dir> to the directory name of*/
/*installation of this design kit. */
/*3 Set<special> to the request based library name. */
/*Add more libraries if needed.                    */
/*4 Rename this file to .synopsys_pt.setup or          */
/*synopsys_dc.setup depands on which tool is used.    */
/*Dochange_names command like an example below.         */
/************************************************************/

/*  === tc240c sample === *-
tsb_lib_path = { "<install dir>","<install dir>/tc240c" }
search_path = search_path + tsb_lib_path

link_library={"*"tc240c.db_WCCOM25tc240ct_io_macro.db  <special>.db_MAX }
target_library = { tc240c.db_WCCOM25 }

set_min_library:tc240c.db_WCCOM25 -min_version  tc240c.db_BCCOM25
set_min_library:<special>.db_MAX -min_version  <special>.db_MIN
symbol_library = { tc240c.workview.sdb }
-*  === tc240c sample end === */

/* --- for asynnchronous set/reset path and timing checks (with recovery check) : 26-JAN-2000 -
-- */
enable_recovery_removal_arcs = true


/*************************/
/* Variables for compiling */
/*************************/
/* ----------- Make ports avoid in connecting to the same net ----------- */
/* compile_fix_multiple_port_nets = true */
set_fix_multiple_port_nets -all

/*********************************/
/* Variables for writing Verilog HDL */
/*********************************/
/* ----------- 1. Bus controls ----------- */
hdlout_internal_busses = "false" /* <-- For D.C. v1997.01 or later */
```

verilogout_single_bit = "false"

A.2 Scripts and/or Commands Used for Simulation and Synthesis

Command used for Simulation : vcs +v2k tb_processor.v processor8bit.v;./simv
Command used for Pre Synthesis: dc_shell -xg -f synthesis.script | tee presynthesis.txt
Command used for Post Synthesis: vcs +v2k -y /export/apps/toshiba/sjsu/verilog/tc240c/ +libext+.tsbvlibp tb_processor.v project_netlist.v ; ./simv
Command for Gtk wave : Gtkwave filename.vcd&
Command for observing the schematics : Design_vision&

Scipt file :

```
#set link_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25}
#set link_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_BCCOM25}
set link_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_WCCOM25}
#set target_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_NOMIN25}
#set target_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_BCCOM25}
set target_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_WCCOM25}
set symbol_library {/apps/toshiba/sjsu/synopsys/tc240c/tc240c.workview.sdb}
set synthetic_library {dw_foundation.sldb standard.sldb}
set_min_library /apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_WCCOM25 -min_version
/apps/toshiba/sjsu/synopsys/tc240c/tc240c.db_BCCOM25

read_verilog processor8bit.v

current_design processor8bit
link
check_design
create_clock clk -name clk -period 5.10000
set_propagated_clock clk
set_clock_uncertainty .25 clk


set_max_delay 2 -from [all_inputs]
set_max_delay 2 -to [all_outputs]
#set_max_delay 2 -to sum
#set_load 160 sum
set_max_area 1300
compile -map_effort high
report_cell
report_net
update_timing
report_timing -max_paths 10
report_area
report_power
write -hierarchy -format verilog -output project-_netlist.v
quit
```

Appendix B
Completed Verilog Source Codes and Testbenches

Verilog Code:

```
module processor8bit (input rst,
                      input      clk,
                      inout wire [7 :0] dat,
                      output reg [7:0] add,
                      output reg         rd,
                      output reg         wrt);

  //-------------Declaring Registers----------------------//

  reg [7:0]   ar,dr,gr,pr;
  reg [3:0]   fr;
  reg         en,c_in;
  reg  [4:0]  state,nxtstate;
  reg [7:0]   data;
  reg [7:0]   w,z;
  reg [7:0]   data1,data2;

  //------------ Declaring Intermideate signals-----------//

  wire [7:0]  sum;
  wire [7:0]  dataout;
  wire [7:0]  opcode;
  wire        cout;

  //-------------Declaring Parameters----------------------//

  parameter reset=4'b0000;
  parameter fetch=4'b0001;
  parameter update=4'b0010;
  parameter execute=4'b0011;
  parameter load=4'b0100;
  parameter read=4'b0101;
  parameter write=4'b0110;
  parameter nop=4'b0000;
  parameter jmp=4'b0001;
  parameter rdm=4'b0010;
```

```
  parameter wrm=4'b0011;
  parameter cpr=4'b0100;
  parameter adder=4'b0101;
  parameter sub=4'b0110;
  parameter lls=4'b0111;
  parameter lms=4'b1000;
  parameter cfr=4'b1001;

  //--------------Calling Alu-----------------------------//

  alu(.sum(sum),.cout(cout),.w(w),.z(z),.c_in(c_in),.en(en),.clk(clk));

  //---------------Operation with Databus ----------------//

  assign opcode=rd?dat:8'bz;
  assign dat=wrt?data:8'bz;

  //---------------Execution of Processor-----------------//

  always @ (posedge clk or posedge rst)
  begin
  if(rst)
  nxtstate<=4'b0000;
  else
  begin
  state<=nxtstate;
  $monitor( " value of Ar=%h,Dr=%h,GR=%h,Pr=%h  " ,ar,dr,gr,pr);
  case(nxtstate)
 //-------------------Reset State--------------------------//
  reset:begin  pr<=8'b0;
                ar<=8'b0;
                gr<=8'b0;
               dr<=8'b0;
               fr<=4'b0;
               data<=8'b0;
               nxtstate<=fetch;
       end
//---------------------Fetch State-------------------------//
  fetch:begin  add<=pr;
                rd<=1'b1;
```

```
                wrt<=1'b0;
                nxtstate<=update;
                end
 //--------------------Update and Decode  state--------------------------//
  update:begin
            rd<=1'b0;
            pr<=pr+1;
            data1=opcode[3:0];
            data2=opcode[7:4];
            nxtstate<=execute;
            end
//---------------------Execution State-----------------------------//
  execute:begin
            case(data2)

        nop: begin                                          // No Operation Case
                $display ("No operation");
                nxtstate<=fetch;
                end

        jmp: begin                                          // Jump State
                pr<=ar;
                nxtstate<=fetch;
                end

        rdm: begin                                        // Read From Memory
               add<=ar;
                rd<=1'b1;
                wrt<=1'b0;
                nxtstate<=read;
                end

        wrm: begin                                          // Write onto Memory
            add<=ar;
                rd<=1'b0;
                wrt<=1'b0;
                nxtstate<=write;
                end

        cpr: begin                                  // Copy 1 register value to other
```

```
            case (data1[3:0])
                  4'b0000:ar<=ar;
                   4'b0100:dr<=ar;
                   4'b1000:gr<=ar;
                   4'b1100:pr<=ar;
                   4'b0001:ar<=dr;
                   4'b0101:dr<=dr;
                   4'b1001:gr<=dr;
                   4'b1101:pr<=dr;
                   4'b0010:ar<=gr;
                   4'b0110:dr<=gr;
                   4'b1010:gr<=gr;
                   4'b1110:pr<=gr;
                   4'b0011:ar<=pr;
                   4'b0111:dr<=pr;
                   4'b1011:gr<=pr;
                   4'b1111:pr<=pr;
                 endcase
                   nxtstate<=fetch;
                   end

adder: begin                          // Addition of two  Registers
                   c_in<=1'b0;
                  en<=1'b1;
                   case (data1[3:0])            // Source Destination Case
                   4'b0000:begin w<=ar;
                                  z<=ar;
                                 nxtstate<=load;
                                end
                   4'b0100:begin w<=dr;
                                  z<=ar;
                                  nxtstate<=load;
                             end
                   4'b1000:begin w<=gr;
                                  z<=ar;
                                nxtstate<=load;
                                end
                   4'b1100:begin w<=ar;
                                   z<=pr;
                                   nxtstate<=load;
```

```
                end
4'b0001:begin w<=dr;
                  z<= ar;
                nxtstate<=load;
                end
4'b0101:begin w<=dr;
                z<=dr;
                  nxtstate<=load;
              end
4'b1001:begin w<=dr;
                  z<=gr;
                  nxtstate<=load;
                end
4'b1101:begin w<=dr;
                  z<=pr;
                 nxtstate<=load;
                 end
4'b0010:begin w<=gr;
                   z<=ar;
                 nxtstate<=load;
                 end
4'b0110:begin w<=gr;
                 z<=dr;
                 nxtstate<=load;
                   end

 4'b1010:begin w<=gr;
                   z<=gr;
                    nxtstate<=load;
                   end
4'b1110:begin w<=gr;
                  z<=pr;
                nxtstate<=load;
                end
4'b0011:begin w<=pr;
                z<=ar;
                 nxtstate<=load;
                end
4'b0111:begin w<=pr;
                   z<=dr;
```

```
                                        nxtstate<=load;
                    end
            4'b1011:begin w<=pr;
                        z<=gr;
                        nxtstate<=load;
                        end
            4'b1111:begin w<=pr;
                        z<=pr;
                        nxtstate<=load;
                    end
            endcase
            end

    sub: begin                                          // subtraction
          c_in<=1'b0;
          en<=1'b0;                         // enable mode to enable MUX
case (data1[3:0])                           // Source Destination Case
            4'b0000:begin w<=ar;
                        z<=ar;
                        nxtstate<=load;
                    end
            4'b0100:begin w<=dr;
                        z<=ar;
                        nxtstate<=load;
                  end
            4'b1000:begin w<=gr;
                        z<=ar;
                    nxtstate<=load;
                    end
            4'b1100:begin w<=ar;
                        z<=pr;
                        nxtstate<=load;
                    end
            4'b0001:begin w<=dr;
                        z<= ar;
                    nxtstate<=load;
                    end
            4'b0101:begin w<=dr;
                    z<=dr;
                        nxtstate<=load;
```

```
                end
4'b1001:begin w<=dr;
                z<=gr;
                nxtstate<=load;
               end
4'b1101:begin w<=dr;
                z<=pr;
               nxtstate<=load;
               end
4'b0010:begin w<=gr;
                z<=ar;
               nxtstate<=load;
               end
4'b0110:begin w<=gr;
               z<=dr;
               nxtstate<=load;
                end

4'b1010:begin w<=gr;
                z<=gr;
                 nxtstate<=load;
               end
4'b1110:begin w<=gr;
               z<=pr;
              nxtstate<=load;
              end
4'b0011:begin w<=pr;
              z<=ar;
               nxtstate<=load;
              end
4'b0111:begin w<=pr;
                z<=dr;
               nxtstate<=load;
              end
4'b1011:begin w<=pr;
                z<=gr;
                nxtstate<=load;
               end
4'b1111:begin w<=pr;
                z<=pr;
```

```
                    nxtstate<=load;
                end
            endcase
            end
lls:        begin       //LLS i.e. copy to least bits of GR
            gr[3:0]<=data1[3:0];
            nxtstate<=fetch;
            end

lms:        begin       // LMS : Copy to Most signficant bits
            gr[7:4]<=data1[3:0];
            nxtstate<=fetch;
            end

 cfr:        begin              // Copy value of Flag register to GR
            gr[3:0]<= fr[3:0];
            nxtstate<=fetch;
            end

 endcase
 end

load: begin                 // Load value of addition and Subtraction
 case (data1[3:2])
      2'b00:ar<=sum;
      2'b01:dr<=sum;
      2'b10:gr<=sum;
      2'b11:pr<=sum;
      endcase
            begin
            if(sum==8'b0) fr[3]<=1;
            else fr[3]<=0;
            if(sum[7]==1) fr[2]<=1;
            else fr[2]<=0;
            if(cout==1) fr[1]<=1;
            else fr[1]<=0;
            if(sum[7]^cout==1) fr[0]<=1;
            else fr[0]<=0;
            end
            nxtstate<=fetch;
```

```
                    end

    read: begin                                     // Read frm memory
                    case(data1[3:2])
                    2'b00:ar<=dat;
                    2'b01:dr<= dat;
                    2'b10:gr<=dat;
                    2'b11:pr<=dat;
                    endcase
                    nxtstate<=fetch;
                    end
       write : begin                                    // Write in Memory
                    wrt<= 1'b1;
                    case(data1[1:0])
                    2'b00:data<=ar;
                    2'b01:data<=dr;
                    2'b10:data<=gr;
                    2'b11:data<=pr;
                    endcase
                    nxtstate<=fetch;
                    end
        default: nxtstate<=fetch;

  endcase
  end


 end
 endmodule

//------------------Alu Module-----------------------//
 module alu(sum,cout,w,z,c_in,en,clk);

//-----------InputPorts-------------------------------//
 input [7:0] w,z;
 input c_in,en,clk;

//------------Output Ports-----------------------//
 output  [7:0] sum;
 output cout;
```

```
//-------------Local Variables---------------------//
 reg [7:0] b;


 always @ (*)
 begin
 if(en==1'b1) begin
 b=z;
 end
 else begin
 b=-z;
 end
 end
 add x1(sum,cout,w,b,c_in);
 endmodule

 module add(sum,cout,w,b,c_in);
 input c_in;
 output  [7:0]sum;
 output cout;
 input [7:0]w,b;
 wire c_in4;
 Add_rca_4 M1 (sum[3:0], c_in4, w[3:0], b[3:0], c_in);
 Add_rca_4 M2 (sum[7:4], c_in8, w[7:4], b[7:4], c_in4);
 endmodule

 module Add_rca_4 (sum, cout, w, b, c_in);
 output  [3: 0] sum;
 output cout;
 input [3: 0] w, b;
 input c_in;
 wire c_in2, c_in3, c_in4;
 Add_full M1 (sum[0], c_in2, w[0], b[0], c_in);
 Add_full M2 (sum[1], c_in3, w[1], b[1], c_in2);
 Add_full M3 (sum[2], c_in4, w[2], b[2], c_in3);
 Add_full M4 (sum[3], cout, w[3], b[3], c_in4);
 endmodule

 module Add_full (sum, cout, w, b, c_in);
```

```
output   sum, cout;
input w, b, c_in;
wire w1, w2, w3;
Add_half M1 (w1, w2, w, b);
Add_half M2 (sum, w3, w1, c_in);
or M3 (cout, w2, w3);
endmodule

module Add_half (sum, cout, w, b);
output  sum, cout;
input w, b;
xor M1 (sum, w, b);
and M2 (cout, w, b);
endmodule
```

Testbench:

```
`timescale 1ns/1ps

module tb_processor();
//----------------inputs-------------//

reg rst,clk;
wire [7:0] dat;

//----------------outputs------------//

wire wrt,rd;
wire [7:0] add,wrt_data;

//-------------memory----------------//
reg [7:0]  mem_data [0:255]; //memory.

//---------------Calling the Main module----------------------//

processor8bit uut(.rst(rst),.clk(clk),.dat(dat),.add(add),.rd(rd),.wrt(wrt));

 //---------------Creating Dump file ----------------------//
initial begin
      $dumpfile("project.vcd");
      $dumpvars(0,tb_processor);
      end
```

```
//----------------assigning memory and data bus----------//

assign  dat=rd? mem_data[add]:8'bz;
assign  wrt_data = wrt ? dat : 8'bz;

//----------------------Initialising Reset---------//
initial begin
       clk=1'b0;
       rst=1;
       #2 rst=0;
       #2000 $finish ;
       end
//--------------------------Filling up the Memory-----//
initial
  begin
  mem_data[0] = 8'b0111_0000 ; // LLS  0 to GR[3:0]
  mem_data[1] =8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 80h
  mem_data[2] = 8'b0100_0010 ; // CPR  GR[7:0] to AR[7:0]   // Now AR had 80h
  mem_data[3] = 8'b0010_0100 ; // RDM  - Reads Address 80h  // DR has data 80h

  mem_data[4] = 8'b0100_1001 ; // CPR  DR[7:0] to GR[7:0]   // Now GR had 80h

  // ------------------------------------------------------------------------------------------------
  mem_data[5] = 8'b0111_0101 ; // LLS  5 to GR[3:0]
  mem_data[6] = 8'b1000_0100 ; // LMS  4 to GR[7:4]        // Now GR has 45h
  mem_data[7] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 45h
  mem_data[8] = 8'b0010_1000 ; // RDM  - Reads Address 45h  // GR has data 45h
  // ------------------------------------------------------------------------------------------------
  mem_data[9] = 8'b0101_0110 ; // ADD contents of GR to DR
  // ------------------------------------------------------------------------------------------------
  mem_data[10] = 8'b0111_0000 ; // LLS  0 to GR[3:0]
  mem_data[11] = 8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 80h
  mem_data[12] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 80h
  mem_data[13] = 8'b0011_0101 ; // WRM - Writes to address 80h // 80h addr has the sum

  // ------------------------------------------------------------------------------------------------
  mem_data[14] = 8'b0111_0001 ; // LLS  1 to GR[3:0]
  mem_data[15] = 8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 81h
  mem_data[16] = 8'b0100_0010 ; // CPR  GR[7:0] to AR[7:0]   // Now AR had 81h
  mem_data[17] = 8'b0010_0100 ; // RDM  - Reads Address 81h  // DR has data 81h
  mem_data[18] = 8'b0100_1001 ; // CPR  DR[7:0] to GR[7:0]   // Now GR had 81h
  // ------------------------------------------------------------------------------------------------
  mem_data[19] = 8'b0111_0001 ; // LLS  1 to GR[3:0]
  mem_data[20] = 8'b1000_1001 ; // LMS  9 to GR[7:4]        // Now GR has 91h
  mem_data[21] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 91h
  mem_data[22] = 8'b0010_1000 ; // RDM  - Reads Address 91h  // GR has data 91h
```

```
// ------------------------------------------------------------------------------------------------
mem_data[23] = 8'b0101_0110 ; // ADD contents of GR to DR
// ------------------------------------------------------------------------------------------------
mem_data[24] = 8'b0111_0001 ; // LLS  1 to GR[3:0]
mem_data[25] = 8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 81h
mem_data[26] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 81h
mem_data[27] = 8'b0011_0101 ; // WRM - Writes to address 81h // 80h addr has the sum
// ------------------------------------------------------------------------------------------------
// Subtraction and Storing the Difference
// ------------------------------------------------------------------------------------------------
mem_data[28] = 8'b0111_0010 ; // LLS  2 to GR[3:0]
mem_data[29] = 8'b1000_1001 ; // LMS  9 to GR[7:4]        // Now GR has 92h
mem_data[30] = 8'b0100_0010 ; // CPR  GR[7:0] to AR[7:0]   // Now AR had 92h
mem_data[31] = 8'b0010_0100 ; // RDM  - Reads Address 92h  // DR has data 92h

// ------------------------------------------------------------------------------------------------
mem_data[32] = 8'b0111_0010 ; // LLS  2 to GR[3:0]
mem_data[33] = 8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 82h
mem_data[34] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 82h
mem_data[35] = 8'b0010_1000 ; // RDM  - Reads Address 82h  // GR has data 82h
// ------------------------------------------------------------------------------------------------
mem_data[36] = 8'b0110_0110 ; // SUB GR fro DR and store in DR
// ------------------------------------------------------------------------------------------------
mem_data[37] = 8'b0111_0010 ; // LLS  2 to GR[3:0]
mem_data[38] = 8'b1000_1000 ; // LMS  8 to GR[7:4]        // Now GR has 82h
mem_data[39] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 82h
mem_data[40] = 8'b0011_0101 ; // WRM - Writes to address 82h // 82h addr has the diff
// ------------------------------------------------------------------------------------------------
// Store Carry
// ------------------------------------------------------------------------------------------------
mem_data[41] = 8'b0111_0000 ; // LLS  0 to GR[3:0]
mem_data[42] = 8'b1000_1111 ; // LMS  F to GR[7:4]        // Now GR has F0h
mem_data[43] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had F0h
mem_data[44] = 8'b1001_1001 ; // COPY contents of FLAG to LSB of GR
// ------------------------------------------------------------------------------------------------
mem_data[45] = 8'b0011_1010 ; // WRM - Writes to address F0h value of GR
// ------------------------------------------------------------------------------------------------
mem_data[46] = 8'b0000_0000 ; // NOP
mem_data[47] = 8'b0000_0000 ; // NOP
// ------------------------------------------------------------------------------------------------
// JUMP to address location 00(dec) = 00(hex)
mem_data[48] = 8'b0111_0000 ; // LLS 4'h0 to GR[3:0]
mem_data[49] = 8'b1000_0000 ; // LMS 4'h0 to GR[7:4]       // Now GR has 00h
mem_data[50] = 8'b0100_0010 ; // COPY GR[7:0] to AR[7:0]   // Now AR had 00h
mem_data[51] = 8'b0001_0000 ; // JMP
mem_data[52] = 8'b0000_0000 ; // NOP
```

```
  mem_data[53] = 8'b0000_0000 ; // NOP
  mem_data[54] = 8'b0000_0000 ; // NOP
  mem_data[55] = 8'b0000_0000 ; // NOP
  mem_data[56] = 8'b0000_0000 ; // NOP

 //-----------------------------------------------------------------------------------------------
  // Jump to Zero-th Location
  mem_data[124] = 8'b0111_0000 ; // LLS
  mem_data[125] = 8'b1000_0000 ; // LMS
  mem_data[126] = 8'b0100_0010 ; // COPY contents of GR to AR
  mem_data[127] = 8'b0001_0000 ; // JMP to Address = 00h

  mem_data[69]=8'b0100_0101;
  mem_data[128]=8'b0100_0000;
  mem_data[129]=8'b0101_0110;
  mem_data[145]=8'b0110_0110;
  mem_data[146]=8'b0111_0101;
  mem_data[130]=8'b0011_0011;
  // -----------------------------------------------------------------------------------------------

  end
//----------------Initialising Clock----------------------//
initial begin
        forever clk=#5.1 ~clk;
        end
//---------------monitoring write  data-------------------//
always @(posedge clk)
        begin
        if(wrt==1)
        mem_data[add] <= wrt_data;
        end


endmodule
```

Appendix C
Reports and Circuits from EDA Tools

C.1 Reports (contents) from RTL (Pre-synthesis) Simulations (VCS or NCVERILOG)

report.docx

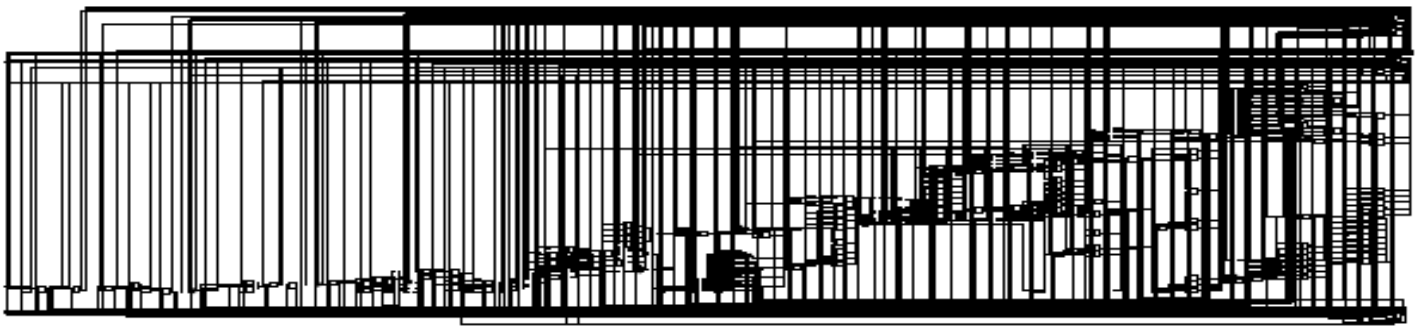
C.2 Reports (contents) from Netlist (Post-synthesis) Simulations (VCS or NCVERILOG)

project_netlist.docx

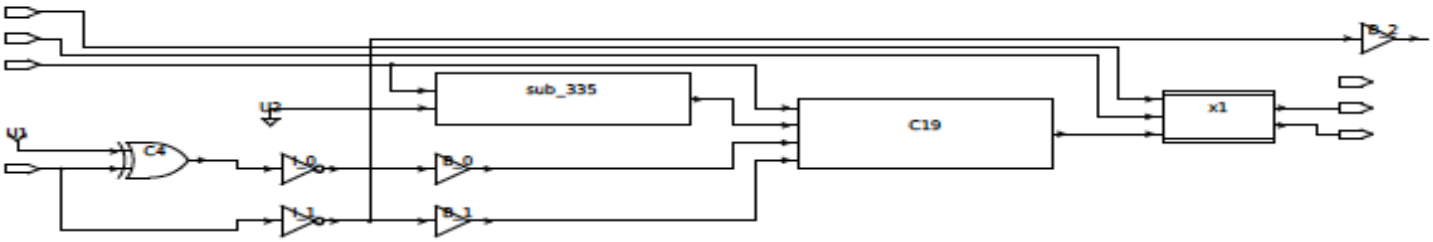
C.3 Reports (contents) from Synthesis (Design Compiler)

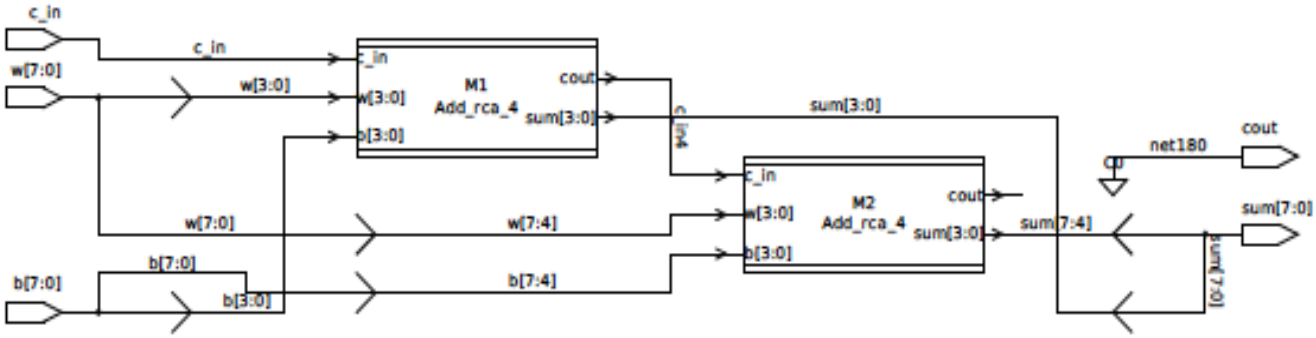
pre_synthesis.docx

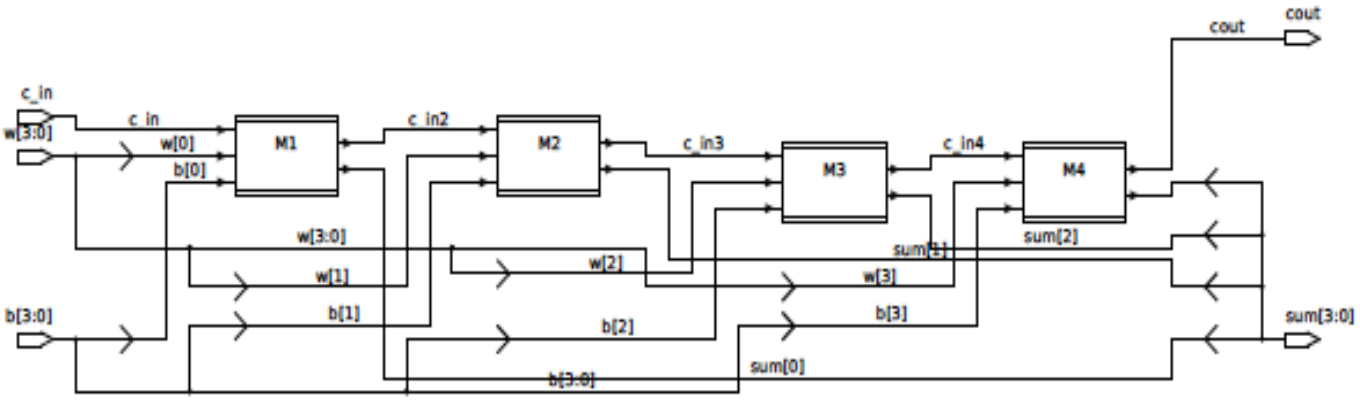
C.4 Screenshot Circuits from Synthesis (Design Compiler)

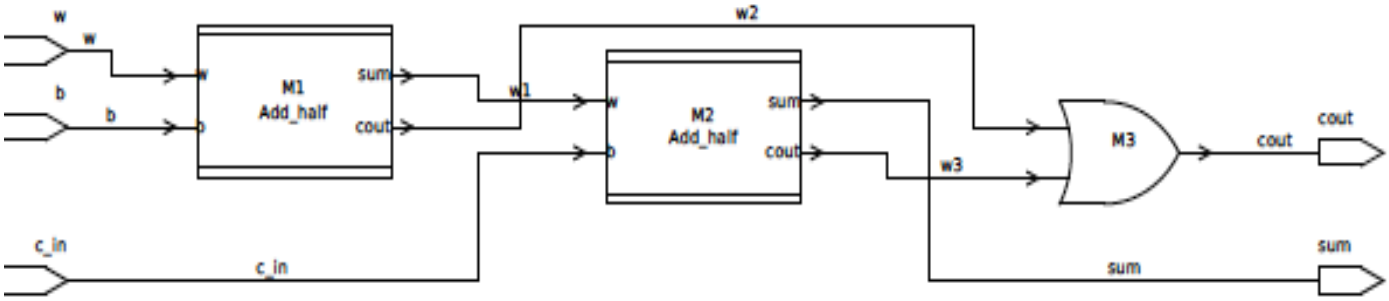


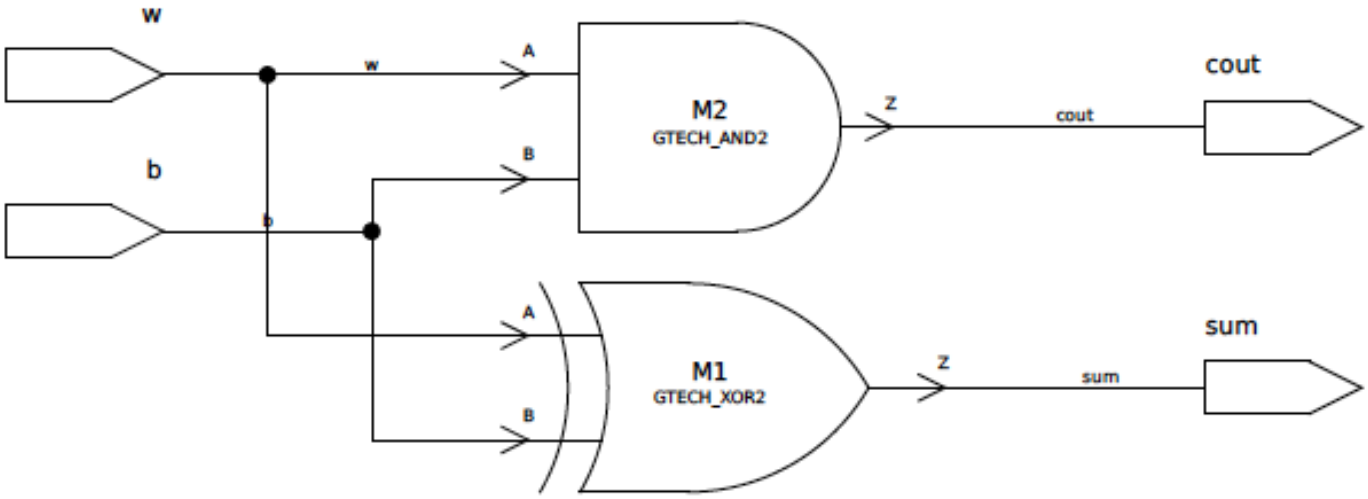
1. Processor Design


sub_335
C19
x1
C4
U2
U1
I_0
I_1
B_0
B_1
B_2


c_in
w[7:0]
b[7:0]
w[3:0]
b[3:0]
M1
Add_rca_4
cout
sum[3:0]
int4
M2
w[7:4]
b[7:4]
sum[7:4]
net180
sum[7:0]


c_in
w[3:0]
c_in
w[0]
b[0]
M1
c_in2
M2
c_in3
M3
c_in4
M4
cout
cout
w[3:0]
w[1]
b[1]
w[2]
b[2]
w[3]
b[3]
sum[1]
sum[2]
b[3:0]
sum[0]
b[3:0]
sum[3:0]


w
w
b
b
M1
Add_half
sum
cout
w1
w2
M2
Add_half
sum
cout
w3
M3
cout
cout
c_in
c_in
sum
sum


w
b
w
b
A
B
M2
GTECH_AND2
Z
cout
cout
A
B
M1
GTECH_XOR2
Z
sum
sum