Andy Jeong, Brenda So, Gordon Macshane

ECE 151A Spring 2016

22 February 2016

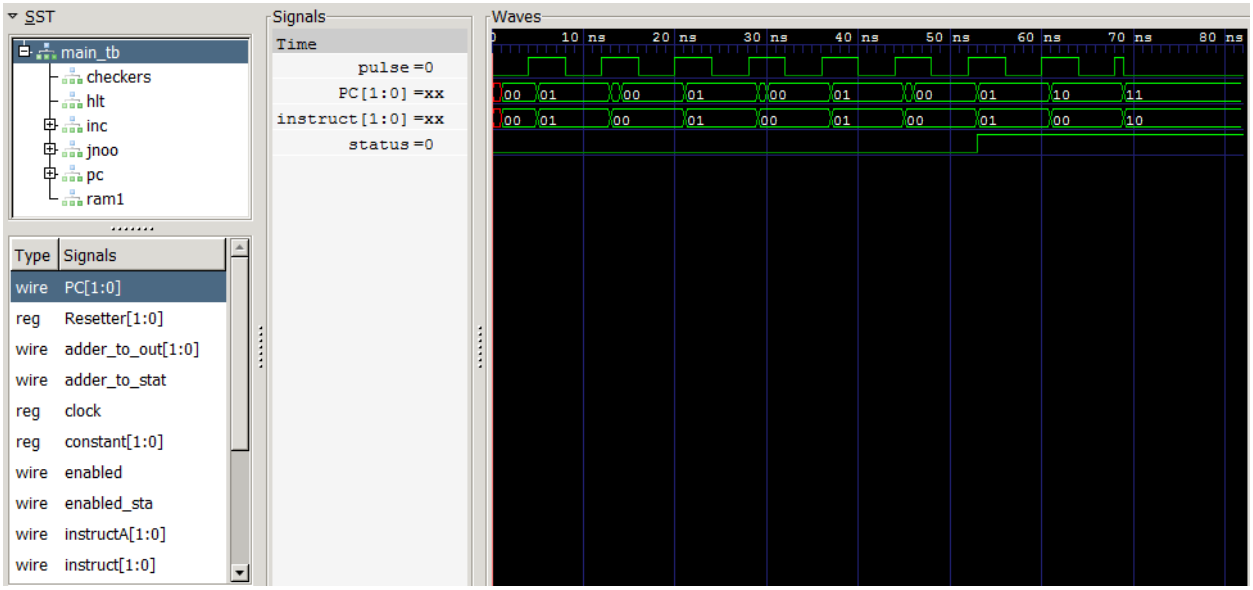
Project 1: Paper Processor

Abstract

This project aims to construct a circuit on Verilog Hardware Description Language whereby the paper processor (https://sites.google.com/site/kotukotuzimiti/Paper_Processor) is implemented. The Verilog code takes in the modules (.v) – *checking, dff, dfff, halt, increment, jno, programcounter, two_bit_fa, twoBitRam* – which take the inputs and outputs the data bits according to the logic gates event-driven cases (jno) and delays. The *instruct.bin* file takes in the instruction codes, which are picked out based on the logic, and determines the data output.

Diagram (refer to *waveform.png*)**)**

Waveform 1: Paper Processor implementation for time duration = 69 (ns)



Sample Output *(refer to sample_output.txt)*

VCD info: dumpfile pp.vcd opened for output.

time = 0 PC = xx instruct = xx out = 00 stat = 0

time = 1 PC = 00 instruct = 00 out = 00 stat = 0

time = 5 PC = 01 instruct = 01 out = 01 stat = 0

time = 13 PC = 10 instruct = 00 out = 01 stat = 0

time = 14 PC = 00 instruct = 00 out = 01 stat = 0

time = 21 PC = 01 instruct = 01 out = 01 stat = 0

time = 22 PC = 01 instruct = 01 out = 10 stat = 0

time = 29 PC = 10 instruct = 00 out = 10 stat = 0

time = 30 PC = 00 instruct = 00 out = 10 stat = 0

time = 37 PC = 01 instruct = 01 out = 10 stat = 0

time = 38 PC = 01 instruct = 01 out = 11 stat = 0

time = 45 PC = 10 instruct = 00 out = 11 stat = 0

time = 46 PC = 00 instruct = 00 out = 11 stat = 0

time = 53 PC = 01 instruct = 01 out = 11 stat = 0

time = 54 PC = 01 instruct = 01 out = 00 stat = 1

time = 61 PC = 10 instruct = 00 out = 00 stat = 1

time = 69 PC = 11 instruct = 10 out = 00 stat = 1

Compiling and Running the Code (refer to *Build.txt*)

To change the directory to the file location:

cd /<location1>/<location2> (...so on)

To compile on terminal:

iverilog -o <objectname> main_tb.v

To run the compiled object file:

vvp <objectname>

(if <objectname> is testbench, type ./testbench)

To change the instruction:

change the bits in instruct.bin

Optional:

To see the waveform using gtkwave simulator (as defined in code as "pp.vcd"):

gtkwave pp.vcd

To remove the object file:

rm <objectname>

Paper Processor Design (refer to *program.asm)*

The Paper Processor Instructions

inc	//increments the address, register
jno $00_{(2)}$	//jump if not overflow
hlt	//halt
nop	//no operation

Verilog Code

Main Testbench Code (refer to main_tb.v)

```
//
// School: The Cooper Union
// Course: ECE151A Spring 2016
// Assignment: Paper Processor
// Group members: Andy Jeong, Brenda So, Gordon Macshane
// Date: 2/22/2016
//
//------------------Instructions----------------------
//       Instructions Guide:
//                INC: 00(2)
//                JNO: 01(2)
//                HLT: 10(2)
//
//       Instructions for the 2-bit paper processor:
//                IS1:     INC
//                IS2,3:  JNO 00(2)
//                IS4:     HLT
//
`timescale 1ns/1ns

`include "twoBitRam.v"
`include "increment.v"
`include "programcounter.v"
`include "halt.v"
`include "jno.v"
`include "checking.v"
`include "dff.v"
`include "dfff.v"

module test_tb();

reg status;                                                    // status register
reg clock;                                                             // clock register
wire monostable;                                       // monostable clock induced by inc
logic
wire pulse;                                                            // clock induced
by halt
wire monostabler;
reg [1:0] out;                                                 // output registers
wire w3, w4, w5, w6;
wire openpulser;                                       //Pulse from JNO, imitating
monostable timer

reg [1:0] constant;                                    // INC logic: add to full adder
wire [1:0] adder_to_out;                    // INC logic :connect adder to reg out
wire adder_to_stat;                                    // ING logic: ddding to
status
```

```
wire [1:0] instruct;                                    // output instruction from RAM
(LSB)
wire [1:0] instructA;                                   // actual instruction after
checking
reg [1:0] Resetter;                                             //used to initialize the
program counter

wire enabled, enabled_sta;
wire [1:0] PC;                                                  // Counter: program counter
output

initial begin

        //print out things to monitor
        $monitor ("time = %g\t PC = %b instruct = %b out = %b stat = %b", $time, PC,
instruct, out, status);

        clock = 0;                                      // initialize clock to be 0
        status = 0;                         // initialize status to be 0

        out[0] = 0;                         // initialize LSB to be 0
        out[1] = 0;                         // initialize MSB to be 0
        constant[0] = 1;                    // constant added to adder
        constant[1] = 0;                    // constant added to adder
        Resetter[1] = 0;
        Resetter[0] = 0;

        #1 Resetter[1] = 1;
        Resetter[0] = 1;
        #1 Resetter[1] = 0;
        Resetter[0] = 0;
        //let the simulation run for 15 seconds
        #80 $finish;
end

initial begin
        $dumpfile ("pp.vcd");    //for waveform file in .vcd format
    $dumpvars;
end

//only when the instructions are 00
always @(posedge monostabler)
begin
        out[0] <= adder_to_out[0];
        out[1] <= adder_to_out[1];
end

//set the status such that once the status turns 1 it would never change back
//to zero
always @(posedge monostabler)
```

```
begin
        status <= adder_to_stat;
end

//Set the clock to alter every second
always begin
        #4 clock = ~clock; // Generate clock
end


//Connect twoBitRam to test bench

twoBitRam ram1(
PC,
instruct
);

//connect increment blackbox to testbench
increment inc(
adder_to_out,
adder_to_stat,
pulse,
monostable,
out,
instructA);

//connect counter to testbench
programcounter pc(
PC,
openpulser,
Resetter,
instruct,
pulse,
enabled_sta);

and (monostabler, monostable, !status);

//connect halt blackbox to program counter
halt hlt(
pulse,
clock,
instructA);

//connect jno blackbox to checking logic
jno jnoo(enabled, enabled_sta, openpulser, status, instructA, pulse);
checking checkers(instructA, enabled, instruct);

endmodule
```

checking.v (refer to checking.v)

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: checking.v
// Description: checks if either of two inputs is HIGH at a certain time
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module checking(checked, enabling, check);

//-------------Input Ports------------------------------

input [1:0] check;        //input a 2 bit register from JNO
input enabling;

//-------------Output Ports-----------------------------

output [1:0] checked;

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] check;
wire enabling;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire [1:0] checked;

//-------------Intermediate Wires-----------------------


//------------------Instructions-----------------------

or o1(checked[1], enabling, check[1]);
or o2(checked[0], enabling, check[0]);


endmodule
```

dff.v (refer to dff.v)

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: dff.v
// Description: D flip flop that sets and resets at negedge
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module dff (clk, s, r, d, q, qbar);
         input clk, s, r, d;
         output q,qbar;
         reg q,qbar;
         always @(negedge r) begin
                  q = 1'b0;
                  qbar = 1'b1;
                  end
         always @(negedge s) begin
                  q = 1'b1;
                  qbar = 1'b0;
                  end
         always @(posedge clk) begin
                  #1
                  q = d;
                  qbar = ~d;
                  end

endmodule
```

dfff.v (refer to dfff.v)

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: dfff.v
// Description: D flip flop that sets and resets at posedge
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module dfff (clk, s, r, d, q, qbar);
         input clk, s, r, d;
         output q,qbar;
         reg q,qbar;
         always @(posedge r) begin
                  q = 1'b0;
                  qbar = 1'b1;
                  end
         always @(posedge s) begin
                  q = 1'b1;
                  qbar = 1'b0;
                  end
         always @(posedge clk) begin
                  #1
                  q = d;
                  qbar = ~d;
                  end

endmodule
```

halt.v *(refer to halt.v)*

```
////////////////////////////////////////////////////////////////////////////////
// Module Name: halt.v
// Description: takes clock input and according to the current instruction it
// determines if the clock should go on or not
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module halt(pulses, clk, instruct);

//-------------Input Ports------------------------------

input [1:0] instruct;    //input a 2 bit register from JNO
input clk;

//-------------Output Ports-----------------------------

output pulses;

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] instruct;
wire clk;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire pulses;

//-------------Intermediate Wires-----------------------
wire w1;
wire w2;

//------------------Instructions-----------------------

and a1(w1, instruct[1], !instruct[0]);
not n1(w2, w1);

and(pulses, clk, w2);

endmodule
```

increment.v *(refer to increment.v)*

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: increment.v
// Description: increments 2-bit value from previous
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

`include "two_bit_fa.v"

module increment(value, sta, pulser, mn, prev_value, instruct);

//-------------Input Ports------------------------------

input [1:0] instruct;     //input a 2 bit register from JNO
input [1:0] prev_value;
input pulser;

//-------------Output Ports-----------------------------

inout [1:0] value;
output sta;
output mn;

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] instruct;
wire [1:0] clk;
wire [1:0] prev_value;
wire pulse;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire [1:0] value;
wire sta;
reg mn;
wire w1;

//-------------Intermediate Wires-----------------------


//------------------Instructions-----------------------

and (w1, pulser, !instruct[1], !instruct[0]);
initial begin
mn = 0;
end

always @(posedge w1)
```

```
begin
        #1
        mn = 1;
        #2
        mn = 0;
end

fulladder adder(sta, value, prev_value);

endmodule
```

jno.v (refer to jno.v)

```
////////////////////////////////////////////////////////////////////////////////
// Module Name: jno.v
// Description: from the current instructions, outputs ENABLE for checking and counter
////////////////////////////////////////////////////////////////////////////////
`timescale 1ns/1ns

module jno(enabling, enabling_sta, openpulse, sta, instruct, pulses);

//-------------Input Ports------------------------------

input [1:0] instruct;    //input a 2 bit register from JNO
input pulses;
input sta;

//-------------Output Ports-----------------------------

output enabling;
output enabling_sta;
output openpulse;

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] instruct;
wire pulses;
wire sta;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire enabling;
wire enabling_sta;

//-------------Intermediate Wires-----------------------
reg s;
reg r;
wire notenabling, notenabling_sta;
```

```
wire w1, w2;
reg mem;
reg pulser;
wire openpulse;
reg openpulser;

//------------------Instructions----------------------
initial begin
s = 0;
r = 0;
pulser = 0;
openpulser = 0;
end

and a1(w1, !instruct[1], instruct[0]);
and a2(w2, !sta, !instruct[1], instruct[0]);
and a3 (openpulse, !sta, openpulser);

always @(posedge w1)
begin
        #1
        pulser = 1;
        #8
        openpulser = 1;
        #4
        openpulser = 0;
        #1
        pulser = 0;
        #1
        pulser = 1;
        #1
        pulser = 0;
end
dff dff3(pulser, s, r, w1, enabling, notenabling); //enabling the checking
dff dff4(pulser, s, r, w2, enabling_sta, notenabling_sta); //enabling for counter

endmodule
```

programcounter.v *(refer to programcounter.v)*

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: programcounter.v
// Description: counter
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module programcounter(select, openpulse, R, jno, clk, enabled);

//-------------Input Ports------------------------------

input [1:0] jno;          //input a 2 bit register from JNO
input clk;                        //input clock signal
input enabled;            //input enabling JNO register addresses
input [1:0] R;            //for resetting
input openpulse;

//-------------Output Ports-----------------------------

output [1:0] select;

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] jno;
wire clk;
wire enabled;
wire [1:0] R;
wire openpulse;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire [1:0] select;

//-------------Intermediate Wires-----------------------

wire [1:0] selectN;
wire w1, w2,w3,w4, w5, w6;
wire [1:0] D;
wire [1:0] S;
wire [1:0] R_temp;

//------------------Instructions-----------------------
and a1(S[0], openpulse, enabled, jno[0]);
and a2(S[1], openpulse, enabled, jno[1]);
and a3(w3, openpulse, enabled, !jno[0]);
and a4(w4, openpulse, enabled, !jno[1]);
or o1(R_temp[0], w3, R[0]); //R[0] is 0, w1 goes from 0 to 1
or o2(R_temp[1], w4, R[1]); //
```

```
dfff dff0(clk, S[0], R_temp[0], D[0], select[0], selectN[0]);
dfff dff1(clk, S[1], R_temp[1], D[1], select[1], selectN[1]);

and (w1, selectN[0], select[1]);
and (w2, selectN[1], select[0]);
or (D[1], w1, w2);
and (D[0], selectN[0],1);

endmodule
```

two_bit_fa.v *(refer to two_bit_fa.v)*

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: fulladder.v
// Description: Full Adder where 1 is always added to the previous number
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module fulladder(stat, sum, a);

//-------------Input Ports------------------------------

inout [1:0] a; //input a 2 bit register

//-------------Output Ports-----------------------------

output [1:0] sum; //output 2 bit registers
output stat; //see whether registers overflown

//-------------Input ports Data Type--------------------
// By rule all the input ports should be wires
wire [1:0] a;
wire [1:0] b;

//-------------Output Ports Data Type-------------------
// Output port can be a storage element (reg) or a wire
wire [1:0] sum;
wire stat;

//-------------Intermediate Wires-----------------------
wire w0, w1, w2, w3, w_sum0, w_sum1, w_stat;

//------------------Instructions-----------------------
//set LSB of input to w_sum0
xor u0(w_sum0,a[0], 1'b1);
```

```
and (sum[0], w_sum0, 1);

//set MSB of input to w_sum1
and u1(w0, a[0], 1'b1);
xor u2(w1, a[1], 1'b0);
and u3(w2, a[1], 1'b0);
and u4(w3, w0, w1);
xor u5(w_sum1, w0, w1);
and(sum[1], w_sum1,1);

//set carry out to be w_stat
or  u6(w_stat,w2, w3);
and (stat, w_stat,1);

endmodule
```

twoBitRam.v *(refer to twoBirRam.v)*

```
////////////////////////////////////////////////////////////////////////////////
// Module Name: twoBitRam.v
// Description: determines data outputs from address
////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ns

module twoBitRam(addr, data);

parameter Instructions = "./instruct.bin";

//-------------Input Ports-----------------------------

input [1:0] addr;

//-------------Output Ports----------------------------

output [1:0] data;

//-------------Input ports Data Type-------------------
// By rule all the input ports should be wires
wire [1:0] addr;

//-------------Output Ports Data Type------------------
// Output port can be a storage element (reg) or a wire
wire [1:0] data;


//------------------Instructions----------------------
/*
```

```
Instructions:
INC: 00
JNO: 01
HLT: 10

in1: INC
in2: JNO
in3: 00
in4: HLT
*/

//-------------Save Instructions In Ram----------------
reg [1:0] ram_reg [0:3];                // make register 2 bits wide  and 4 bits long
        initial begin
                $readmemb(Instructions, ram_reg);       // load program
        end
assign data = ram_reg[addr];

/*
//First Demux
//and and1_msb(a1_msb, !addr[1], !addr[0], 1'b0);
//and and2_msb(a2_msb, !addr[1], addr[0], 1'b0);
//and and3_msb(a3_msb, addr[1], !addr[0], 1'b0);
//and and4_msb(a4_msb, addr[1], addr[0], 1'b1);

//Second Demux
and and1_lsb(a1_lsb, !addr[1], !addr[0], 1'b0);
and and2_lsb(a2_lsb, !addr[1], addr[0], 1'b1);
and and3_lsb(a3_lsb, addr[1], !addr[0], 1'b0);
and and4_lsb(a4_lsb, addr[1], addr[0], 1'b0);

//Final Or Gate
or or_msb(data[1],a1_msb,a2_msb,a3_msb,a4_msb);
or or_lsb(data[0],a1_lsb,a2_lsb,a3_lsb,a4_lsb);
*/

endmodule
```