ENGN3213

Digital Systems & Microprocessors

CLAB 5: Datapaths

V3.0
Copyright 2010 G.G. Borg ANU Engineering
1 CLAB 5: Datapaths

1.1 Aims

In class we had a look at our final abstraction for the course: the datapath. We concentrated on a design for the sequential datapath. In this CLAB we are going to do some examples of sequential datapath design.

The aims for CLAB5 are,

- Learn about sequential datapaths
- View SDs as electronic circuits that implement algorithms
- Implement and simulate some examples in VERILOG

1.2 Background

Read the chapter on datapaths in the course reading brick to familiarise yourself with the material of the lecture on datapaths.

In this next sections we discuss a datapath algorithm implementation in detail.

1.3 A Timer or Signal Generator

Design a sequential datapath to execute an algorithm to timeout a binary counter at a preset count. A signal generator or frequency synthesiser is a slightly more complex device that outputs any number of signals at (usually but not always fixed) predetermined intervals. Examples are direct digital synthesisers (DDS) and Coordinate Rotating Digital Computers (CORDICS).

The variables (some of which have to be stored in registers for example) are

- **TIC**: The input bit from the calling device that starts the timer
- **TARGET**: the input word from the calling device that is the count value for the timeout.
- **CNTR**: An internal variable that indicates the current counter value
- **TOC**: The output bit that indicates the calling device that the count has completed.

It should be clear that TIC and TOC do not need to be stored because they are only ever valid after the active edge of the clock. TARGET and the current value of the CNTR need to be remembered during the execution of the algorithm. For this we use a register, vreg8 in a similar fashion to the vreg16 of MU0.
if(TIC)
[1] Store TARGET
[2] initialise CNTR = 0
    while( |(Target ^ CNTR) )
[3] CNTR = CNTR + 1
    end while
[4] TOC=1
end if

This is a normal FSM design with FSM VERILOG coding style. There are three modules in the timer, timer.v which contains the FSM, vreg8.c the register and a testbench. The modules are shown below. We store the current value of the CNTR in the variable CNT - The construct is similar to the way the program counter worked in MU0. Since TARGET is only available in the initial state we need to catch it and store it in the variable Tlocal. The algorithm begins by initialising the registers containing CNTR and Tlocal. The states are given the variable name line as they correspond to the line numbers of the algorithm.

```
//module timer(clk, TIC, TARGET, TOC);
input clk;
input TIC;
input [7:0] TARGET;
output TOC;
parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
reg [1:0] line;
reg [1:0] lnext;
reg TOC;
reg [7:0] CNTR;
wire [7:0] CNT;
wire [7:0] Tlocal;
wire cond;
reg ent,rst,enc, rsc;
vreg8 Target( clk, Tlocal, TARGET, ent, rst);
vreg8 Count( clk, CNT, CNTR, enc, rsc);
assign cond = |(Tlocal^CNTR);
always@ (posedge clk)
    line <= lnext;
always @(line or TIC or cond) begin
    case(line)
        S0: begin
            if(TIC) begin
                lnext <= S1;
            end else begin
                lnext <= S0;
            end
        end
        S1: begin
            if( cond ) begin
                lnext <= S1;
            end else begin
                lnext <= S2;
            end
        end
        S2: lnext <= S0;
        default lnext <= S0;
    endcase
end
```

4
always @(*) begin
  case(line)
  S0:
    begin
      if(TIC) begin
        ent = 1'b1;
        rst = 1'b0; //Set Tlocal = TARGET
        TOC = 1'b0;
        CNTR = 1'b0;
      end else begin
        ent = 1'b1;
        rst = 1'b1; //Reset Tlocal = 0
        enc = 1'b1;
        rsc = 1'b1; //set the CNTR to 0
        TOC = 1'b0;
      end
    end
  S1:
    begin
      if( |(Tlocal^CNTR) ) begin
        CNTR = CNT + 8'h01; //increment the counter
        ent = 1'b0;
        rst = 1'b0; //Store CNTR
        enc = 1'b1;
        rsc = 1'b0; //Store CNTR
        TOC = 1'b0;
      end else begin
        CNTR = 8'h00;
        enc = 1'b1;
        rsc = 1'b0; //Store CNTR
        TOC = 1'b0;
      end
    end
  S2:
    begin
      TOC = 1'b1;
    end
  default: TOC = 1'b0;
  endcase // case (line)
end // always @ (*)
endmodule // timer

module vreg8( clk, q, d, en, rs );
  output [7:0] q;
  input [7:0] d;
  input clk;
  input en;
  input rs;
  reg [7:0] state;
  assign q = state;
  always @(posedge clk) begin
    if(en && ~rs) state <= d;
    else if(en && rs) state <= 16'h0;
    else state <= state;
  end
endmodule // v_reg8

module TB_timer;
  reg clk;
  reg TIC;
  reg [7:0] TARGET;
  wire TOC;
  integer i;
  timer tmr(clk, TIC, TARGET, TOC);
initial begin
  $clk = 0;
  $clk = 1;
  $TIC = 1'b0;
  $clk = 0;
  $clk = 1;
  $TIC = 1'b1;
  $TARGET = 8'h00010000;
end
```verilog
#1clk = 0;
#1clk = 1;
#1TIC = 1'b0;
#1clk = 0;
#1clk = 1;
for(i=0;i<30;i=i+1) begin
  #1clk = 0;
  #1clk = 1;
end
initial begin
  $dumpfile("timer.vcd");
  $dumpvars;
end
endmodule

Exercise 1: Implement the above verilog code in ICARUS VERILOG (or GTKwave) or XST simulator and reproduce the traces of Fig. 2.

1.4 A One’s Counter

Exercise 2: Along the same lines as the previous exercise, design a one’s counter that will count the number of 1’s in an input 16-bit word, DataWord, and return the result. We will add a strobe Strobe to indicate when DataWord is valid. Ocount is the output number of bits in the word. Ocount is reset when Strobe is first set to 1.

We can write the algorithm as follows.

```
while(Strobe)
[1] Ocount := 0
[2] Store DataWord
[3] Mask := 17’h100
[4] Done := 0;
    while( ~Mask[0] )
end while
[7] Done := 1;
```

Note that the `>>` construct has been avoided for hardware description and has been replaced by a simple rearrangement of the bits of Mask. This is a much better approach in VERILOG.

- What variables in the above algorithm will need to be stored in registers?
- Draw the state diagram of the one’s counter algorithm using a sequential datapath
- Write the Verilog code to implement the state diagram
Figure 1: FSM of the timer.
Figure 2: GTKWAVE traces of the timer with a target of 0x10.