COMPLEX ENGINEERING PROBLEM

Submitted To: **Attique ur Rehman**

Group Members: Muhammad Haseeb

Usama Bin Aqil

Rao Umer Rashid

Roll ID: 200718, 200758, 200776

Department: Electrical Engineering

Class: Electronics (7-A)

Subject: FPGA Lab

Abstract:

This report details the FPGA implementation of the Capture module from a PIC using Verilog. The CCP module captures external events for real-time applications. The architecture, Verilog code, and performance analysis are presented, showcasing precise event capture.

Objective:

- Understand the working of CCP (Capture) module and its implementation of Verilog
- Simulations of Xilinx ISE Design Suite
- Hardware Implementation on FPGA (Spartan 3E)

Introduction:

Depending on the family member, the PIC18 has anywhere between 0 and 5 CCP modules inside it. The multiple CCP modules are designated as CCP1, CCP2, CCP3, and so on (CCPx). In recent years, the PWM feature of the CCP has been enhanced greatly for better DC motor control, producing what is called enhanced CCP (ECCP). Therefore, a given family member can have two standard CCP modules and one or more ECCP modules, all on a single chip.

The CCP Registers:

Each CCP module has three registers associated with it.They are as follows:

a) CCPxCON is an 8-bit control register.We select one of the compare,capture, and PWM modes using this register.
b) CCPRxL and CCPRxH form the low byte and the high byter of the 16-bit register/This 16-bit register can be used either as a 16-bit compare register, or a 16-bit capture register, or an 8-bit duty cycle register by the PWM, but not all at the same time.

CCP1CON Register:

The operation of both CCP1 & CCP2 modules is controlled via the **CCPxCON** Registers which are a couple of 8-Bit SFRs (CCP1CON & CCP2CON) respectively.

he operation of the CCP module in capture mode goes as follows. First, the timer module is set to operate in timer or counter mode and its register's value (**TMR1**) starts incrementing. Second, the CCP module should be configured to operate in capture mode with a selectable trigger event. There are obviously 4-options that fire capture signal (e.g. every rising edge), let's say every rising edge is selected via the CCP1CON register.

Now, upon each rising edge on the CCP1 pin (**RC2**). The 16-Bit value of the **TMR1** will be copied to the 16-Bit **CCPR1** register. This action fires an interrupt signal that is dedicated to the CCP1 module in general. We can actually use this to instantaneously handle CCP-Capture events.

INSIDE CCP1CON/CCP2CON Register:

CCP1CON REGISTER/CCP2CON REGISTER

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 3-0 : **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = CCPx module is off
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 = Compare mode,
Generate software interrupt on compare match (CCPIF bit is set, CCP pin is unaffected)
1011 = Compare mode,
Trigger special event (CCPIF bit is set)
11xx = PWM mode

bit 7-6 : **Unimplemented:** Read as '0‘

bit 5-4 : **DCxB1:DCxB0**: PWM Duty Cycle bit1 and bit0
Capture mode: Unused
Compare mode: Unused
PWM mode: These bits are the two LSbs (bit1 and bit0) of the 10-bitWM duty cycle. The
. upper eight bits(DCx9:DCx2) of the duty cycle are found in CCPRxL.

Capture Module:

In Capture mode, **CCPRxH:CCPRxL** captures the 16-bit value of the TIMER-1/3 register when an event occurs on pin **CCPx**. An event is defined as one of the following:

a) Every **Falling edge** (1 » 0) on the CCPx pin.

b) Every **Rising edge** (0 » 1) on the CCPx pin.
c) Every **4th rising edge** (0 » 1) on the CCPx pin.
d) Every **16th rising edge** (0 » 1) on the CCPx pin.

In Capture Mode these settings are necessary.

- **CCPx** pin must be configured as input.
- **TIMER-1 or 3** module must operate as timer or synchronous counter.

Verilog Implementation:

For our implementation we will be using the internal crystal which generated a pulse of 50Mhz frequency. We will use clock divider in order to observe once we go towards hardware implementation. So instead of CCPx pin we will generate the pulse internally. Also the CCP1CON register is only 5 bits in because Capture mode only utilizes 4 bits for combination .The extra 1 bit is for error indication if a unknown combination is chosen.

Complete Code with Explanation(Software):

```
module capture(
    input clk,
    output reg [15:0] timer_reg=0, //16-bit register
    output reg [15:0] CCP1RL_CCP1RH=0,  //16-bit register
    output reg [3:0] edge_counter=0,     //Register to keep count of edges of pulse
    output reg led=0,       // Indication for when edge of selected mode reached
    output reg error=0,   // Error when incorrect combination selected
    input [4:0] CCP1CON   //Mode selection
    );

   always@(negedge clk)  //To Capture falling edge
   begin
       if (CCP1CON[4] == 1'b0)begin
         if (CCP1CON ==5'b00100)begin
            timer_reg = timer_reg+ 1;   // Increment the timer on each falling edge
            CCP1RL_CCP1RH=timer_reg ;   // Throw timer values
            led = ~led;   //Toggle Led

         end
      end
      else error = ~error ; // Error Indication

   end
```

```
       always@(posedge clk) begin // To capture rising edge
      if (CCP1CON[4] == 1'b0)begin
       case (CCP1CON) //3 cases for rising edges

      5'b00101 : begin
            timer_reg = timer_reg+ 1; // Increment the timer on each rising edge
            CCP1RL_CCP1RH=timer_reg ;
            led = ~led;
       end

      5'b00110 :begin
            timer_reg = timer_reg+ 1; // Increment the timer on each rising edge
            if(edge_counter == 3)begin  //4th rising edge
            CCP1RL_CCP1RH=timer_reg ;
             led = ~led;
            edge_counter = 0;
```

```
    end
    else
            edge_counter = edge_counter+1;
    end

     5'b00111 :begin
            timer_reg = timer_reg+ 1; // Increment the timer on each rising edge
            if(edge_counter == 15)begin  //16th Rising edge
            CCP1RL_CCP1RH=timer_reg ;
            led = ~led;
            edge_counter = 0;
    end
    else
            edge_counter = edge_counter+1;
    end
    endcase
    end
    else error = ~error ;
    end
    endmodule
```

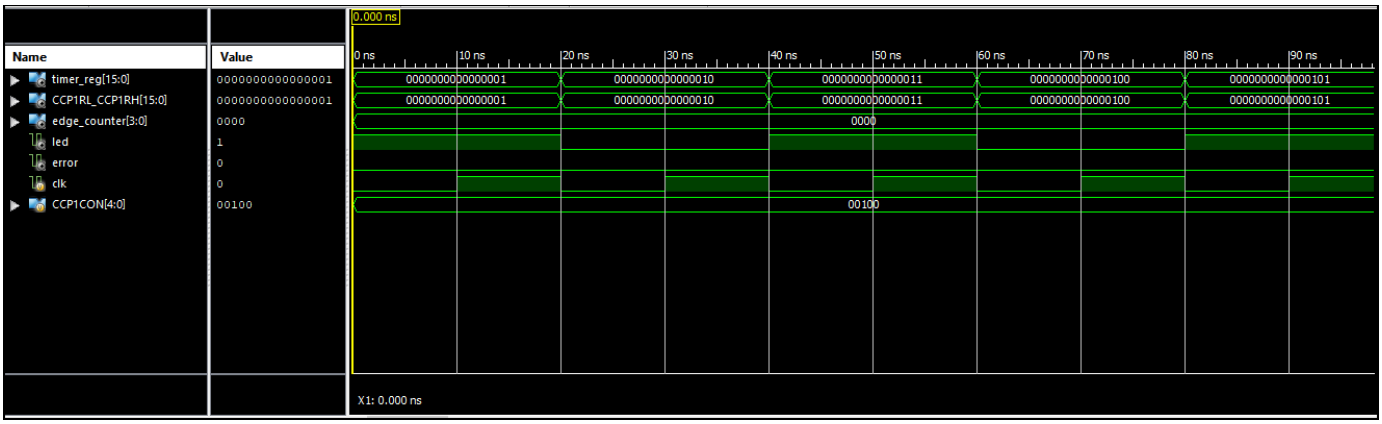
Generic TestBench

```
module test;

   // Inputs
   reg clk;
   reg [4:0] CCP1CON;

   // Outputs
   wire [15:0] timer_reg;
   wire [15:0] CCP1RL_CCP1RH;
   wire [3:0] edge_counter;
   wire led;
   wire error;

   // Instantiate the Unit Under Test (UUT)
   capture uut (
      .clk(clk),
      .timer_reg(timer_reg),
      .CCP1RL_CCP1RH(CCP1RL_CCP1RH),
      .edge_counter(edge_counter),
      .led(led),
      .error(error),
      .CCP1CON(CCP1CON)
   );
```

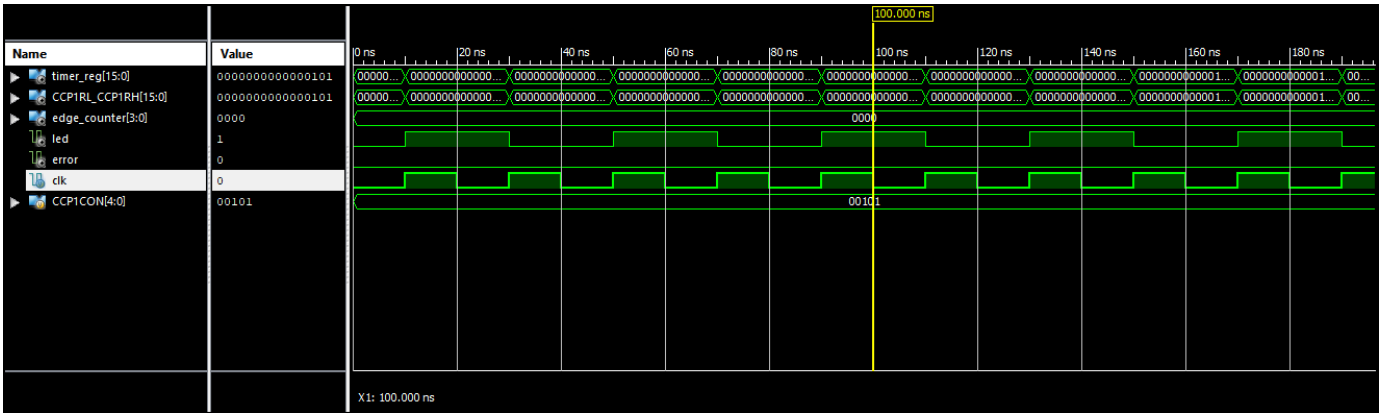
```
    initial begin
        // Initialize Inputs

        clk = 0;
        CCP1CON = 5'b00110; //Control register

        // Wait 100 ns for global reset to finish

        // Add stimulus here

    end
    initial begin
        forever
        #10 clk=~clk;

    end


endmodule
```

Simulation Results

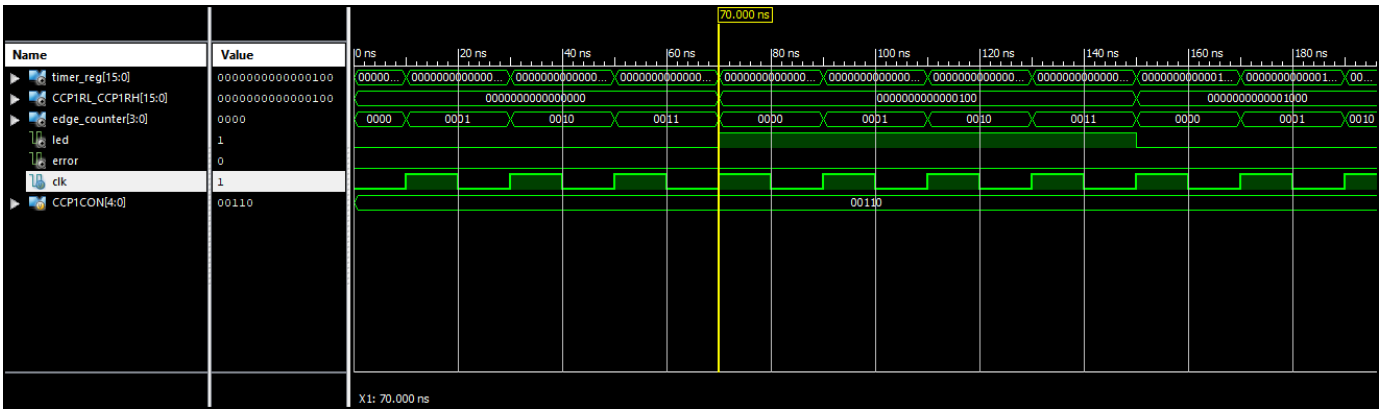
Falling edge (00100)



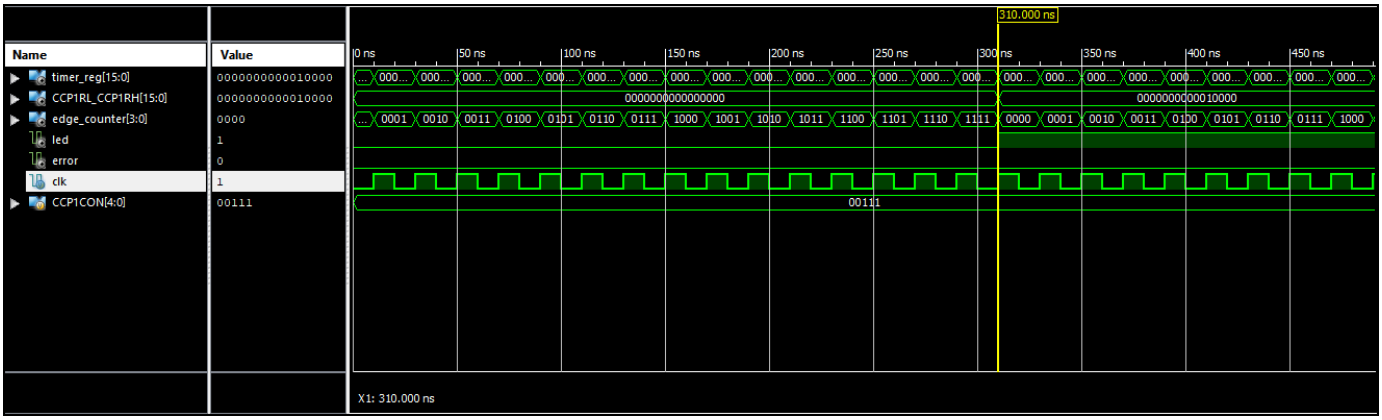
Rising edge (00101)



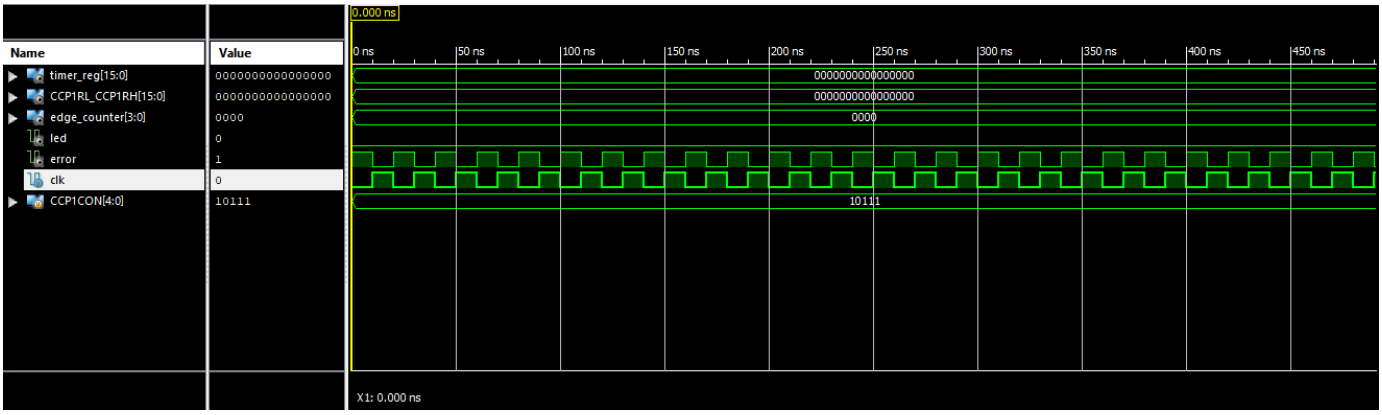
4th Rising edge (00110)



16th Rising edge (00111)



Error Indication in case of Wrong Combination



Hardware Implementation:

```
NET "CCP1CON[0]" LOC = "G18";
NET "CCP1CON[1]" LOC = "H18";
NET "CCP1CON[2]" LOC = "K18";
NET "CCP1CON[3]" LOC = "K17";
NET "CCP1CON[4]" LOC = "L14";
NET "led" LOC = "J14";
NET "error" LOC = "R4";
NET "clk" LOC = "B8";
```

Since there is a lack of output leds on the FPGA Spartan3E so we will not do that in hardware and only show the output of the led toggling on rising and falling edges of the input clock signal The. Error indication is also tested for hardware.

```
module ccphardware(
    input  clk,

    output reg led=0,
    output reg error=0,
    input [4:0] CCP1CON
    );
     reg [3:0] edge_counter=0;
     reg [23:0] count=0;
     reg nclk=0;
     reg cclk=1'b1;
     always@(posedge clk or negedge clk)begin
     count <= count +1;
     if (CCP1CON==5'b00100)begin
     cclk<=~cclk;
     nclk<=cclk;
     end else
     nclk<=clk;
     end
```

```
       always@(posedge nclk) begin
      if (CCP1CON[4] == 1'b0)begin
      error =1'b0;
       case (CCP1CON)

      5'b00100 : begin
           led = ~led;
       end
      5'b00101 : begin
           led = ~led;
       end


      5'b00110 :begin
           if(edge_counter == 3)begin
            led = ~led;
           edge_counter = 0;

    end
    else
           edge_counter = edge_counter+1;
    end
```

```
     5'b00111 :begin
            if(edge_counter == 15)begin
            led = ~led;
            edge_counter = 0;
    end
    else
            edge_counter = edge_counter+1;
    end
    endcase
    end
    else begin
             led = 1'b0;
             error = ~error ;
    end
    end


endmodule
```

Conclusion

In conclusion, the Verilog implementation of the Capture module within the Peripheral Interface Controller (PIC) using the Capture/Compare/PWM (CCP) architecture on an FPGA has been successfully realized. The detailed analysis of the Verilog code, architecture, and performance indicates a robust and efficient design. The FPGA-based CCP module demonstrates precise event capture capabilities, meeting the specified timing requirements for real-time applications.