Name: Lab Section:

CprE 288 Fall 2012 – Homework 7
Due Thu. Oct. 18 in the class

Notes:

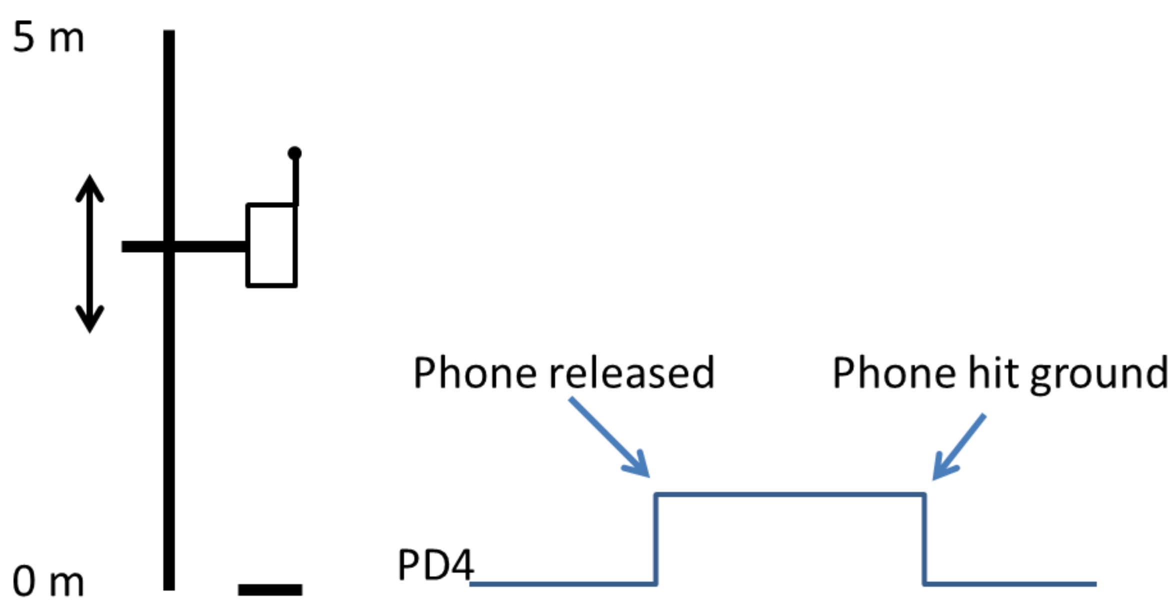
- **Start early on homework**
- Homework answers must be typed using a word editor. Hand in a hard copy in the class.
- Late homework is accepted within three days from the due date. **E-mail late homework to both of the grading TAs, Min Sang Yoon (my222@iastate.edu) and Zhen Chen (zchen@iastate.edu)**. *Late penalty is 10% per day (counting from 10:45am of the due date if you are in the morning class or 2:10pm if you are in the afternoon class).*

Question 1: Cell phone drop results (25 pts)

Students are to use the mechanism below to raise a cell phone between 0 to 5 meters. They must guess how high they can drop the phone from without breaking it. The mechanism has no markings to indicate how high the phone is.

When the phone is released a switch closes setting the TIMER1 Input Capture pin (Port D bit 4, i.e. PD4) to 1. When the phone hits the ground the switch opens, setting PD4 to 0.

Your job is to program the ATMega128 to send a message to putty saying if the phone has broken or not. Part b) gives specifics on what to send to putty using UART0.



Name: Lab Section:

a) Write C code to setup the 16-bit TIMER1 configuration registers for the ATMEGA128 as follows (5 pts)

- 15.625 KHz tick rate
- Input Capture interrupt enabled
- Timer Overflow interrupt enabled
- Noise cancellation disabled

```
init_TIMER1()
{
    TCCR1A = 0;
    TCCR1B = 0x45;  //Noise cancellation disabled, positive edge trigger, and CS=101 for presaler  1,024
    TCCR1C = 0;
    TIMSK = (1<< TICIE1) | (1<<TOIE1);  // Enable Input Capture and Timer Overflow interrupts
}
```

b) Write a C program that will Transmit the following messages from UART0, each time an cell phone is dropped. (20 pts)

SAFE, if the phone was dropped from less than 1 meter

WARN, if the phone was dropped from 1 to 3 meters

BUST, if the phone was dropped from greater than 3 meters.

Assume UART0 has been configured appropriately, and no UART interrupts are to be used.

Name: Lab Section:

```
// Input Capture ISR
ISR(TIMER1_CAPT_vect)
{
    switch (state)
    {
        case LOW:
                rising_time= ICR1;
                TCCR1B &= ~ (1 << ICES1);   // detect falling edge
                state=HIGH;
                break;
        case HIGH:
                 falling_time= ICR1;
                 TCCR1B &=  1 << ICES1;   // detect rising edge
                 state=DONE;          // tell main to compute time
                 break;
        default: break;
    }
}


// Timer 1 overflow ISR
ISR(TIMER1_OVF_vect)
{
     // The Timer will not overflow more than once
     // because the maximum measured time will be well under
     // 4 seconds. For a max height of 5 meters, the phone will
     // take just over 1 second to fall.  Thus, overflow
     // processing is not needed.  But it is fine if you do have
     // code to account for overflow
}


// Transmit a message over UART0
// This is a helper function. You could have placed code in main
// but this is much cleaner
send_UART(char *my_msg)
{
  int index = 0;

  while(my_msg[index] != 0)  // Check for NULL char
  {
    while(!( UCSR0A & (1<<5)) ) // Wait for UART ready for transmission
    {}
    UDR0 = msg[index];  // Send a character
    index++;
  }
}
```

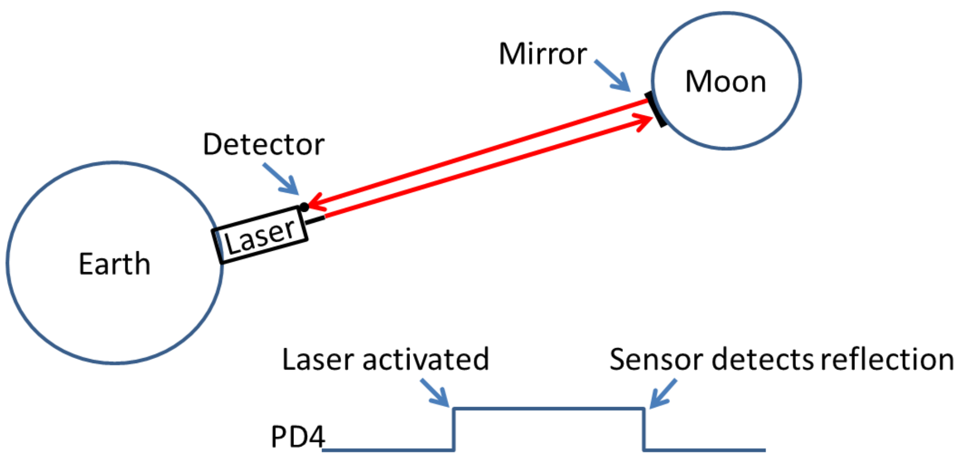
Name: Lab Section:

```
volatile enum {LOW, HIGH, DONE} state = LOW; // Initialize to LOW
volatile unsigned int rising_time;
volatile unsigned int falling_time;

main()
{
  unsigned int pulse_count; // count number of clock cycles
  float drop_time;          // Amount of time phone falls
  float phone_height;       // d = 1/2 * a * t^2, initial phone height
  char my_SAFE[] = "SAFE";
  char my_WARN[] = "WARN";
  char my_BUST[] = "BUST;

  init_TIMER1();

  while (1)
  {
    while(state != DONE){}   // Wait for a rising and falling edge
                             // to occur
    state = LOW;             // Reset state to LOW
    pulse_count = falling_time - rising_time;
    drop_time   = .000064*pulse_count;
    phone_height = .5 * 9.8 * drop_time * drop_time; // compute height


    // Transmit message over UART

    if(phone_height < 1.0)
    {
      send_UART(my_SAFE); // Print SAFE
    }
    else if(phone_height > 3.0)
    {
      send_UART(my_BUST); // Print BUST
    }
    else
    {
      send_UART(my_WARN); // Print WARN
    }

}
```

Name: Lab Section:

Question 2: Timer Accuracy (10 pts)

A laser that works similar to your Lab 7 Ping sensor is used to measure the distance to the Moon from Earth. The laser is fired at a mirror placed on the Moon, and a sensor attached to the laser detects when the reflection arrives back to the laser. As shown below. This is similar to an actual method used for measuring the Earth-Moon distance, see:
http://en.wikipedia.org/wiki/Lunar_Laser_Ranging_experiment



When the laser is activated, TIMER1's Input Capture pin is set to a 1. When the sensor detects the reflection, this pin is set to 0. A program has been written that uses Input Capture to compute the distance between the Earth and the Moon.

Given that Input Capture measures time in Timer ticks (i.e. clock cycles), how different can the programs calculation of the Earth-Moon distance be from the actual distance?

a) Explain what causes the error in the measured distance

Input Capture computes time from the number of clocks between events. If an event does not occur exactly on a positive clock edge, then there will be error in the measured time.

For example, if the "Sensor Detect" event happens right after a positive clock edge, then this time will be too long by nearly 1 clk cycle. And if the "Laser activated" event happens right after a positive edge, then this time will be too short by nearly 1 clk cycle.

Thus a worst case time measurement error will occur when "Laser activated" occurs exactly on a positive edge of the Timers clock, and "Sensor Detect" occurs right after a positive edge. This gives a maximum error in measuring time of 1 clk tick.

***Note: If a student says 2 clk cycles, then give full credit. They are thinking in the correct direction.**

Name: Lab Section:

Since:

Distance = Velocity * Time

Having error when measuring time directly impacts one's calculation of distance.

b) Compute the maximum error in distance for each **legal** prescaler of TIMER1.

First compute the maximum error in time (i.e. period of one Timer tick) for each prescalar, then compute how far light can travel in that amount of time (velocity of light is 3x10^8 m/s)

Prescalar 1: Freq = 16 MHz, Period = 1 / 16 MHz

Distance = 3x10^8 * 1/16 MHz = 18.75 m // height of a 5 story building

Prescalar 8: Freq = 2 MHz, Period = 1 / 2 MHz

Distance = 3x10^8 * 1/2 MHz = 150 m

Prescalar 64: Freq = 250 KHz, Period = 1 / 250 KHz

Distance = 3x10^8 * 1/250 KHz = 1,200 m

Prescalar 256: Freq = 62.5 KHz, Period = 1 / 62.5 KHz

Distance = 3x10^8 * 1/62.5 KHz = 4,800 m

Prescalar 1024: Freq = 15.625 KHz, Period = 1 / 15.625 KHz

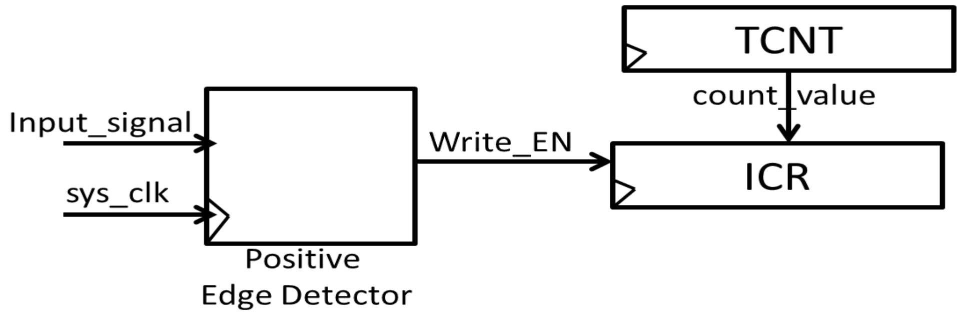
Distance = 3x10^8 * 1/15.625 KHz = 19,200 m // more than twice height of Mt. Everest

****Note: If a student skipped the prescalar of 1 it is fine**
****Note: If a student assumed worst case time is 2 clks it is fine also**

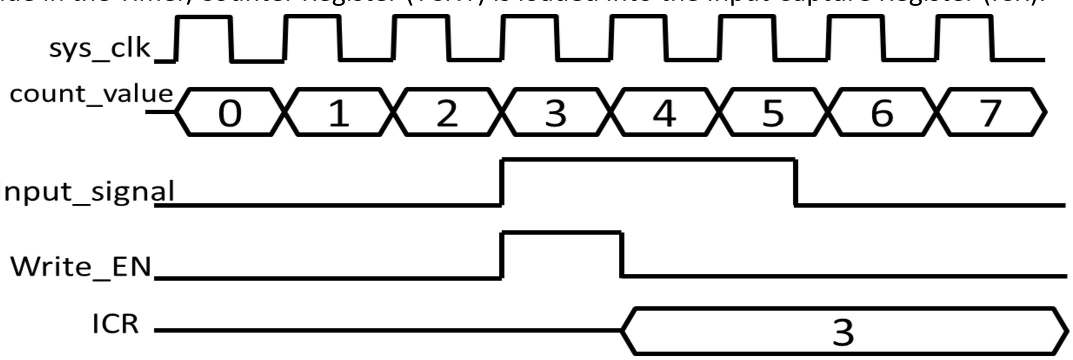
Name: Lab Section:

Question 3: Edge Detection (15 pts)

The following simple block diagram illustrates how one may implement an Input Capture circuit.

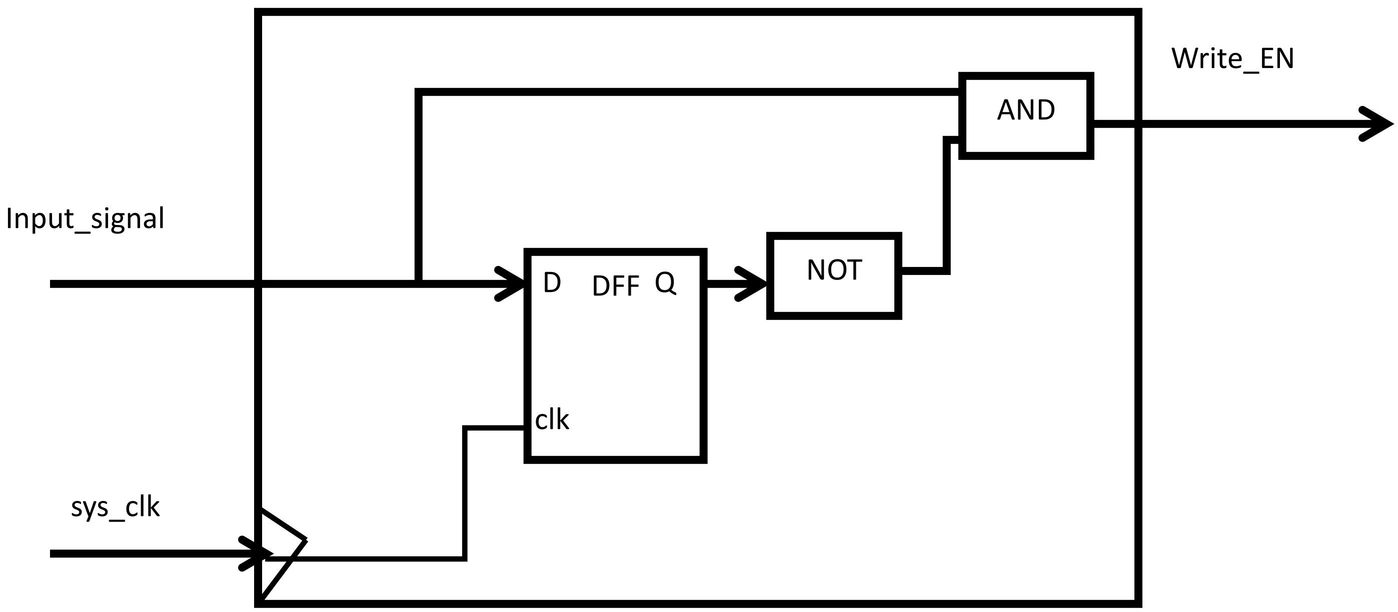


An example timing diagram for capturing the rising edge of Input_signal is given below. In summary, on the occurrence of a positive edge on Input_signal: 1) Write_EN pulses ‘1’ for one sys_clk cycle, and the value in the Timer/Counter Register (TCNT) is loaded into the Input Capture Register (ICR).



Name: Lab Section:

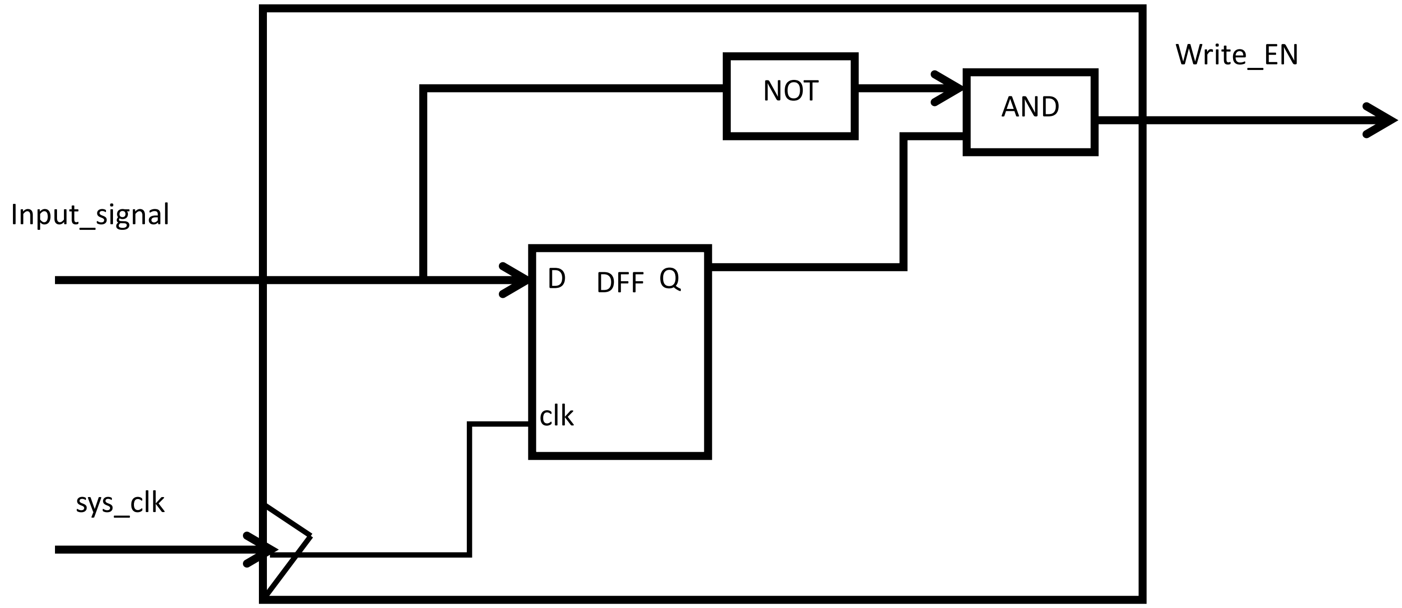
a) Draw out a digital circuit to implement an Edge Detector that creates a 1-sys_clk-wide pulse on Write_EN to load the TCNT register into the ICR register when a rising edge occurs on Input_signal. The only components you can use are D-Flip Flops, and AND, OR, NOT gates (5 pts)



Positive Edge Detector

Name: Lab Section:

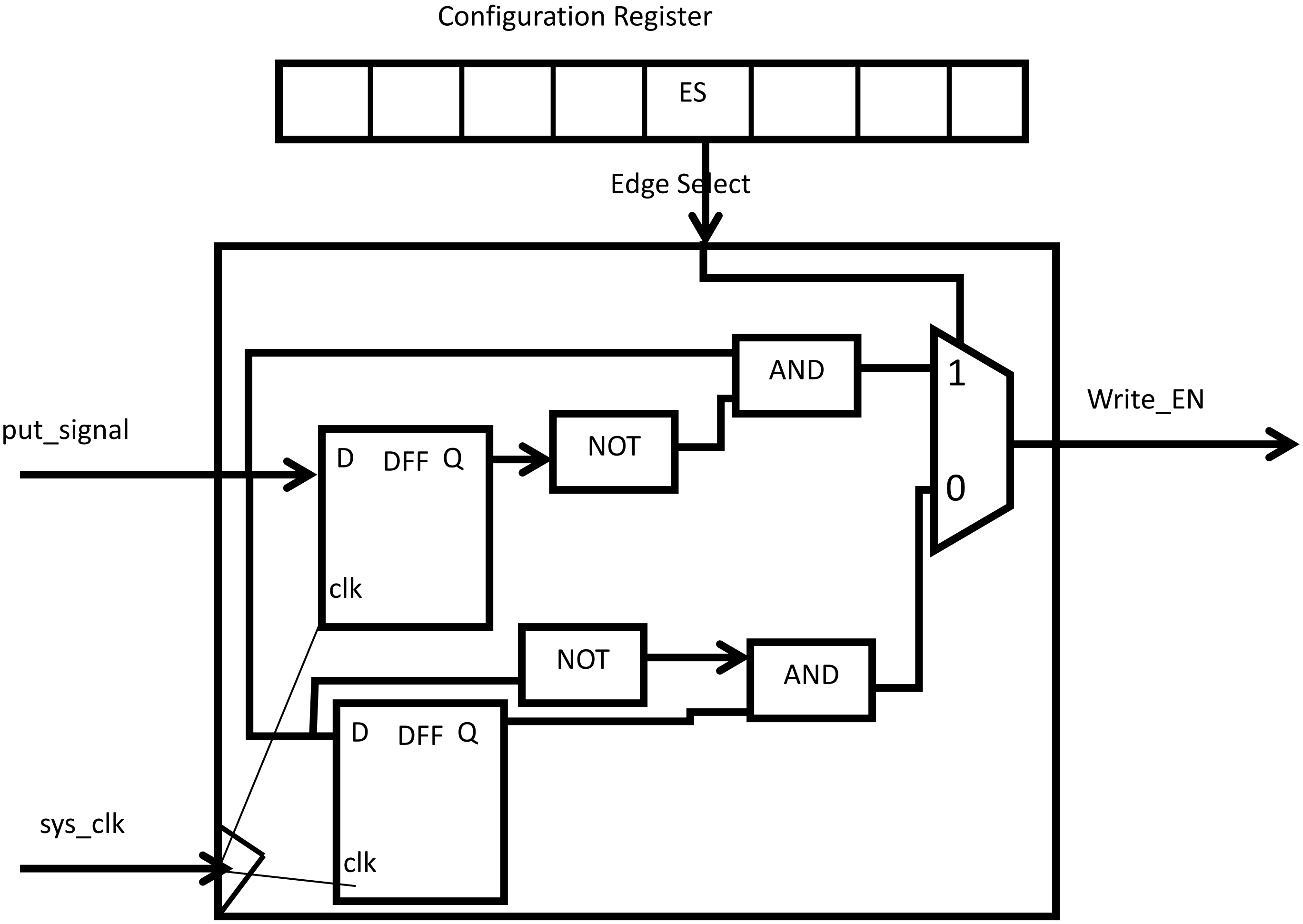
b) Repeat a) for an edge detector that creates a Write_EN pulse for detecting the falling edge of Input_signal. (5 pts)



Negative Edge Detector

Name: Lab Section:

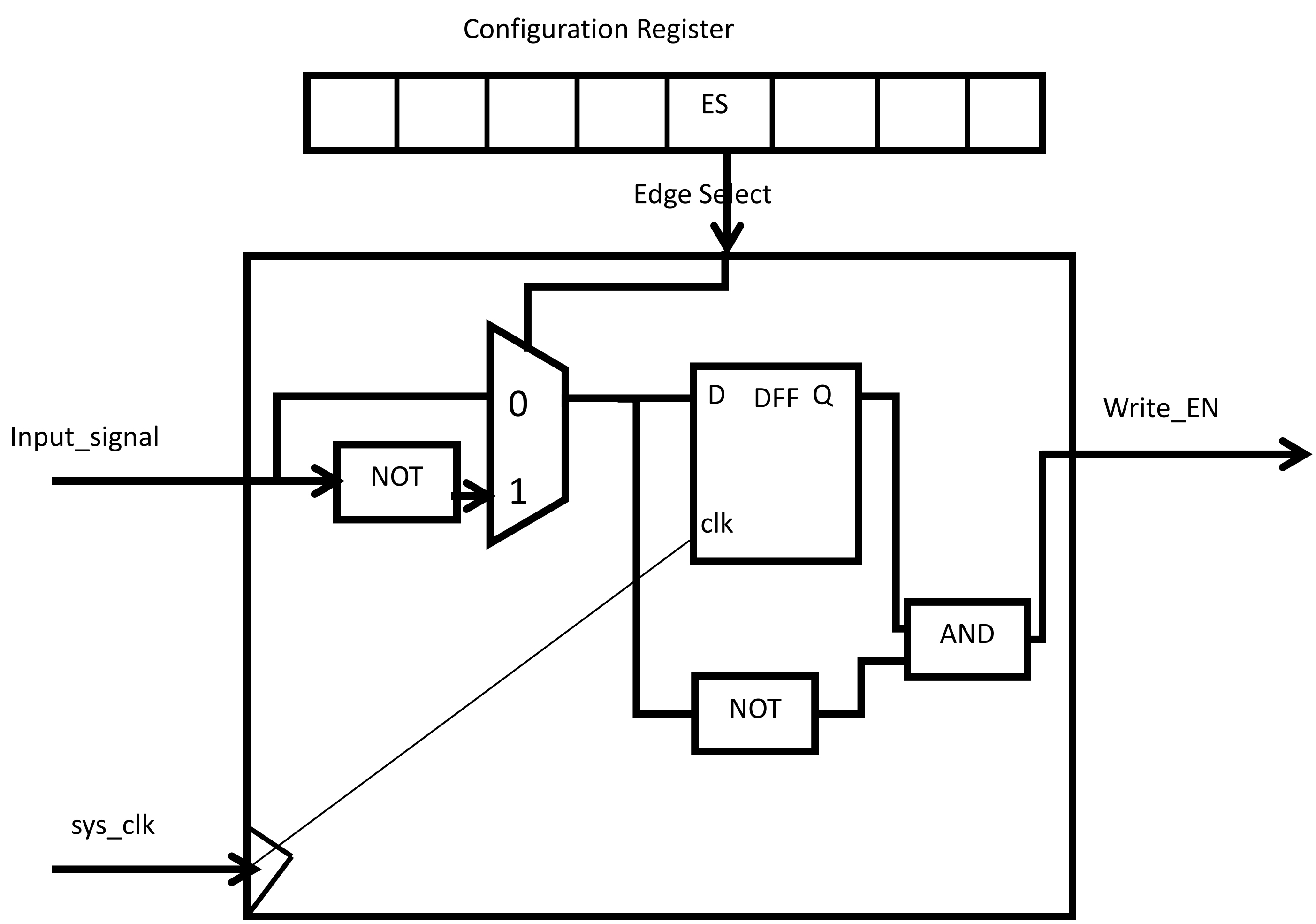
c) The ATMEGA128 allows for detecting either the positive edge or negative edge of an input by configuring the "Edge Select" bit of a configuration register. Draw the digital circuit to allow the Edge Detector to detect a positive edge when ES=1, and a negative edge when ES=0. You may now use multiplexers in addition to D-Flip Flops, and AND, OR, NOT gates (5 pts)



Configurable Edge Detector

Or

Name: Lab Section:



Configurable Edge Detector

Name: Lab Section:

Question 4: Software implemented Input Capture (15 pts)

a) Assume the ATMEGA128 does not have Interrupts or Input Capture hardware. Write a C program to save TIMER1's count value when a positive edge event occurs on PortD, pin4 (9 pts)

```
int main(void)
{
  unsigned int rising_time;

  while(1)
  {
    if( ( PIND& 0b00010000)==0)  // check if the Port D, bit 4 is 0
    {
      while( !(PORTD & 0b00010000)) // now wait for it to go to 1
      {}
      rising_time = TCNT1; // store time that positive edge occurred
    }
  }

  return 0;  // Program should never get here
}
```

b) Describe two disadvantages of your software-implemented input capture program, as compared to using Input Capture hardware and Interrupts. (6 pts)

1. **It does not allow other parts of the code to run, while it is running.**

2. **Another issue the can occur with the "software only" approach is that an ISR could occur after detecting a positive, but before the TIMER value is read. Thus adding to the error of the TIMER value captured.**