1. (30%) UNO digital pins 0 to 7 are connected to Port D of the ATmega328 microcontroller

while digital pins 8 to 13 are to Port B. Without using pinMode( ), write down as few C

statements as possible on DDRB/DDRD/PORTB/PORTD to achieve the configuration of

• digital pins 2, 4, and 5 as OUPUT pins,

• digital pins 7 and 8 as INPUT pins,

• digital pins 3 and 11 as INPUT\_PULLUP pins, and

• all the other digital pins should not be changed.

*DDRD = DDRD | B00110100;*

*DDRD = DDRD & B01110111;*

*PORTD = PORTD & B01111111;*

*DDRB = DDRB & B11110110;*

*PORTB = DDRB & B11111110;*

*PORTB = PORTB | B00001000;*

2. (20%) Given a 16 MHz UNO board, is it possible to have the timer 1 overflow less than

once a second? If yes, conceptually how to do it? If no, why not?

*Yes; Using a prescaler value greater than 256 will cause the timer to overflow less than once per second.*

3. (50%) The keypad library used in Lab 1 requires the constant scanning of keypads. A lower

power design would scan the keypad only when some keys have been pressed down.

(1) Explain conceptually how to modify the keypad library to implement such a lower power

design. Note that the keypad library source codes are inside the folder, My

Documents\Arduino\libraries\keypad\src.

*By using a pin change interrupt, the board can sleep until a key is pressed, which will wake it. This can be done by setting all column pins OR all row pins to INPUT\_PULLUP and set an interrupt to trigger when a key is pressed (pressing a key should connect the HIGH in a row to the LOW in a column or vice versa, triggering an interrupt and waking the board).*

(2) Using the HelloKeypad source code as an example, explain conceptually how your

modified library can be used.

*An ‘If’ statement should be added to put the board to sleep if the keypad is idle. When a key is pressed, the code in the library would interrupt and wake the board and allow the key to be read.*

// Input capture interrupt  
#define icpPin 8        // ICP input pin on arduino  
// some variables to help see that something is happening in the interrupt handlers  
volatile unsigned int Value;     // this stores the current ICR1 value  
volatile unsigned int MinValue;       
volatile unsigned int MaxValue;  
volatile unsigned int Overflows;  
volatile unsigned int PulseCount;  
float pulseWidth;

float dutyCycle;  
  
/\* Overflow interrupt vector \*/  
ISR(TIMER1\_OVF\_vect){                 // here if no input pulse detected   
  Overflows++;                       // incriment overflow count    
}  
  
/\* ICR interrupt vector \*/  
ISR(TIMER1\_CAPT\_vect){  
  TCNT1 = 0;                            // reset the counter    
  if( bit\_is\_set(TCCR1B ,ICES1)){       // was rising edge detected ?     
       digitalWrite(outPin,HIGH );      // yes, set our output pin high to mirror the input  
  }  
  else {                                // falling edge was detected   
       Value = ICR1;                    // save the input capture value  
       PulseCount++;  
       if(Value < MinValue)             // update min or max values as appropriate     
           MinValue = Value;  
       if (Value > MaxValue)  
           MaxValue = Value;      
  }       
  TCCR1B ^= \_BV(ICES1);          // toggle bit value to trigger on the other edge      
}  
  
void setup() {  
 beginSerial (19200);  
 pinMode(outPin, OUTPUT);               // declare the ledPin as an OUTPUT  
 pinMode(icpPin, INPUT);             // ICP pin (digital pin 8 on arduino) as input  
 MinValue =    30000;                   // Initialise with extreme values   
 MaxValue =    0;  
 TCCR1A = 0 ;            // this register set to 0!  
 TCCR1B =\_BV(CS11);     // NORMAL MODE!!, prescaller 8, rising edge ICP1 - this works  
 TCCR1B |= \_BV(ICES1);     // enable input capture       
 TIMSK1 =  \_BV(ICIE1);     // enable input capture interrupt for timer 1  
 TIMSK1 |= \_BV(TOIE1);    // enable overflow interrupt to detect missing input pulses   
 Serial.print("Finished setup\r\n");  
}  
  
void loop() {  
 pulseWidth = ((MinValue/2) + (MaxValue/2))/2;

dutyCycle = ((1 / (1000 / PulseCount)) \* pulseWidth) \* 100;          
 Serial.print(“Duty Cycle: “ + dutyCycle);   
 /\* reset variables ready for the next reading \*/     
 PulseCount = Overflows = 0; // reset counts  
 MinValue =    30000;        // set values to extremes  
 MaxValue =    0;         
 delay(1000);             // wait 1 second for next update  [/font]              
}