**BRANDON VO**

Due F 11/10

The description below will be updated with additional information so check this page regularly.

**Project specifications:**

* Objective: measure and display the period and duty cycle of a square signal connected to the AVR microcontroller.
* Input signal can be as low as 1mHz and as high as 100MHZ. Input duty cycles vary from 0% to 100% in increments of 1%.
* The two measurements must be displayed on the LCD screen concurrently. The period must be displayed in seconds with units "s" at the end. Use "%" for the duty cycle measurement. For the period you may use "u" or "m" for micro or milli.
* The LCD 4-bit mode is preferred, but 8-bit mode is OK.
* Measurements must be continuous: this means that measurements are constantly being updated, such that if the input changes, the measurements will automatically be updated.
* You should build your project in Simulation IDE and use Microchip Studio for programming.
* Use the function/clock generator and the oscilloscope in Simulation IDE to test your design.

**Process:**

1. Setup the Timer ICP
2. Detect 2 consecutive positive edges (or negative edges) for a period count.
3. Detect a positive and a negative edge for duty cycle count.
4. Convert period count to seconds and convert the duty to a percentage.
5. Convert numbers (period and duty cycle) to strings
6. Setup the LCD and display the strings.
7. Repeat step 2, infinitely.

**Algorithms:**

1. Setup the Timer ICP:

* Consult the notes on how to setup ICP on a 16-bit Timer.

2. Detect 2 consecutive edges for a period count:

* Consult the notes on how to detect the 2 consecutive rising edge for period calculation.
* The difficulty lies in setting the prescaler. There are 2 options for the prescaler: (a) Set the prescaler to 1 and rely on the overflow for calculations  
  (b) dynamic (changing) prescaler, that changes as needed. (Hint: decrease the prescaler if there's an overflow).
* It is advisable to use long int variable for the period and overflows to help with with calculations.

4. Convert period count to seconds and convert the duty to a percentage:

* Period (s) = **(Period count/16MHz) \* prescaler**
* Duty % = (duty count/period count) \* 100
* Use long int variables definition for Period and Duty
* Perform the multiplication BEFORE division (100/1000 = 0 as an integer!)

5. Convert number to strings:

* The numbers are the period and duty cycle calculations.
* Use successive division and modulus to isolate each digit, e.g.
  + num = 1234
  + d1 = num%10 = 4, d2 = num/10%10 = 3, d3 = num/100%10=2, ...
  + Convert each digit to ASCII (add 0x30) and store in a string

**Submission:**

Submit **1 zip** file that contains

* a report document,
* a simulation IDE file (.sim)
* a C program (.C)
* a hex file for the simulation (.hex)

**Report:**

In your report include the following (don't do a full report format):

* Clearly state:
  + if your code use overflow to detect low varying signals
  + automatically changes the prescaler as needed.
* Schematic showing all components and their pin connections (ie screenshot you simlation IDE). The circuit may be too big to fit on a page, so split it into separate images. Label wires if needed (e.g if they're cut-off).
* Derive:
  + MAX frequency that can be detected accurately.
  + MIN frequency that can be detected accurately.
  + The max/min duty cycle that can be detected at the max frequency
  + The max frequency at which duty cycle is properly detected in 10% increments
  + The max frequency at which duty cycle is properly detected in 1% increments.
* Code:
  + Code will not be graded if you don't follow instruction below:
    - INDENT code
    - DON'T double space the code or use a font larger that 11pts
    - DON'T use running comment lines (that continue on the next line)
* Test results (image screenshots):  
  + MAX freq detected properly
  + MIN freq detected properly
  + MAX freq at which 1%, 99% duty is captured and detected properly
  + MAX freq at which 5%, 95% duty is captured and detected properly
  + MAX freq at which 10%, 90% duty is captured and detected properly
  + Screenshot should clearly show the settings on the input and output of the scope.

**Project 3 Report**

Explanation:  
Code uses overflow, does not change prescaler

**Circuit Diagram**

A computer screen shot of a blue circuit board

Description automatically generated

**Max Freq Detected Accurately ( ~25 KHz, 50% duty)**

**A screenshot of a computer

Description automatically generated**

**­**

**Min Freq Detected (~200 Hz, 50% duty)**

**A computer screen shot of a computer

Description automatically generated**

**Max Freq Detected with 99% Duty (~2 KHz)**

**A computer screen shot of a computer

Description automatically generated**

**Max Freq Detected with 95% Duty (~12 KHz)**

**A computer screen shot of a computer

Description automatically generated**

**Max Freq Detected with 90% Duty (~25 KHz)**

**A computer screen shot of a circuit board

Description automatically generated**

**Code**

// Project 3

#define *F\_CPU* 16000000L //16MHz

#include <avr/io.h>

#include <util/delay.h>

#include <stdint.h>

#include <avr/interrupt.h>

volatile *uint16\_t* tovCnt = 0; // use volatile for interrupt variables

void setupLCD(void)

{

DDRF = 0xFF; //LCD 8bit data

DDRK = 0x03; //PTK0= En, PTK1=RS

char2LCD(0x38, 0); // set 8-bit data, 2-line display, 5x8 font

char2LCD(0x0F, 0); // turn on display, cursor, blinking

char2LCD(0x06, 0); // move cursor right

char2LCD(0x01, 0); // clear screen, move cursor to home

}

void char2LCD (unsigned char data, unsigned char IR\_DR)

{ PORTK = IR\_DR; // set instr or character

PORTK = IR\_DR|0x01; // pull E signal to high

PORTF = data;

PORTK = IR\_DR;

}

void str2LCD (unsigned char \*ptr)

{

while (\*ptr) //ascii-Z strings (null-terminated)

{

char2LCD(\*ptr, 2);

// \_delay\_ms(500);

ptr++;

}

}

// Inputs a char at a specific area of a string to the LED

void input\_char(*int8\_t* character, *uint8\_t* \*str, *int8\_t* len)

{

*int8\_t* i;

for (i = len - 1; i >= 0; i--){

str[i] = character;

}

}

// Converts stuff into arrays/strings for LCD output

void output\_LCD(*uint32\_t* num, *uint8\_t* \*str, *int8\_t* len)

{

*int8\_t* i;

for (i = len -1; i >= 0; i--){

str[i] = (num % 10) + 0x30;

num /= 10;

}

}

ISR (TIMER4\_OVF\_vect) {

tovCnt++;

}

int main(void)

{

setupLCD();

*uint16\_t* period, duty\_cycle, time\_start, time\_end;

TCCR4A = 0;

DDRL &= 0XFE;

TIFR4 = 0X2F;

TIMSK4 = 0X01;

sei();

while(1)

{

char2LCD(0X01,0);

str2LCD("Period: ");

str2LCD(" us");

char2LCD(0XC0,0);

str2LCD("Duty Cycle: %");

// Period

TCCR4B = 0X43; // Rising Edge Flag

TIFR4 = 0X21; // clear ICF4 and OVF

TCNT4 = 0;

tovCnt = 0; // Reset overflow counter

while(!(TIFR4 & 0x20)); // 1st Rising Edge

*uint16\_t* period\_start = tovCnt \* 65536 + ICR4;

TIFR4 = 0x21; // clear ICF4 and OVF

tovCnt = 0; // Reset overflow counter

while(!(TIFR4 & 0x20)); // 2nd Rising Edge

*uint16\_t* period\_end = tovCnt \* 65536 + ICR4;

*uint16\_t* period = period\_end - period\_start;

// Duty Cycle

TIFR4 = 0x21;

TCNT4 = 0;

tovCnt = 0; // Reset overflow counter

while(!(TIFR4 & 0x20)); // 1st Rising Edge

*uint16\_t* time\_start = tovCnt \* 65536 + ICR4;

TIFR4 = 0x21; // clear ICF4 and OVF

tovCnt = 0; // Reset overflow counter

TCCR4B = 0x03; // Falling Edge Flag

while(!(TIFR4 & 0x20)); // Falling Edge

*uint16\_t* time\_end = tovCnt \* 65536 + ICR4;

*uint16\_t* duty\_cycle = time\_end - time\_start;

// Duty Cycle and Period

duty\_cycle = (duty\_cycle \* 100) / period;

period = (period \* 64) / 16;

output\_LCD(period," us",5);

output\_LCD(duty\_cycle," %",3);

*\_delay\_ms*(300);

}

}