# Rochester Institute of Technology SWEN 563/CMPE 663/EEEE 663

# Project 1 - Timing

**Overview:**

Using the STM32 Nucleo board, design and implement an embedded, bare-metal (no operating system) program that will measure the inter-arrival times (rising edge to rising edge) of one thousand pulses. A measurement increments the count in a histogram consisting of 101 “buckets” (one bucket per microsecond) centered around the expected arrival time. After measuring 1000 pulses, the program will print a listing of the non-zero counts of each histogram bucket. As an example: at 1khz, the inter-arrival time between pulses is *expected* to average around 1.0 millisecond, but the listing should represent the range of 101 “buckets” ranging from 950 to 1050 microseconds. The expected arrival time must be user configurable via the virtual terminal.

**User interface:**

After successfully completing the POST routine (see below) design your user interface to operate as follows (using the virtual terminal):

* On startup, the program shall display a default expected inter-arrival time. The user shall accept this default value, or be able to change it to any other value between 100 and 10000 microseconds (100 Hz to 10 kHz).
* The program shall measure all inter-arrival times at the expected value +/- 50 microseconds with 1 microsecond resolution.
* After reviewing and possibly changing the limits the program issues a start prompt and waits for the user to press the Enter key.
* After completing the 1000 measurements (1001 rising edges) the program displays every non-zero via the PuTTY logging feature. Do not display entries with 0 counts.
* The list of counts displayed should have the time in microseconds as the first column and the count in the second column.
* The list of counts should be in ascending time order. A typical display would have 3 lines like this:  
   998 5  
   999 950  
  1000 45
* After completing the display of the non-zero entries provide the option to run again with either the same or different limits.
* Test the oscilloscope test output (it is 1 kHz) and compare your results with the signal generator.

**POST:**

Perform a Power On Self-Test (POST) when the program starts. This test must confirm that the GPIO port is seeing pulses at least once in 100 milliseconds. If it fails this test, give the user the option to try the POST again. If the POST is successful then proceed to the normal user interface operation described above.

Note that this program will be relevant to subsequent projects so be sure to use good software development practices. Refer to the C programming standard for the course.

**Report and Submission:**

In addition to the standard report requirements the report for this project must include the following:

* Screenshot of results for two significantly different signal frequencies.
* Comparison of your results at 1 kHz when using the signal generator and using the oscilloscope test signal.

Submit your report and source code to the Project 1 Dropbox in myCourses.

The project report and grading should conform to the Report Specification, and grading will conform to the Project Grading Criteria.   (myCourses > Content > Course Syllabus).

**This project will be demonstrated in class to the instructor or Course Assistant. The report is due on the same day as the project.**

For the next class…

DESIGN

For the next class, be prepared to answer the following questions about Project 1…

* What is a timer peripheral? What functionality does it provide?
* What 4 timer modes are supported on our STM32 microcontroller? What mode will you use for this lab? In general, where do you find detailed information on the STM32 microcontroller peripherals?
* What CLK sources may be selected to drive the prescaler of an STM32 timer? What is the finest resolution clock source available to drive the prescaler (CK\_PSC)? What is the accuracy of this clock? Why is there a prescaler before the counter clock (CK\_CNT)? For the STM32, where do you find electrical information (power, tolerances, pinouts, packages, memory configuration, etc.)?
* In this lab, what does a ‘bucket’ represent? How many buckets do you need to keep track of? What is the collection of buckets called?

When writing your code, you should follow this general template…

//////////////////////////////////////////////////////////////

// You may want to start with the UART demo project.

// Write the following methods…and invoke in main() as below…

// This is a high level sketch only. You will need to add more methods/variables to complete.

////////////////

**char** message[50]; // message to print to the user

**char** buffer[20]; // holds the user response

//////////////////////////////////////////////////////////////

// Function declarations

////////////////

// runs the power on self-test. Returns true if the test passes, false otherwise

**\_Bool** **power\_on\_self\_test**( **void** );

// initializes all variables prior to a measurement.

**int** **init\_measurement**( uint32\_t limit );

// measures timing of 1000 rising edges.

**int** **make\_measurements**( uint32\_t limit );

// print the non-zero bucket times and counts

**int** **print\_measurements**( uint32\_t limit );

// Captures 1 line of text from the console. Returns nul terminated string when \n is entered

**void** **get\_line** ( **void** \*buffer, **int** max\_length );

// Parses a line of user input into a new lower limit (unchanged if no response or invalid response)

**void** **get\_limit** ( **void** \*buffer, uint32\_t \*lower\_limit );

// initializes the timer (Similar to USART2\_Init(). Place in timer.c)

**void** **TIM\_Init**(**void**);

//////////////////////////////////////////////////////////////

// Embedded code usually consists of 2 components

// - The init section is run once at startup and initializes all low level drivers and modules

// - The main loop runs forever and calls the application tasks repeatedly.

////////////////

**int** **main**(**void**) {

uint32\_t lower\_limit = 1000 - 50; // the default lower limit in the problem statement

//////////

// Initialization executed once at startup

//////////

USART2\_Init(115200);

TIM\_Init();

**while**( power\_on\_self\_test() != 0)

;

//////////

// Main loop runs forever

//////////

**while**(1)

{

// 1. Print “Enter expected period or <CR> if no change”. Wait for user response

// print(message); // Consider using use print code from P0-UART

get\_line(buffer, **sizeof**(buffer));

// 2. Set a new expected interval time based on user input

get\_limit(buffer, &lower\_limit);

// 3. measure 1000 pulses

init\_measurement( lower\_limit );

make\_measurements( lower\_limit );

// 4. print time/count for non-zero counts

print\_measurements( lower\_limit );

}

}

//////////////////////////////////////////////////////////////

// Function implementation stubs

////////////////

// runs the power on self-test. Returns true if the test passes, false otherwise

**\_Bool** **power\_on\_self\_test**( **void** ) {

**return** 0;

}

// initializes all variables prior to a measurement.

**int** **init\_measurement**( uint32\_t limit ) {

**return** 0;

}

**int** **make\_measurements**( uint32\_t limit ) {

**return** 0;

}

// print the non-zero bucket times and counts

**int** **print\_measurements**( uint32\_t limit ) {

**return** 0;

}

// Captures 1 line of text from the console. Returns nul terminated string when \n is entered

**void** **get\_line** ( **void** \*buffer, **int** max\_length ) {

**return**;

}

// Parses a line of user input into a new lower limit (unchanged if no response or invalid response)

**void** **get\_limit** ( **void** \*buffer, uint32\_t \*lower\_limit ) {

**return**;

}

// initializes the timer (Similar to USART2\_Init(). Place in timer.c)

**void** **TIM\_Init**(**void**) {

**return**;

}