

CE-2812, Lab Week 7, Timer Input

1 PURPOSE

The purpose of this lab is to explore interrupts and implement an interrupt service routine.

2 PREREQUISITES

- The Nucleo-F446RE board had been mounted onto the Computer Engineering Development Board.

3 ACTIVITIES

3.1 BACKGROUND

The last two labs have dealt with generating waveforms with little or no programmatic intervention. This lab will focus on using the same peripheral, a timer/counter, to deal with input. The application will be to measure the frequency of an incoming square wave. This application will be added to the existing console program such that it can be invoked by a command, and will be able to operate concurrent with “background music” as implemented in the previous lab assignment.

In the simplest form, you might have the square wave signal trigger an external interrupt (EXTI). In the ISR you would grab a “timestamp” for the signal’s edge from a free-running counter. After at least two invocations of the ISR, you would be able to calculate the period of the signal and therefore would know the frequency. The only real issue with this approach is that interrupt latency and more importantly interrupt jitter can affect your result as you are reading the free-running counter within the ISR.

TIC to the rescue. The concept behind TIC (timer input capture) is that the free-running counter’s value will be copied into a register at the exact moment the edge is detected. Thus, when the ISR does run, the timestamp is accurate regardless of latency and jitter. Unlike TOC that can generate waveforms with no intervention once configured, TIC would normally require an ISR to service the timestamp before the next edge arrives and overwrites the stored register. Most of the general purpose timers on our STM32F446RE processor offer TIC functionality generally on the same pin that can also be used for TOC (labeled TIMx-CH1, etc). For those pins, TIC or TOC, not both...

3.2 GENERAL REQUIREMENTS

- Add a menu option to “measure frequency”
- The “measure frequency” function should:
 - Set-up timer with input capture mode
 - Collect samples for some period of time (at least 10 edges)
 - Calculate and report frequency information (in Hz) to console
 - Frequency reported should be average of samples taken
 - You must also report min/max of samples
- After measuring frequency, you should return to your command prompt and accept additional commands
- You should support measuring frequencies over a range of at least 50 Hz to 10 kHz

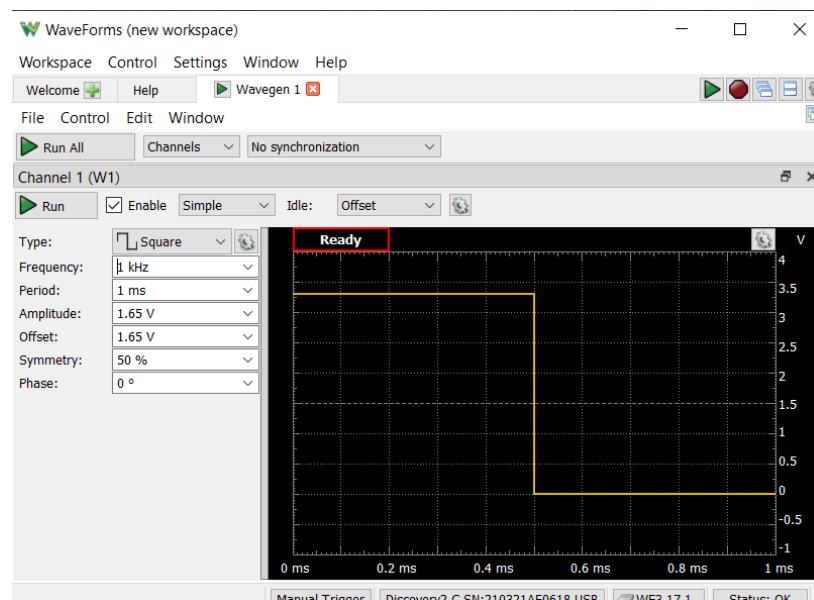
With a given input signal, run your measurement both with and without background music playing. You should get consistent results.

3.3 IMPLEMENTATION DETAILS

- You may choose any timer you wish that can map to a usable pin. TIM2 may be a good choice. You could theoretically use the rotary encode to test (although that may prove difficult to consistently generate a steady frequency). TIM2 Channel 1 is also available on PA15 which is otherwise unused.
- You probably want the timer/counter to count its full range. For this, you will want to set ARR to its maximum value.
- You should never stop the counter nor should you ever reset the count. Either of these will make your measurement inaccurate.
- You will need to use a frequency generator to generate a steady frequency for testing. The Analog Discovery devices work great for this. Be careful. The I/O pins generally prefer 0 to 3.3 volts which will need to be set in the AD.
- Be sure to read section 17.3.5 of the reference manual. It describes usage of the input capture function. There is an input filter function that should not really be needed when connected to a signal generator, so you can probably leave that at default settings.
- Do you need a pre-scaler? May, maybe not. Consider the measurement range specified and how many times the counter will need to count to cover that range.
- Do you need to use interrupts? Technically no. You could poll the CC1IF (if on channel 1) flag to see when an edge is detected and so forth. However, for this assignment you should use interrupts.
- The ISR will be invoked whenever an edge is detected. Within the ISR, it will then need to grab the value in CCR1 (if using channel 1) and store it. After the ISR has stored timestamps for at least ten edges, it can set a flag that it is done collecting data and disable itself.
 - In the meantime, your “mainline code” is polling the done flag and when the ISR is done collecting, it can retrieve the array of timestamps and calculate the frequency.
 - What if the ISR never sets the done flag? Perhaps you could have some sort of timeout...
- Do you need to handle overflow of the timer. Maybe, maybe not. See below.

3.4 USING ANALOG DISCOVERY SIGNAL GENERATOR

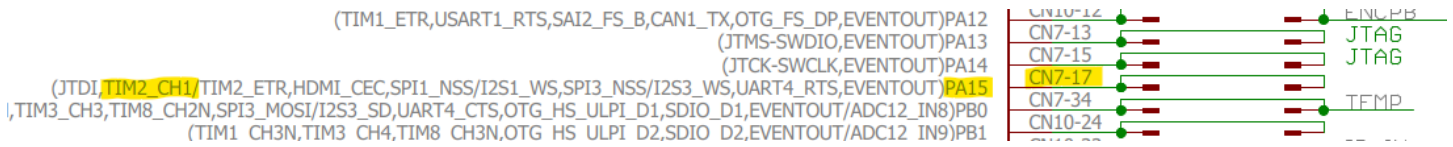
Set up the Waveform Generator like this:



Note the Amplitude and Frequency settings. These will produce a signal that varies between 0 and 3.3 volts to be compatible with our device’s input pins. Vary the frequency to test over the entire range. The signal will “appear” on the W1 wire (yellow). Be sure to also connect a black wire (GND) to an appropriate place on your dev board.

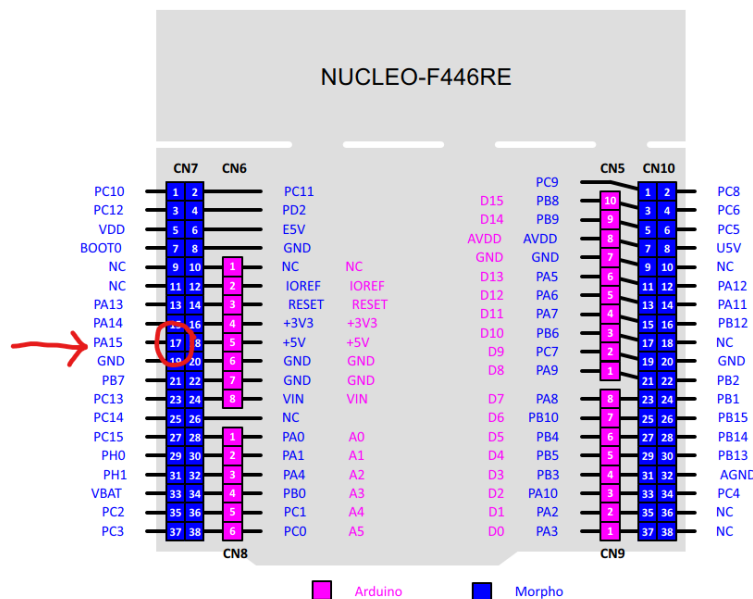
3.5 CONNECTING TO DEV BOARD

So, the schematic let's you pick a pin. Let's say you choose to use PA15 as the input, and thus will be using TIM2_CH1. Schematic shows:



Schematic shows that this signal is available on CN7-17. CN7 is a connector on the STM32 Nucleo board itself. You can find more information in the “STM32 Nucleo User Manual” which is in the Box folder. Be careful, and this manual covers many flavors of processors. It will be safest to search for our particular version – F446RE. Ultimately, you should refer to Figure 23:

Figure 23. NUCLEO-F446RE



This gives you a visual location for PA15 on CN7. You can also see nearby GND connections for the black wire.

3.6 HANDLING COUNTER OVERFLOW

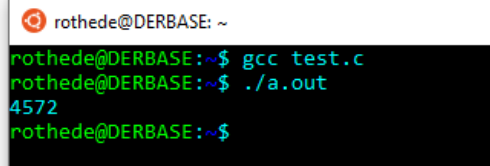
Since the counter is free-running, any two consecutive samples represent the period in clock ticks. So, if the first edge is 1527 and the second edge is at 3675, the difference is $3675 - 1527 = 2148$ ticks. If running with no prescaler, the time per tick is 62.5 ns, so the period here is $134.25 \mu\text{s}$ which in turn is a frequency of about 7449 Hz.

Now, what if one sample is right before the counter overflows, and the next one is after? As an example, let's say we are using a 16-bit counter that overflows at 65535 (which would be the value in ARR). Assume we have one sample before the overflow, say 65487, and one after the overflow, say 4523.

There are a few approaches. One, you could have another interrupt on timer update (overflow) and set a flag informing of the overflow. But, you really already know this because of the magnitude. As long as the lowest frequency you want to measure has a period less than the period of the counter's overflow, whenever a later sample is smaller than a previous sample, you can just add to the second sample to compensate. With the numbers above, since $4523 < 65487$, we know there must have been an overflow, and we can add 65536 to the second sample, then proceed with the subtraction to calculate period. $(4523 + 65536) - 65487 = 4572$ ticks.

But, if we use the correct data types, we do not even have to do this. In this example, if we are using `uint16_t` for each of these samples, the math will “underflow” and give me the correct answer. Try it:

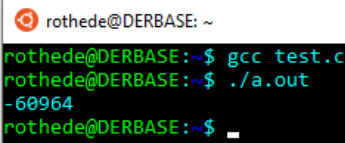
```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main(void)
5  {
6      uint16_t second = 4523;
7      uint16_t first = 65487;
8      uint16_t result = second - first;
9      printf("%d\n",result);
10     return 0;
11 }
```



```
rothede@DERBASE: ~
rothede@DERBASE:~$ gcc test.c
rothede@DERBASE:~$ ./a.out
4572
rothede@DERBASE:~$
```

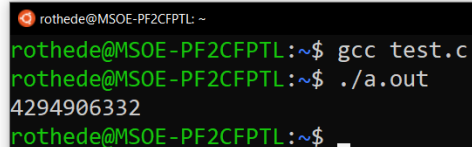
But be careful. If you do not assign result to correct type, it may promote. Check out this variation:

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main(void)
5  {
6      uint16_t second = 4523;
7      uint16_t first = 65487;
8      printf("%d\n",second-first);
9      return 0;
10 }
11
```



```
rothede@DERBASE: ~
rothede@DERBASE:~$ gcc test.c
rothede@DERBASE:~$ ./a.out
-60964
rothede@DERBASE:~$
```

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  int main(void)
5  {
6      uint16_t second = 4523;
7      uint16_t first = 65487;
8      printf("%u\n",second-first);
9      return 0;
10 }
```



```
rothede@MSOE-PF2CFPTL: ~
rothede@MSOE-PF2CFPTL:~$ gcc test.c
rothede@MSOE-PF2CFPTL:~$ ./a.out
4294906332
rothede@MSOE-PF2CFPTL:~$
```

4 DELIVERABLES

When completed:

1. Submit to Canvas a **single pdf** printout of your completed source code to Canvas. **Include in a comment block at the top of your code a summary of your experience with this project.**
 2. Ask to demo your lab to instructor. You can do this via writing your name on the whiteboard.
 - a. If you demo during lab in Week 7, you will earn a 10% bonus on this lab.
 - b. If you demo during lab in Week 8, you will be eligible for full credit.
- Demos are ONLY accepted during lab periods. If you are unable to demo by the end of lab in Week 7, you lose the 10% of the assignment attributed to the demo (per syllabus).
 - Demos must be ready a reasonable amount of time before the end of the lab period. If you write your name on the board at 9:45 and lab ends at 9:50, and there are five names in front of yours, you will be unlikely to complete your demo by the end of lab and hence lose the bonus or demo points.

4.1 GRADING CRITERIA

For full credit, your solution must:

- Use a timer/counter in an appropriate mode to measure incoming signal information.
- Implement an interrupt service routine to automatically sequence notes in a song.
- Add “play background music” options to existing menu structure, when option is selected, menu remains usable for other menu functions while music plays in background.
- Minor errors usually result in a deduction of ~ 3 points (three such errors results in ~ a letter grade reduction)
- Major errors, such as not achieving a requirement, usually result in a deduction of 5 to 10 points.