SWEN 563

Real Time & Embedded Systems

Project 1

Christopher Broderick cjb9317@rit.edu

Muhammad Adib Yahaya my3197@rit.edu

February 13, 2018

Overview

Using the STM32 board, we designed an embedded system that displayed the counts for one thousand rising edge pulses. The system had the ability to let users choose upper and lower limits of pulses by using the virtual terminal. The program can be ran multiple times with user-configured lower and upper limits.

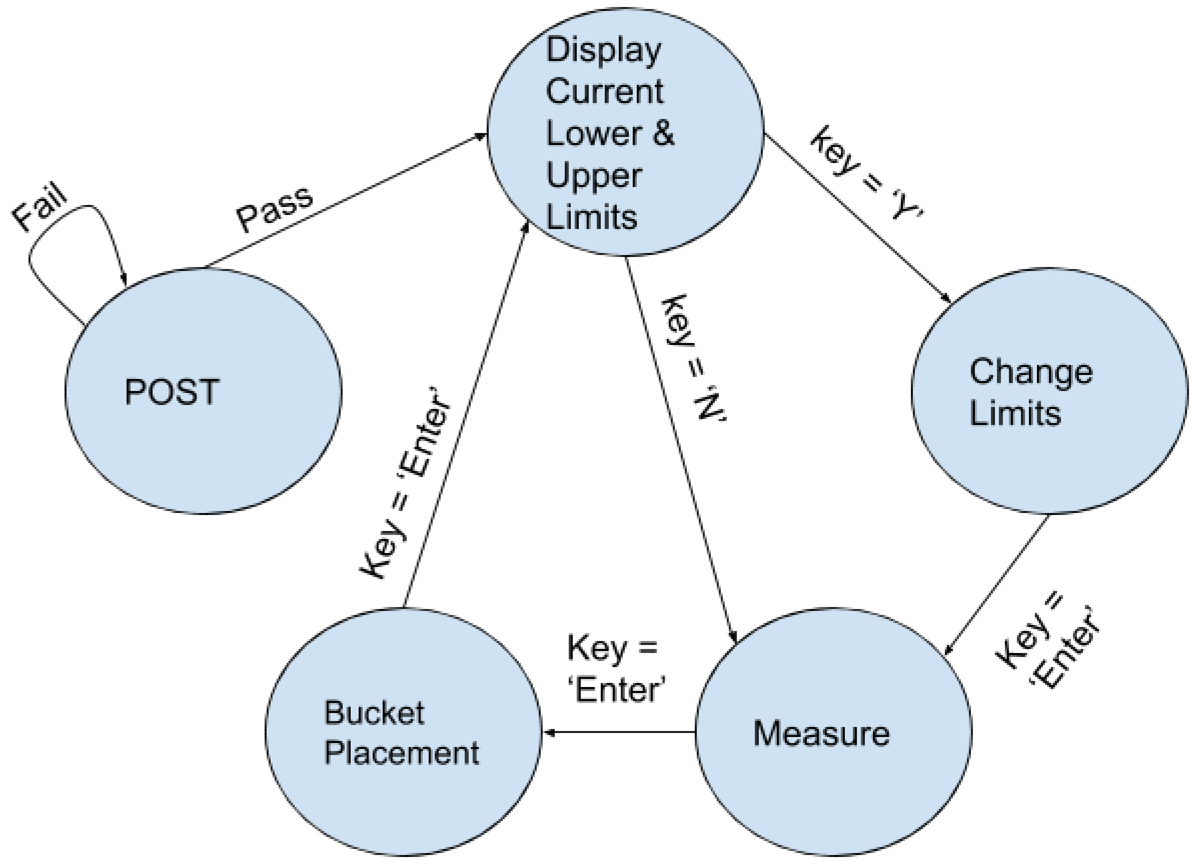
This was achieved by using a general-purpose timer and a GPIO pin. The timer increments a counter in every clock cycle to measure the time for a pulse from the GPIO pin to reach a rising edge.

Areas of Focus

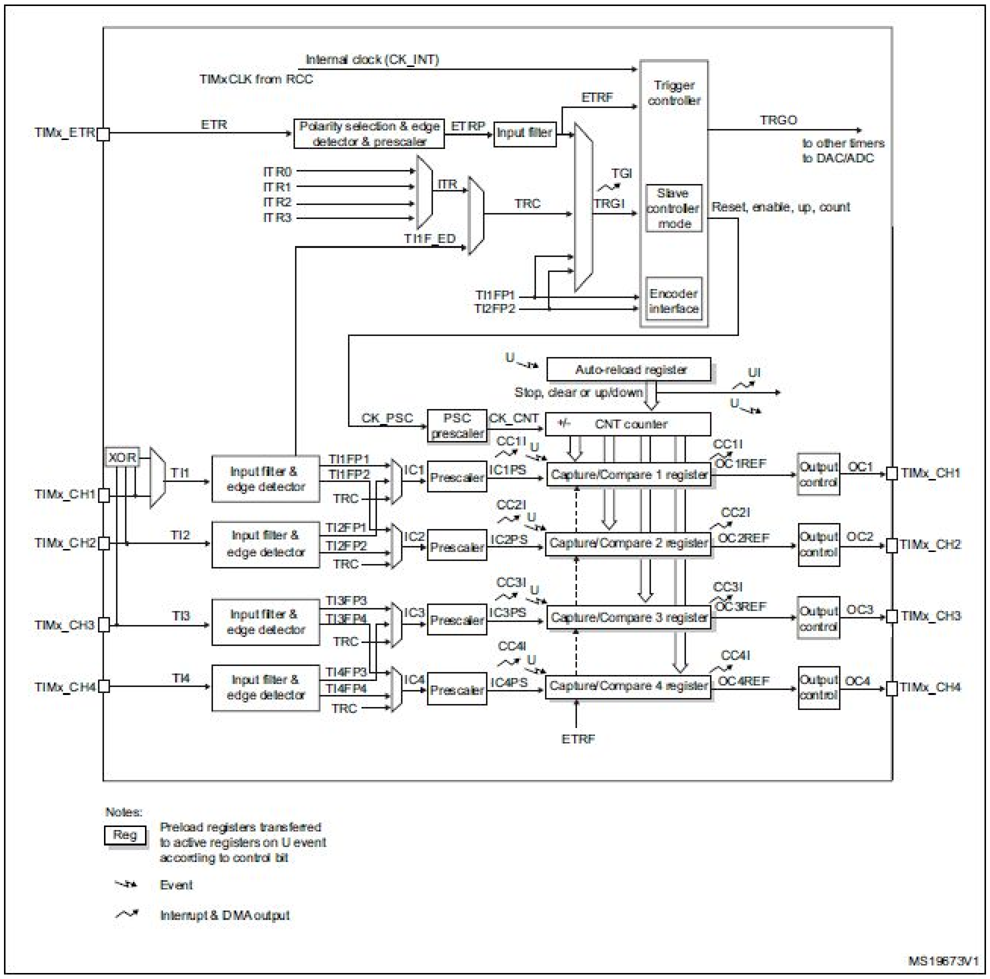
1. Determine which timer to use.
2. Timer 2 and GPIO pin A0 configurations.
3. Power On Self Test.
4. Display the current upper and lower limits to the terminal.
5. User has the option to change the default lower limit.
6. If user input is out of bounds (50 microsecond - 9950 microsecond) or is not a valid number, program should use a default value of 950 microsecond for the lower limit.
7. Program start after user presses “Enter”.
8. Display of buckets in ascending order with time and count.
9. Upon completion, the program can be re-run with the same or different user-configured limits.

Analysis / Design

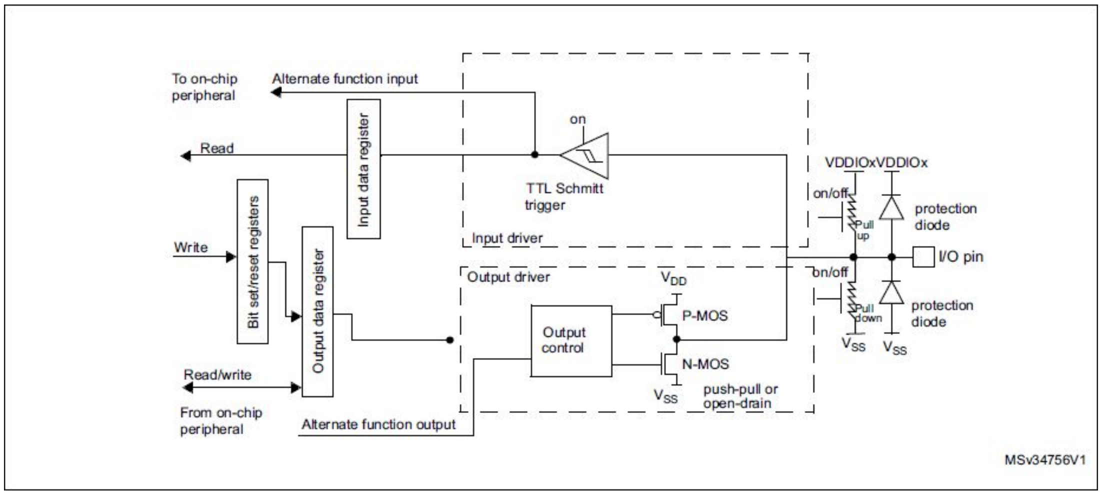
Flow Chart:



Timer:


Internal clock (CK_INT)
TIMxCLK from RCC
Trigger controller
ETRF
TIMx_ETR
ETR
Polarity selection & edge detector & prescaler
ETRP
Input filter
TRGO
to other timers to DAC/ADC
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TGI
TRGI
Slave controller mode
Reset, enable, up, count
TI1F_ED
TI1FP1
TI2FP2
Encoder interface
U
Auto-reload register
UI
Stop, clear or up/down
CK_PSC
PSC prescaler
CK_CNT
+/- CNT counter
XOR
TI1
Input filter & edge detector
TIMx_CH1
IC1
Prescaler
IC1PS
CC1I
Capture/Compare 1 register
OC1REF
Output control
OC1
TIMx_CH2
TI2
IC2
IC2PS
CC2I
Capture/Compare 2 register
OC2REF
OC2
TIMx_CH3
TI3
IC3
IC3PS
CC3I
Capture/Compare 3 register
OC3REF
OC3
TIMx_CH4
TI4
IC4
IC4PS
CC4I
Capture/Compare 4 register
OC4REF
OC4
ETRF
Notes:
Reg
Preload registers transferred to active registers on U event according to control bit
Event
Interrupt & DMA output
MS19673V1

GPIO:


To on-chip peripheral
Alternate function input
Read
Input data register
on
TTL Schmitt trigger
Input driver
VDDIOx VDDIOx
on/off
Pull up
protection diode
I/O pin
Write
Bit set/reset registers
Output data register
Output driver
VDD
P-MOS
Output control
N-MOS
VSS
push-pull or open-drain
Pull down
VSS VSS
Read/write
From on-chip peripheral
Alternate function output
MSv34756V1

Test Plan

When the program starts, it will run Power On Self Test (POST) to check if it is receiving pulses from GPIO pin A0. It if does not, the user is given the option to run the test again. If it fails to detect a pulse for the second time, the program will output an error and quit running.

After POST, the program allows the user to change the lower limit. If the entered limit is out of bound or not a valid number, the program defaults to a lower limit of 950 microseconds.

Then, the user can press Enter/Return to start measuring the time between pulses. The program completes 1000 measurement and displays only non-zero counts of the pulses. If the measurements are outside of the lower or upper limits, the display outputs nothing.

When a 1 kHz signal is supplied to the GPIO pin, pulses of 1000 microsecond period are expected. The CCR1 register should increment by 1000 for each pulses detected.

Project Results

POST was able to detect the status of input capture register. The default values of lower and upper limits were displayed before measurements are done.

The user can change the default values by entering ‘Y’ and the limit. The upper limit is always 100 microseconds higher that the lower limit.

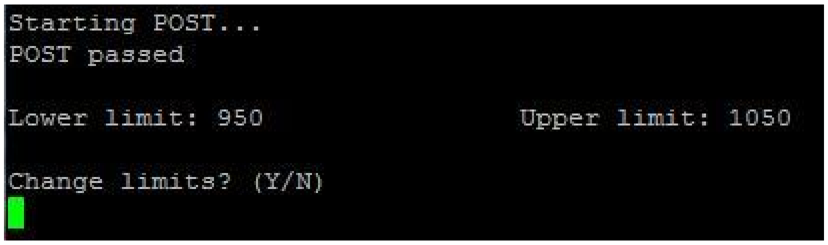
When ‘Enter’ is pressed, the program ran 1000 measurements. The terminal displayed a list of counts that were within the lower and upper limits. No zero counts were displayed. The measurements were not 100% accurate. Each 1 millisecond pulse was measured to be between 999 and 1006 microseconds.

After completing the measurements and displaying the result, the program can be ran again, using either the previous lower and upper limits, or different limits.

When the program was ran repeatedly, the range of measurements were similar. This means the timer and counter were working as expected.

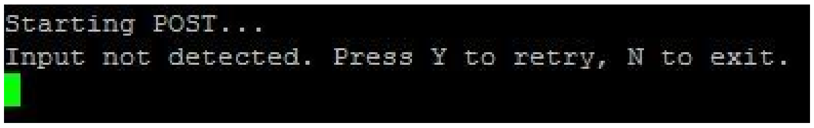
By using a lower period square wave at 500 Hz, the measurements were between 2003 and 2008. This range of values were within expectation.

POST passed. Program displays the limits.



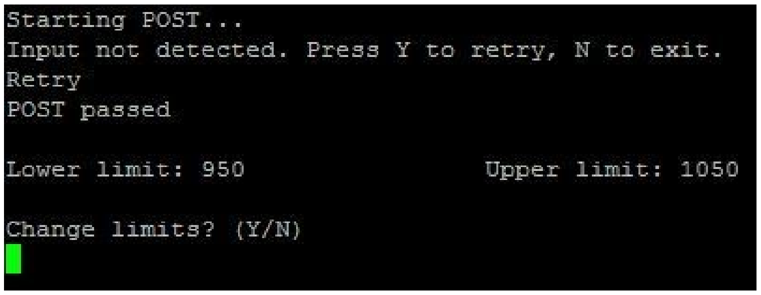
```
Starting POST...
POST passed

Lower limit: 950               Upper limit: 1050

Change limits? (Y/N)
```

POST failed. User can retry.



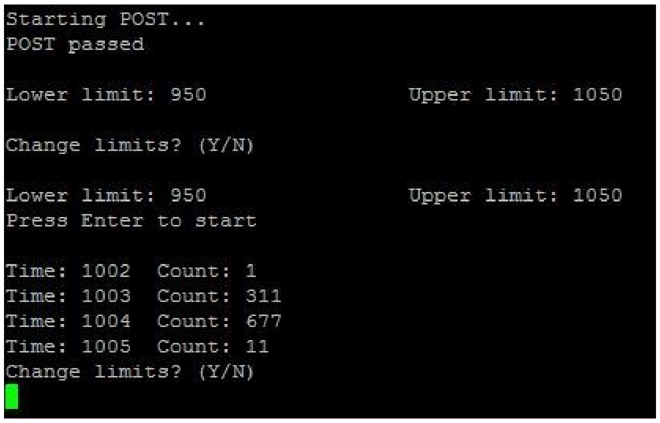
```
Starting POST...
Input not detected. Press Y to retry, N to exit.
```

POST initially failed. But user requested retry. Now POST has passed.



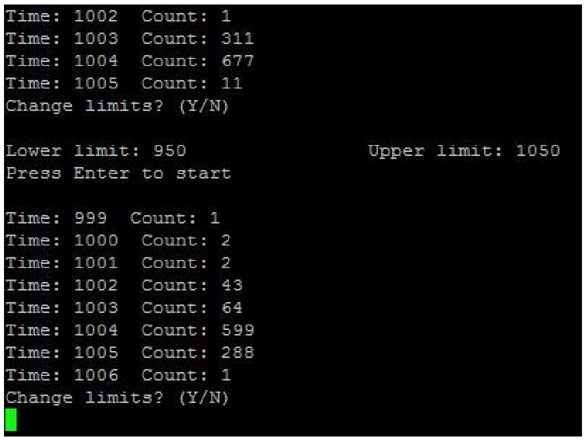
```
Starting POST...
Input not detected. Press Y to retry, N to exit.
Retry
POST passed

Lower limit: 950               Upper limit: 1050

Change limits? (Y/N)
```

Measuring 1000 1 millisecond pulses and showing the list of counts



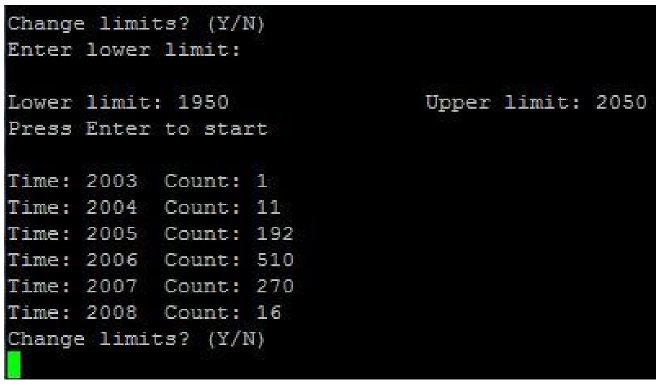
```
Starting POST...
POST passed

Lower limit: 950             Upper limit: 1050

Change limits? (Y/N)

Lower limit: 950             Upper limit: 1050
Press Enter to start

Time: 1002  Count: 1
Time: 1003  Count: 311
Time: 1004  Count: 677
Time: 1005  Count: 11
Change limits? (Y/N)
```

Re-run measurements



```
Time: 1002  Count: 1
Time: 1003  Count: 311
Time: 1004  Count: 677
Time: 1005  Count: 11
Change limits? (Y/N)

Lower limit: 950             Upper limit: 1050
Press Enter to start

Time: 999  Count: 1
Time: 1000  Count: 2
Time: 1001  Count: 2
Time: 1002  Count: 43
Time: 1003  Count: 64
Time: 1004  Count: 599
Time: 1005  Count: 288
Time: 1006  Count: 1
Change limits? (Y/N)
```

Changing lower limits to 1950 microseconds and the function generator to 500 Hz



```
Change limits? (Y/N)
Enter lower limit:

Lower limit: 1950            Upper limit: 2050
Press Enter to start

Time: 2003  Count: 1
Time: 2004  Count: 11
Time: 2005  Count: 192
Time: 2006  Count: 510
Time: 2007  Count: 270
Time: 2008  Count: 16
Change limits? (Y/N)
```

Lessons Learned

In this project, several fundamental concepts were introduced and we have learned how to work with the features and limitations that the microcontroller has.

The first one was using bitwise operators to make changes to bits in a register. This is particularly important because each bit in a register is tied to a function. Correct use of bitwise operations are crucial to ensure the program works like it is expected to be.

The use of ‘while’ loops to make a program to run indefinitely. This is common to all programs that are ran in embedded systems.

Timer and its features. We have learned to set up the timer and get the counter to increment when there is an input captured on a GPIO pin. Since the counter is directly related to the system clock, we could use its value to measure time.

There were some difficulties that we faced during the completion of this project.

Figuring out how to make USART read and write to work with integers, arrays, and pointers was one of them. Those functions only accept 8-bit wide pointer. So we have to use ‘sprintf’ function to convert integer types to strings.

Some registers require the other bits to be cleared when setting a bit. We could not get the GPIO pin to capture an input signal because the respective bits were defaulted to ‘11’. But it has to be ‘10’ to be in ‘alternate function’ mode.

Initially, we stored the all 1000 measurements in an array. But the board does not have enough memory to store the list of counts. That caused the program to stop running if there more than 100 measurements done.