

Project Report

Real Time & Embedded Systems

Title: Histogram of Pulse Inter-arrival Times

09.15.2014.

Suvam Bag

(MS-Computer Engineering)

Andrew Landman

(Senior -Software Engineering)

Contact:

sb5124@rit.edu

anl8094@rit.edu

Contents

1. Introduction

2. Assignments

3. MC9S12DT256

3.1- Timer

3.1.1 - Overview

3.1.2 -Features

3.1.3 – Modes of operation

3.1.4 - Block diagram

3.1.5 – Timer Registers

4. CodeWarrior and Putty

5. Analysis/Design

5.1- Hardware

5.2 – Software

6. Testing & Results

7. Lessons Learned

1. Introduction

This was the first project using the freescale embedded systems board using the HCS12 microcontroller. The primary objective of the project was to design and implement a stand alone 68HCS12 program, displaying a histogram of 1000 rising edge pulse inter-arrival times. The input signal was generated from a function generator with a 1000 Hz frequency giving an expected value of 1.0 millisecond. However this is where the other part of the program comes into the picture. The input signal always has some kind of distortion associated with it, hence the input frequency is never exactly constant. Hence to display the exact value, a range from 950-1050 micro seconds was set for the input values. The microcontroller was programmed in such a way so as to count the number of samples for each frequency within this range, and display the values starting from the least value obtained. This was an excellent way to start a freescale project, as the tool to measure the software jitter associated with real time operating systems.

2. Assignments

The project was a collaboration between the software engineering department and the computer engineering department. Understanding both the hardware and the software were equally important for the project. But given the edge the computer engineering student and the software engineering student holds over each other in the area of hardware and software respectively, tasks were divided keeping it collaborative at the same time, between the students from the two departments. However, given the software oriented nature of the project, both of our primary focus was to understand the software code and implement it in the necessary conditions.

3. MC9S12DT256

According to the ECT_16B8C Block User Guide V01.06 by Motorola Inc. revised release date on July 05,2004

The MC9S12DT256 microcontroller unit (MCU) is a 16-bit device composed of standard on-chip peripherals including a 16-bit central processing unit (HCS12 CPU), 256K bytes of Flash EEPROM, 12Kbytes of RAM, 4K bytes of EEPROM, two asynchronous serial communications interfaces (SCI), three serial peripheral interfaces (SPI), an 8-channel IC/OC enhanced capture timer, two 8-channel, 10-bit analog-to-digital converters (ADC), an 8-channel pulse-width modulator (PWM), a digital Byte Data Link Controller (BDLC), 29 discrete digital I/O channels (Port A, Port B, Port K and Port E), 20 discrete digital I/O lines with interrupt and wakeup capability, three CAN 2.0 A, B software compatible modules (MSCAN12), and an Inter-IC Bus. The MC9S12DT256 has full 16-bit data paths throughout. However, the external bus can operate in an 8-bit narrow mode so single 8-bit wide memory can be interfaced for lower cost systems. The inclusion of a PLL circuit allows power consumption and performance to be adjusted to suit operational requirements.

The detailed explanation of each feature of the MC9S12DT256 is not given in the report but only the relevant portions used primarily in the project.

3.1. Timer

According to the ECT_16B8C Block User Guide V01.06 by Motorola Inc. revised release date on July 05,2004

3.1.1. Overview

The HCS12 Enhanced Capture Timer module has the features of the HCS12 Standard Timer module enhanced by additional features in order to enlarge the field of applications, in particular for automotive ABS applications. This design specification describes the standard timer as well as the additional features. The basic timer consists of a 16-bit, software-programmable counter driven by a prescaler. This timer can be used for many purposes, including input waveform measurements while simultaneously generating an output waveform. Pulse widths can vary from microseconds to many seconds. A full access for the counter registers or the input capture/output compare registers should take place in one clock cycle. Accessing high byte and low byte separately for all of these registers may not yield the same result as accessing them in one word.

3.1.2. Features

- 16-Bit Buffer Register for four Input Capture (IC) channels.
- Four 8-Bit Pulse Accumulators with 8-bit buffer registers associated with the four buffered IC channels. Configurable also as two 16-Bit Pulse Accumulators.
- 16-Bit Modulus Down-Counter with 4-bit Prescaler.
- Four user selectable Delay Counters for input noise immunity increase.

3.1.3. Modes of Operation

STOP: Timer and modulus counter are off since clocks are stopped.

FREEZE: Timer and modulus counter keep on running, unless TSFRZ in TSCR(\$06) is set to one.

WAIT: Counters keep on running, unless TSWAI in TSCR (\$06) is set to 1.

NORMAL: Timer and modulus counter keep on running, unless TEN in TSCR(\$06) respectively.

MCEN in MCCTL (\$26) are cleared.

3.1.4. Block Diagram

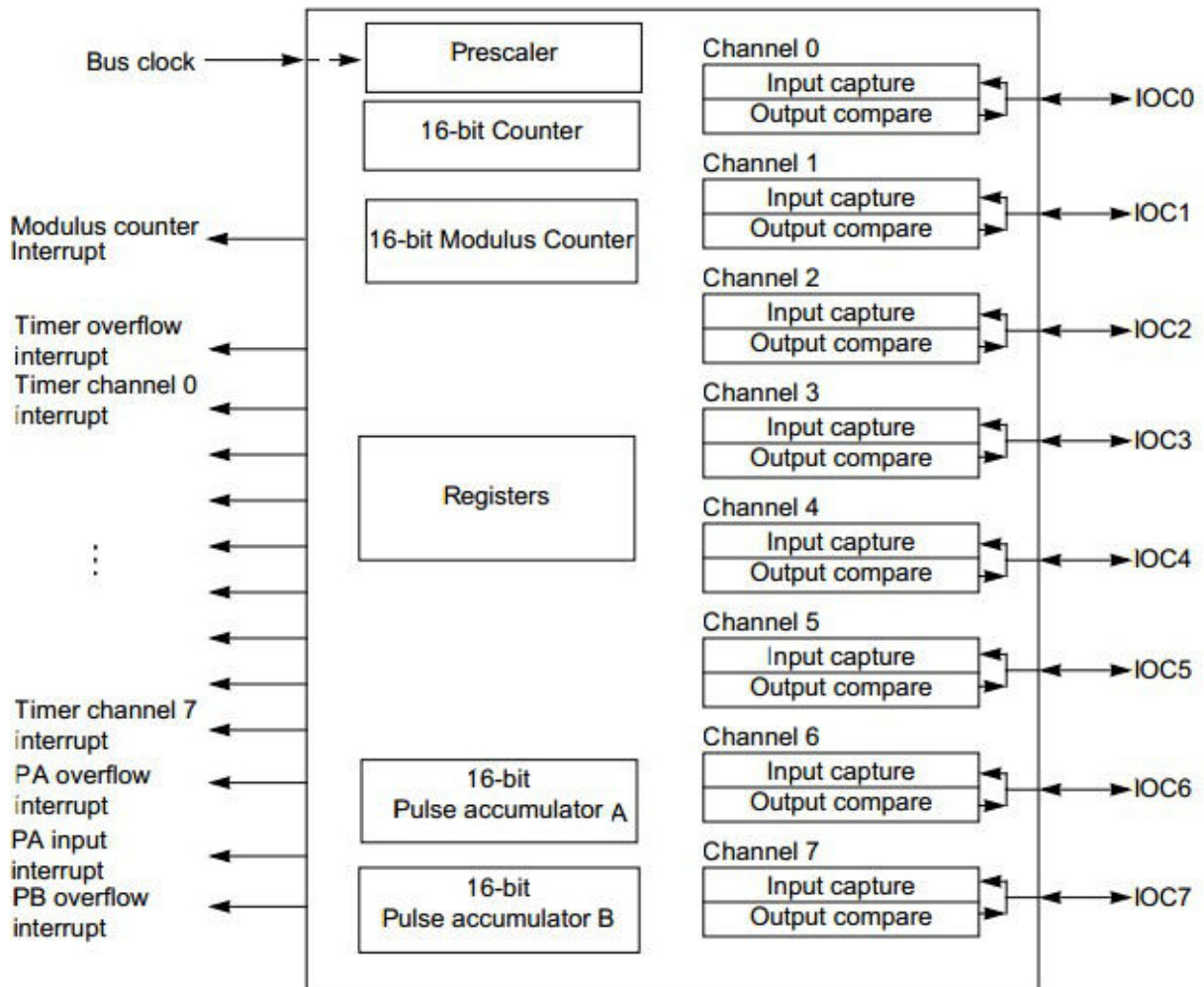


Figure: Timer Block Diagram

3.1.5. Timer Registers

TIOS — Timer Input Capture/Output Compare Select Register

Register offset: \$_00

	BIT7	6	5	4	3	2	1	BIT0
R	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
W	IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0
RESET:	0	0	0	0	0	0	0	0

Figure 3-1 Timer Input Capture/Output Compare Register (TIOS)

Read or write anytime.

IOS[7:0] — Input Capture or Output Compare Channel Configuration

0 = The corresponding channel acts as an input capture

1 = The corresponding channel acts as an output compare.

TSCR1 — Timer System Control Register 1

Register offset: \$_06

	BIT7	6	5	4	3	2	1	BIT0
R	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0
W	TEN	TSWAI	TSFRZ	TFFCA				
RESET:	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 3-6 Timer System Control Register 1 (TSCR1)

Read or write anytime.

TEN — Timer Enable

0 = Disables the main timer, including the counter. Can be used for reducing power consumption.

1 = Allows the timer to function normally.

TCTL4 — Timer Control Register 3/Timer Control Register 4

Register offset: \$_0B

	BIT7	6	5	4	3	2	1	BIT0
R	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
W								
RESET:	0	0	0	0	0	0	0	0

Figure 3-9 Timer Control Register 3/Timer Control Register 4 (TCTL3/TCTL4)

Read or write anytime.

EDGnB, EDGnA — Input Capture Edge Control

These eight pairs of control bits configure the input capture edge detector circuits.

Table 3-3 Edge Detector Circuit Configuration

EDGnB	EDGnA	Configuration
0	0	Capture disabled
0	1	Capture on rising edges only
1	0	Capture on falling edges only
1	1	Capture on any edge (rising or falling)

TSCR2 — Timer System Control Register 2

Register offset: \$_0D

	BIT7	6	5	4	3	2	1	BIT0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

PR2,PR1,PR0- Timer Prescalar Select

TFLG1 — Main Timer Interrupt Flag 1

Register offset: **\$_0E**

	BIT7	6	5	4	3	2	1	BIT0
R	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
W								
RESET:	0	0	0	0	0	0	0	0

TIE — Timer Interrupt Enable Register

Register offset: **\$_0C**

	BIT7	6	5	4	3	2	1	BIT0
R	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
W								
RESET:	0	0	0	0	0	0	0	0

4. CodeWarrior and Putty

The software development tools used for this project was CodeWarrior which accepts 'C' as its programming language. It supports register based calling hence is easy to use for this kind of implementation. The classes, methods, member functions, data members and other structures can be viewed in an organized manner helping the user to understand them if required.

Putty is the serial command window via which the user can view the data sent via serial communication, in this case from the freescale controller to the PC.

5. Analysis / Design

5.1. Hardware

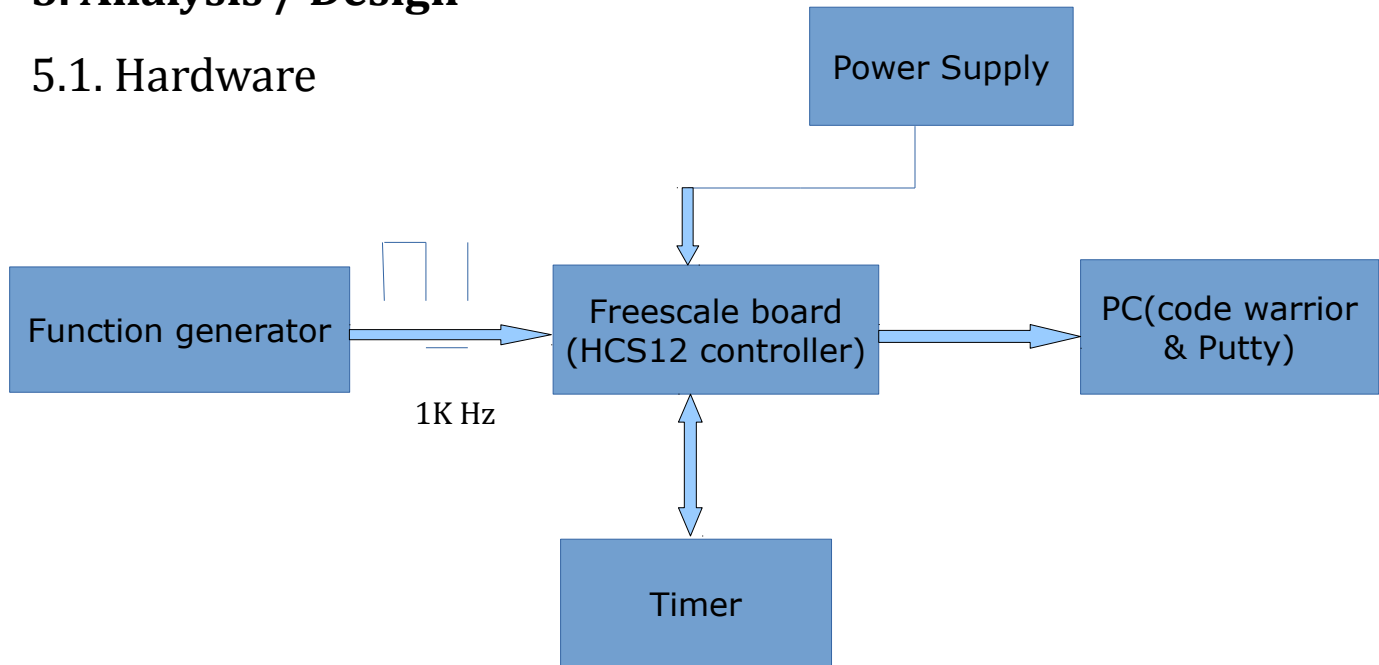


Fig: Block Diagram

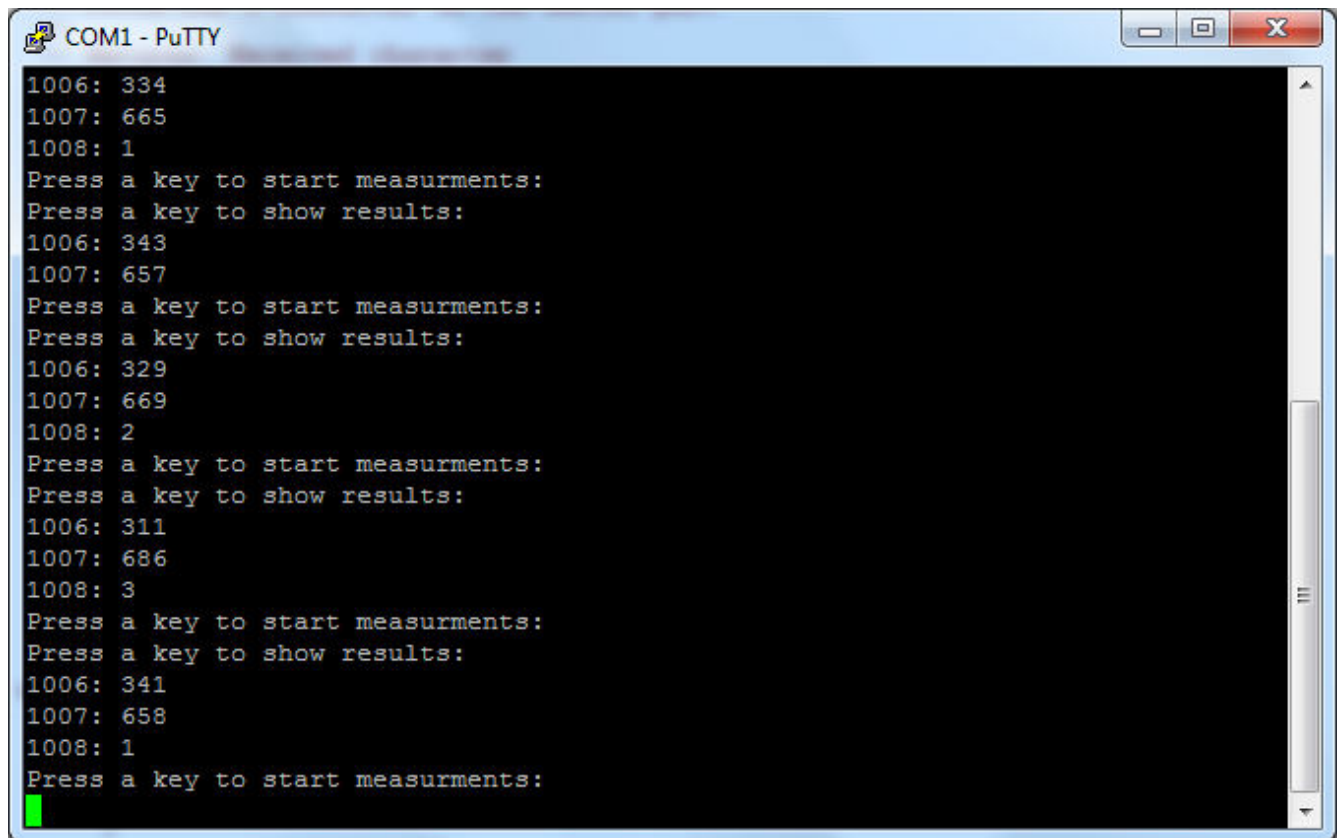
Assuming the code has been uploaded to the HCS12 microcontroller, the block diagram is pretty simple. The function generator sends a pulse of 1K Hz approximately to the freescape board. The timer starts counting from the first rising edge of the pulse and the interrupt routine is called (it is programmed that way) and disabled in the following step. It's once again enabled at the next rising edge when the timer detects the rising edge of the pulse. In this way the timer can count the time between two interrupt subroutines and gives us the time period of the signal. This is the basic logic of the program and the hardware. But there was a slight complication in the software. It has been explained after the flowchart and the code itself.

5.2 Software

We started the project with the hello world example as a template. From there we realized we needed to do input capture to read data from the timer as opposed to doing output compare that the hello world example was doing. In `InitializeTimer()` we changed `TIOS_IOS1` to 0 to enable input capture on channel 1. We then set `TCTL4_EDG1A` to 1 and `TCTL4_EDG1B` to 0 as suggested in the manual to enable capture on rising edge only. Additionally, we set `TIE_C1I` to 1 to enable the input capture interrupt on channel 1. This way our interrupt service routine would get called on each rising edge. Once this was done the next step was to implement the interrupt service routine. We changed this function to read from `TC1` and stored the result. We configured the interrupt service routine to store both the

previous and current value of the timer and then took the difference to get the time between two rising edges. We found the index of the difference in the histogram we created and added one to the count. This way we could see how many times it took 950 microseconds to go between two rising edges, how many times it took 951 microseconds to go between two rising edges, etc. Lastly, we set up the main function of to prompt the user to start collecting data, wait for 1000 interrupts, and prompt the user to display the data. This was put in a loop so that the user could collect data continuously to get multiple results.

6. Testing and Results



```
COM1 - PuTTY
1006: 334
1007: 665
1008: 1
Press a key to start measurments:
Press a key to show results:
1006: 343
1007: 657
Press a key to start measurments:
Press a key to show results:
1006: 329
1007: 669
1008: 2
Press a key to start measurments:
Press a key to show results:
1006: 311
1007: 686
1008: 3
Press a key to start measurments:
Press a key to show results:
1006: 341
1007: 658
1008: 1
Press a key to start measurments:
█
```

The above screenshot shows several runs of the program. The results show that the time between rising edges is between 1006 to 1008 microseconds. The most common value is 1007 microseconds. These results are in line with the expected value of it being an average of 1000 microseconds between two rising edges. The only difference is that our signal generator was tuned slightly differently causing it to take a little longer between rising edges.

7. Lessons Learned

Most of the lessons learned in doing this lab involved learning how the hardware worked. We spent quite some time going through the manual figuring out which registers we need to access and which ones we needed to modify. We also learned how to work with interrupts. This included how to enable/disable them and set up interrupts to get called on rising edges. Finally, we learned how to setup the hardware properly. Upon starting the lab we had plugged the serial cable into the port on the main board as opposed to the port on the daughter card. It took us hours to find the cause of the issue, but thankfully we finally did and were able to make progress on the rest of the lab.