**Lab # 10 Recording a Periodic Signal using Input Capture Feature of Timers/Counters**

# Objectives:

### To introduce input capture feature of the Timers/Counters in microcontrollers.

* To emphasize preference of interrupt based designs over polling mechanisms.

**Tools:**

**Software Tools:**

### Microchip Studio

* Proteus ISIS
* Arduino IDE

**Hardware Tools:**

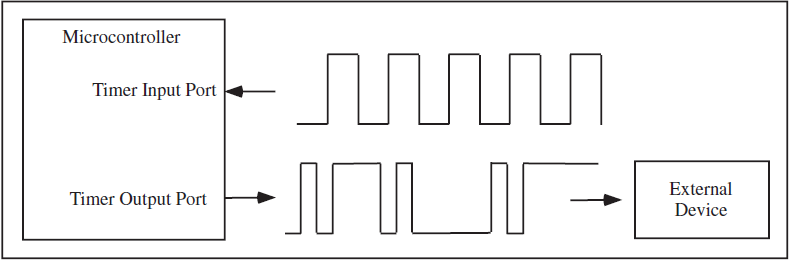
|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Quantity** |
| Arduino Nano | - | 1 |
| Resistor | 4.7k Ω | 1 |
| Push button | - | 1 |

*Table 10.1: List of Components*

**Pre-Lab:**

**Input Capture:**

In many applications, we are interested in measuring the elapsed time or the frequency of an external event using a microcontroller. Using the hardware and functional units discussed in the previous sections, we now present a procedure to accomplish the task of computing the frequency of an incoming periodic signal. Figure 10.1 shows an incoming periodic signal to our microcontroller.



*Figure 10.1 Use of the timer I/O systems of a microcontroller. The signal on top is fed into a timer*

The first necessary step for the current task is to turn on the timer system. To reduce power consumption, a microcontroller usually does not turn on all of its functional systems after reset until they are needed. In addition to a separate timer module, many microcontroller manufacturers allow a programmer to choose the rate of a separate timer clock that governs the overall functions of a timer module.

Once the timer is turned on and the clock rate is selected, a programmer must configure the physical port to which the incoming signal arrives. This step is done using a special input timer port configuration register. The next step is to program the input event to capture. In our current example, we should capture two consecutive rising edges or falling edges of the incoming signal. Again, the programming portion is done by storing an appropriate setup value to a special register.

Now that the input timer system is configured appropriately, you now have two options to accomplish the task. The first one is the use of a polling technique; the microcontroller continuously polls a flag, which holds a logic high signal when a programmed event occurs on the physical pin. Once the microcontroller detects the flag, it needs to clear the flag and record the time when the flag was set using another special register that captures the time of the associated free-running counter value. The program needs to continue to wait for the next flag, which indicates the end of one period of the incoming signal. A programmer then needs to record the newly acquired captured time represented in the form of a free-running counter value again. The period of the signal can now be computed by computing the time difference between the two captured event times, and based on the clock speed of the microcontroller, the programmer can compute the actual time changes and consequently the frequency of the signal.

In many cases, a microcontroller cannot afford the time to poll for one event. Such situation introduces the second method: interrupt systems. Most microcontroller manufacturers have developed built-in interrupt systems with their timer input modules. Instead of continuously polling for a flag, a microcontroller performs other tasks and relies on its interrupt system to detect the programmed event. The task of computing the period and the frequency is the same as the first method, except that the microcontroller will not be tied down constantly checking the flag, increasing the efficient use of the microcontroller resources. To use interrupt systems, of course, we must pay the price by appropriately configuring the interrupt systems to be triggered when a desired event is detected. Typically, additional registers must be configured, and a special program called an Interrupt Service Routine (ISR) must be written.

Suppose that for an input capture scenario, the two captured times for the two rising edges are

$1000 and $5000, respectively. Note that these values are not absolute times but the representations of times reflected as the values of the free-running counter. The period of the signal is $4000, or 16384 in a decimal form. If we assume that the timer clock runs at 10 MHz, the period of the signal is 1.6384 ms, and the corresponding frequency of the signal is approximately 610.35 Hz.

**In Lab Tasks:**

**Task 1:**

The expected input signal is a periodic pulse train. The task is to find the frequency/time period and duty cycle of this signal. The frequency of this signal is expected to be less than 4 KHz. The given code configures Atmega328P interrupt subsystem to work with external interrupts and input capture of Timer1. It also performs the necessary calculations to find the frequency and duty cycle of the input signal and displays it on a terminal. You will have to understand the code and then configure Timer1 for input capture.

**Code:**

// This program configures the 16-bit Timer1 of Atmega328p for Input Capture.

// The expected input signal is a periodic pulse train. The task is to find

// the frequency/time period and duty cycle of this signal. The frequency of

// this signal is expected to be less than 4 KHz.

//

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*/

#include <inttypes.h>

#include <stdlib.h>

#include <avr/io.h>

#include <avr\interrupt.h> // Add the necessary ones

#ifndef F\_CPU

#define F\_CPU 16000000UL

#endif

#include <util/delay.h>

#include <string.h>

#include <math.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Definitions for UART \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define SERIAL\_DEBUG

#ifdef SERIAL\_DEBUG

#include "debug\_prints.h"

#include "debug\_prints.c"

#endif

#define BAUD0 9600 // Baud Rate for UART

#define MYUBRR (F\_CPU/8/BAUD0-1) // U2X = 1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char icp\_low = 0;

unsigned char icp\_high = 0;

unsigned int input\_capt\_counter=0;

unsigned int rising1 = 0;

unsigned int rising2 = 0;

unsigned int falling1 = 0;

unsigned char capture\_complete=0;

float sig\_freq = 0;

float sig\_dc = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define GLB\_INTR\_EN asm("sei");

#define GLB\_INTR\_DIS asm("cli");

#define TRUE 1

#define FALSE 0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function Prototypes here \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void timer\_init(void); // Function to initialize Timer1 for Input Capture

void display\_counter\_values(void);

void display\_signal\_parameters(void);

void calculate\_signal\_param(void);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Main program

//

int main(void)

{

DDRB &= (1<<PB0); // Set PB0 (ICP1) for input

PORTB|= (1<<PB0); // Activate internal pull-up

UART0\_init(MYUBRR);

printSerialInt(MYUBRR);

printSerialStrln("");

printSerialStr("F\_CPU = ");

printSerialInt((int)(F\_CPU/1000000));

printSerialStrln("MHz");

printSerialStrln("Lab 10: Input Capture");

timer\_init();

//\*\*\*\*\*\*\* Write Code For This \*\*\*\*\*\*\*\*//

//Set interrupt sense control bits to falling edge for INT0

//Enable INT0 locally

GLB\_INTR\_EN; // Enable interrupts globally

int i=0;

while(1)

{

if(capture\_complete == TRUE)

{

display\_signal\_parameters(); // display frequency and duty cycle

capture\_complete = FALSE;

for(i=0; i<8; i++) // wait for 2 seconds

\_delay\_ms(250);

printSerialStrln("Press the button ..");

}

}

}

/\* This function will setup Timer1 for Input Capture Mode.

if Fclk = 16MHz, and prescaler = 8, then 1 count = 0.5us. 2000

counts will fit a Time period of 1ms (f = 1 KHz). 500 Counts

will fit a Time period of 0.25 ms (f = 4 KHz).

\*/

void timer\_init(void) // Write code for this function

{

ACSR &= ~(1<<ACIC); // Disconnect the Analog Comparator output from the Input Capture Unit

// Normal Mode, OC1A and OC1B disconnected.

// Initially capture rising edge. Prescaler = 8

TCNT1L = 0;

TCNT1H = 0;

// Enable interrupt of Timer 1 input capture

}

/\* Interrupt Service Routine for INT0. When the user presses

the button, the ISR clears TCNT1, capture\_complete, rising1,

rising2 and falling1 global variables. Then it turns on the

interrupts for Timer1 Input Capture. \*/

ISR(INT0\_vect)

{

//printSerialStrln("Processing External Interrupt 0: ");

rising1 = 0;

rising2 = 0;

falling1 = 0;

capture\_complete = FALSE;

input\_capt\_counter = 0;

sig\_freq = 0;

sig\_dc = 0;

TCCR1B |= (1<<ICES1); // for rising edge on ICP1

TCNT1 = 0; // clear the free running counter of Timer 1

TIMSK1 |= (1<<ICIE1); // Enable interrupt of Timer 1 input capture

}

/\* ISR for Input Capture of Timer1. You need to write

code and complete this function\*/

ISR(TIMER1\_CAPT\_vect)

{

//printSerialStrln("Processing Timer Interrupt: ");

icp\_low = ICR1L;

icp\_high = ICR1H;

input\_capt\_counter ++;

//printSerialInt((int)input\_capt\_counter);

//printSerialStrln("");

if(input\_capt\_counter == 2)

{

// Record the counter value on first Rising Edge

}

if(input\_capt\_counter == 3)

{

// Record the counter value on first Rising Edge

// Chnage the polarity of sensing

// Clear ICF flag as prescribed in the Datasheet Page 157 Section 20.9.3

}

if(input\_capt\_counter == 4)

{

// Record the counter value on first Rising Edge

// disable further interrupts of Timer 1 input capture

this point

capture\_complete = TRUE; // capture is complete at

calculate\_signal\_param(); // calculate the Frequency and Duty Cycle of the Input Signal

}

}

/\*\* This function displays the values of the

captured edges 'rising1', 'rising2', and

'falling1'.

\*/

void display\_counter\_values()

{

printSerialStr("Rising 1: ");

printSerialLong((long int) rising1);

printSerialStrln("");

printSerialStr("Rising 2: ");

printSerialLong((long int) rising2);

printSerialStrln("");

printSerialStr("Falling 1: ");

printSerialLong((long int) falling1);

printSerialStrln("");

}

/\*\* Function to display the Frequency and Duty Cycle

of the Captured signal.\*/

void display\_signal\_parameters(void)

{

printSerialStr("Frequency = ");

printSerialFloat(sig\_freq);

printSerialStrln(" Hz");

printSerialStr("Duty Cycle = ");

printSerialFloat(sig\_dc);

printSerialStrln(" %");

}

/\*\* This function is called after all the required edges

are captured and saved in global variables 'rising1',

'rising2', and 'falling1'. It calculates the input signal

frequency and its duty cycle and saves them in corresponding

variables.\*/

void calculate\_signal\_param(void)

{

printSerialStrln("Calculating Signal Parameters ... ");

if((rising2-rising1) != 0) // if denominator is non-zero

{

sig\_freq = F\_CPU/(8\*(float)(rising2-rising1)); //Here Prescalar = 8 so Ftimer == F\_CPU/8

sig\_dc = 100.0 \* ((float)(falling1 - rising2))/((float)(rising2-rising1)); // ON-Time/ Total Time \* 100 %

}

else

{

sig\_freq = 0;

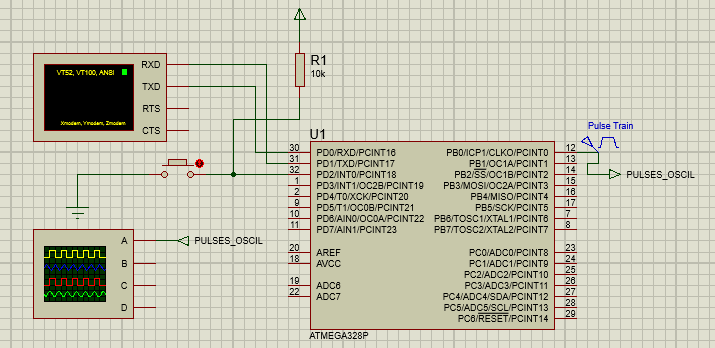
sig\_dc = 0;

}

display\_counter\_values();

display\_signal\_parameters();

}

**Simulation:**

**Hardware Design:**

Implement In lab task on hardware. Connect a function generator to supply the input signal, and a push button for interrupts. Use serial monitor of Arduino IDE to display the values.

**Post lab Task:**

Find out how input capture feature of microcontrollers can be used to record a signal from a TV remote control. Submit a report of your findings.

**Critical Analysis / Conclusion**

(By Student about Learning from the Lab)

|  |  |  |
| --- | --- | --- |
| **Lab Assessment** | | |
| **Pre Lab** | **/ 1** | **/10** |
| **In Lab** | **/5** |
| **Post Lab** | **/4** |
| **Instructor Signature and Comments** | | |