

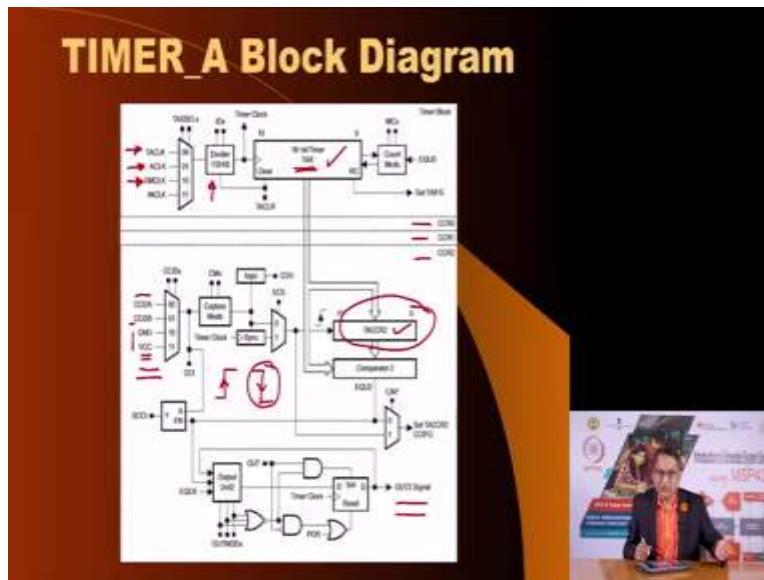
Introduction to Embedded System Design
Professor Dhananjay V. Gadre
Electronics and Communication Engineering
Netaji Subhas University of Technology
Badri Subudhi
Electrical Engineering Department
Indian Institute of Technology, Jammu
Lecture 34
MSP430 Timer in Capture Mode

Hello and welcome to a new session for this ongoing online course on Introduction to Embedded System Design. I am your instructor Dhananjay Gadre. We are going to continue our discussions on using the timer feature of the MSP430 Microcontroller in the new mode today. In the past we have seen how to enable the timer, how to use it for measuring time for some duration.

And then in subsequent lecture we saw how we could use the compare function of the timer to compare the value in the timer register with some value stored in the compare registers R1 and R2 and using that feature we were able to generate pulse width modulation signals.

Now what we want to do is, we want to use a timer for capturing external events or internal events and measure store the value of the timers, so this is called MSP430 timer in a capture mode.

(Refer Slide Time: 01:29)



As usual this is the block diagram of the timer and let us go through it from top to bottom. Here is our 16 bit timer register. The source of the timer can be an internal clock ACLK or it can be SM clock and now what I want to introduce you to you is an external pin TA clock.

That signal can be derived from a pin of the MSP430 of the microcontroller which means an external signal would be clocking the timer, would be incrementing the timer and this would effectively put the timer in a counter mode because if we do not know the frequency of that incoming signal all we can say is that we have counted X number of pulses from that signal.

We can also divide once we select what source we want to have we can divide it with the prescaler here and then it will be used to feed the clock for the timer. Coming at the bottom we have 3 compare registers, compare capture registers CCR0, CCR1 and CCR2. CCR0 is very special as we have seen in a compare lecture, in a previous session that you store a value in the capture compare register 0 and this is used to decide at what time the timer will reset.

But now what we want to do is, we want to see the capture mode. Now capture mode means that we want to use an external event or an internal event to capture the values of this timer. These values are coming here and they can be captured in this register, they can be captured in R0 capture compare register 0 as well as 1 but this is repeated here so this is what is being shown.

And what, what event would capture it? You could have a pin called CCI to A or to B for the capture compare register 2 similarly you would have pins for CCR1 and CCR0. And you also

have this, now why do you have a ground and VCC? So that you can generate a signal growing from high to low which you can select the multiplexer, you can set the multiplexer to ground selection.

And then when you will change it to 1 it would generate a high to low signal this could be used to capture into this register the value from the timer. Similarly, if you want high to low you set the value this multiplexer to VCC and then you change it to 0 to ground and this will generate a high to low transition which could be used to capture the value from the timer in to the capture compare register.

And then you can do various things with it you could read the values and so on and so forth. So, in the previous lecture in the compare mode we use this and use that information of comparison and outputted the signal as a PWM waveform. Now, in this lecture what we are going to do is we are going to see input pins which could be used to create events for the timer.

(Refer Slide Time: 04:49)



As usual these are our timer registers you have the timer control register, you have the timer register as it is which actually counts pulses or signals clock signals, you have capture compare control register 0, you have capture compare register, capture compare control 1, capture compare 1, capture compare control 2 and capture compare register 2 as well as the interrupt vector register.

(Refer Slide Time: 05:21)


TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
TA0	TA1	TA2	TA3	TA4	TA5	TA6	TA7
7	6	5	4	3	2	1	0
ICx		MCx		Unused	TACLx	TAE	TAPG
TA0	TA1	TA2	TA3	TA4	TA5	TA6	TA7



TACTL Continued

TASSELx	Bits 9-8	Timer_A clock source select
	00	TACLK ←
	01	ACLK
	10	SMCLK
	11	INCLK (INCLK is device-specific and is often assigned to the inverted TACLK; see the device-specific data sheet)
ICx	Bits 7:4	Input divider. These bits select the divider for the input clock.
	00	/1
	01	/2
	10	/4
	11	/8
MCx	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use consumes power.
	00	Stop mode: the timer is halted.
	01	Up mode: the timer counts up to TACCR0.
	10	Continuous mode: the timer counts up to 0xFFFF.
	11	Up/down mode: the timer counts up to TACCR0 then down to 0x0000.
Unused	Bit 3	Unused
TACLx	Bit 2	Timer_A clear. Setting this bit resets TAE, the clock divider, and the count direction. The TACLx bit is automatically reset and is always read as zero.
	Bit 1	Timer_A interrupt enable. This bit enables the TAPG interrupt request.
TAPG	Bit 0	Timer_A interrupt flag
	0	No interrupt pending
	1	Interrupt pending



Now, in the control register these are the bits lets go through this bits relevant to the capture part. As I mentioned we would like to select the source of the timer instead of being the internal clock signals we would like an external pin to provide the signal and in this case we will use these bits TASSELx and select this combination TACLK this is, this would be a map to a pin we will see which one.


We could if you like you could divide this signal coming there but we may not want to do that and then you want to decide whether you want to operate the timer in up mode, continuous mode

or up down mode and then whether you want to have the interrupt associated with the timer and the flag which tells you whether an interrupt is pending or not.

(Refer Slide Time: 06:14)


TACCTLx, Capture/Compare Control Register

15	14	13	12	11	10	9	8
CMx		CC0x		SCS	SCCI	Unused	CAP
rw(0)	rw(0)	rw(0)	rw(0)	rw(0)	r	rw(0)	rw(0)
7	6	5	4	3	2	1	0
OUTMODx		CCE		CCI	OUT	COV	CCIFG
rw(0)	rw(0)	rw(0)	rw(0)	r	rw(0)	rw(0)	rw(0)



TACCTLx Continued

CMx	Bit 15-14	Capture mode	
		00	No capture
		01	Capture on rising edge
		10	Capture on falling edge
CC0x	Bit 13-12	Capture/compare input select. These bits select the TACCTLx input signal. See the device-specific data sheet for specific signal connections.	
		00	CC0A
		01	CC0D
		10	GND
SCS	Bit 11	Synchronize capture scheme. This bit is used to synchronize the capture input signal with the timer clock.	
		0	Asynchronous capture
		1	Synchronous capture
SCCI	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the ED0x signal and can be read via this bit	
Unused	Bit 9	Unused. Read only. Always read as 0.	
CAP	Bit 8	Capture mode	
		0	Compare mode
		1	Capture mode



Here is the second part of the capture control register and let's see the bits. Here we now we want to say, do you want to capture mode or not? Now, in the previous one we said we want to do compare mode so the CAP bit was set to 0. In this lecture we are going to see how we will set this CAP bit to 1 to indicate to the timer that we want to capture.

Now the question is, once you decide that you want to capture, these bits will decide on what event are you going to capture are you going to capture on a rising edge or falling edge or do you want to capture on rising as well as falling this option will allow you to do that. And then the source of the signal that capture signal is it going to be a pin CC1A or 1B or a transition of these two signals. And do you want to synchronize this capture within internal clock you can select using this.

(Refer Slide Time: 07:16)

TACCTLx Continued

OUTMODx	Bits 7:5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCRx, because EQUx = EQUx.
	000	OUT bit value
	001	Set
	010	Toggle/reset
	011	Set/reset
	100	Toggle
	101	Reset
	110	Toggle/reset
	111	Reset
CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCFG flag.
	0	Interrupt disabled
	1	Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output.
	0	Output low
	1	Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
	0	No capture overflow occurred
	1	Capture overflow occurred
CCFG	Bit 0	Capture/compare interrupt flag.
	0	No interrupt pending
	1	Interrupt pending

This is the timer control register we saw this in the previous lecture these are the output bits and we if you recall for PWM we selected this combination. But now we want to actually go here that do you want to have the capture control interrupt enabled? Do you want to have find out the value what is the value of capture input? And in this case whether there is a capture overflow and interrupt flag associated with the capture signal.

(Refer Slide Time: 07:55)


TAIV, Timer_A Interrupt Vector Register

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
0	0	0	0	TAIVx			0
15	14	13	12	11(0)	10(0)	9(0)	8

TAIVx Bits 15:0 Timer_A interrupt vector value

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	--	--
02h	Capture/compare 1	TACCR1 COFG	Highest
04h	Capture/compare 2 ¹	TACCR2 COFG	--
08h	Reserved	--	--
0Ch	Reserved	--	--
0Eh	Timer overflow	TAFG	--
10h	Reserved	--	--
12h	Reserved	--	Lowest

1 Not implemented in MSP430C2xx devices




Now associated with the various you would have lot of interrupt sources, you have capture compare register 1, register 2 and a timer overflow and this would be available for both the timers.

(Refer Slide Time: 08:12)

Timer Capture Mode

- The capture mode of timer is selected when CAP = 1 in TACCTLx register.
- CCISx bits decide the capture input. The capture inputs CC1xA and CC1xB are connected to external pins or internal signals.
- When an event occurs at input signal, the value in TAR register is stored in TACCRx register.
- The CMx bits select the capture edge of the input signal as rising, falling, or both. A capture occurs on the selected edge of the input signal.
- If a capture occurs:
 - The timer value is copied into the TACCRx register ←
 - The interrupt flag CCIFG is set in TACCTLx register ←




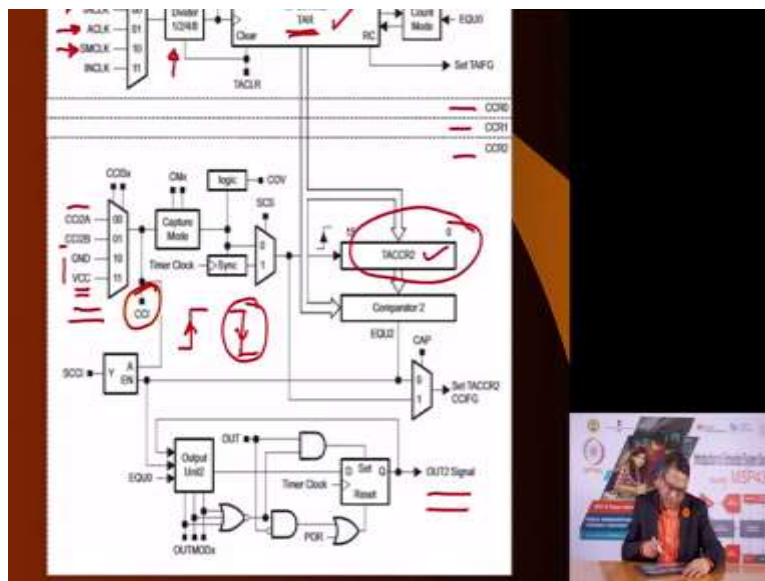
The timer capture select mode you have to select that we want to do a capture so CAP is equal to 1. The CCIS bits will decide the capture inputs and the capture inputs this or this will be selected map to the external pins. When an event occurs at the input signal the value in the timer register will be stored in the selected capture compare register 1 or 2 or 0.

The CMx bits will select the capture edge of the input signal do you want it on rising or falling and so on and if a capture occurs then the timer value is copied into this register and the interrupt flag CCIFG set in this control register so that an interrupt is generated provided the global interrupts have been enabled.

(Refer Slide Time: 09:09)

Timer Capture Mode

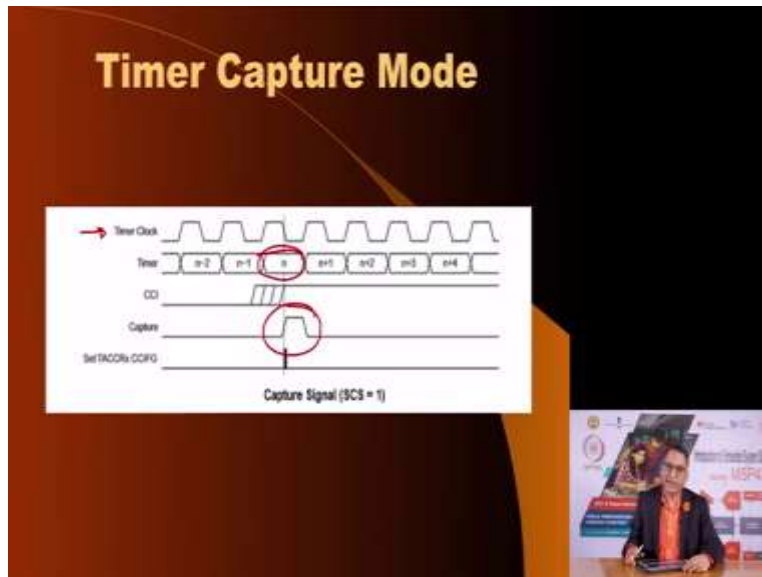
- State of selected input can be read in the CCI bit of TACCTLn.
- This input is asynchronous and can cause hardware "race condition".
- The hardware of capture mode includes a synchronizer which can be enabled by setting the SCS bit.
- Capture events are then synchronised with the next falling edge of the timer clock that follows the trigger, midway between increments of TAR.
- SCS bit should normally be set for safety

In the timer capture mode the state of the selected input can be read in the CCI bit and to show you where that bit is let me go back to the block diagram here. This is the CCI bit here, this one. Now this input can be synchronous, synchronized to the input clock or it can be asynchronous

you can choose what you want. Capture events are then synchronized with the next following edge of the timer clock that follows the trigger, midway between increments of the timer register.

(Refer Slide Time: 09:55)

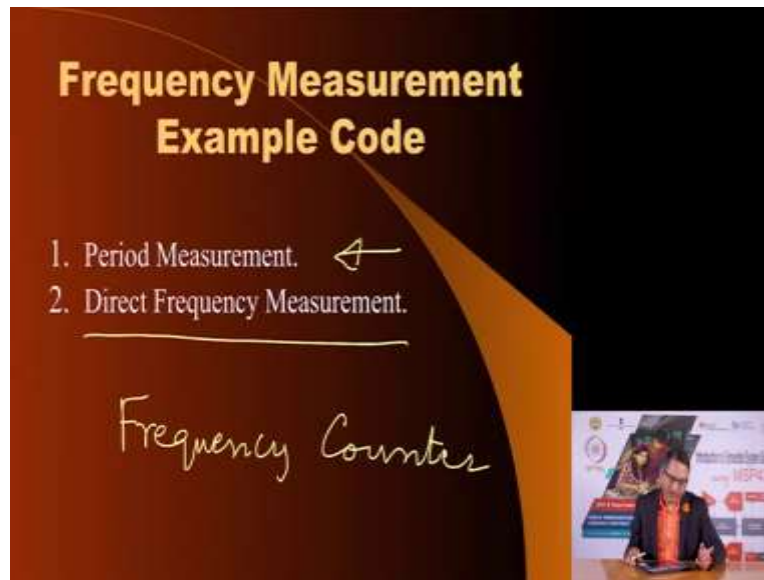


Now this is what happens your timer clock whatever be the source you are going to capture the value into the capture register whenever the appropriate condition arrives it will capture the value here of the timer into the selected register. Now, why do we want the capture a mode? So, that many applications require you to measure time between two events.

So the way to do that would be on first event you capture the value of the timer, let the timer continue operating. Who is going to clock the timer? A known signal internally generated and then on the second event you capture the second value of the timer and the difference of the two values will give you the time that has elapsed between two events. This is what is listed in the slide.

Now to illustrate the capture features we are going to perform an interesting exercise which is to measure frequency of external signals.

(Refer Slide Time: 10:55)



For example, there is an instrument called frequency counter. What does a frequency counter do? It measures the frequency of external signals. Of course if the signal is sinusoidal or triangular it converts it into a digital equivalent signal and then using some mechanism it measures the frequency of that signal.

Now, to measure the frequency of the signal obviously you need to measure the pulses, the incoming pulses for a certain period of time. If you, if your measurement period is 1 second the number of pulses you have counted in that one second period will give you directly the frequency of the signal. Now to derive this one second reference you would have an internal clock which must be very accurate because if the clock is not accurate that one second period will be wrong and you will infer a wrong frequency.

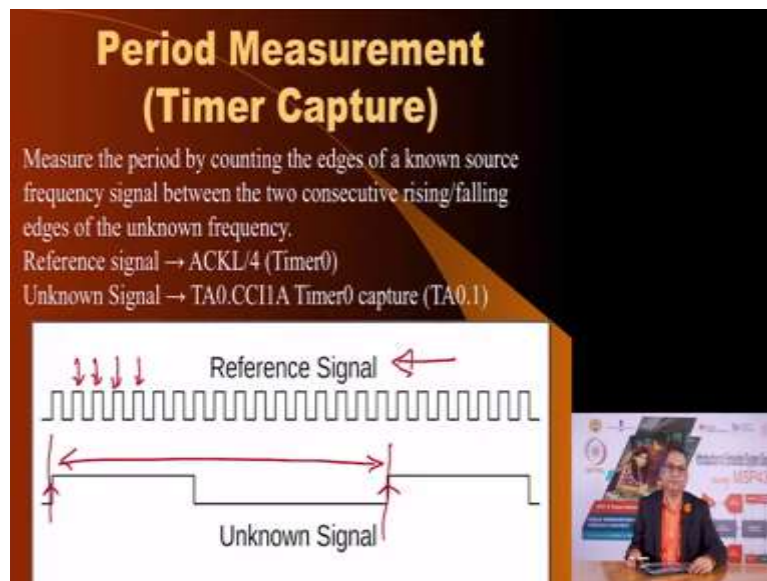
Now depending upon the source of the clock which is responsible for generating the one second period you have to decide if the incoming frequency that you want to measure is much higher than this clock signal or is it much lower.

If the frequency is much lower, if the frequency of the input signal that we want to measure is much-much smaller, very low then it is better instead of measuring the period it is better to measure the frequency. Once you measure the period you can do one by period to estimate the frequency.

If the frequency of the incoming signal on the other hand is much higher than the reference timer clock in the measurement instrument then it is better to measure just the pulses in that 1 second period and you would directly get the value of the frequency to a resolution of 1 second, 1 Hertz. If you want to have a resolution of 0.1 hertz, what should you do? You must increase your measurement window to 10 seconds and then divided by 10 so that you know is it 25.1 Hertz or 25.2 Hertz. So that is the method of measuring frequency directly.

Now, we will see how MSP430 could be used to perform both of these operations: the direct frequency measurement and the period measurement.

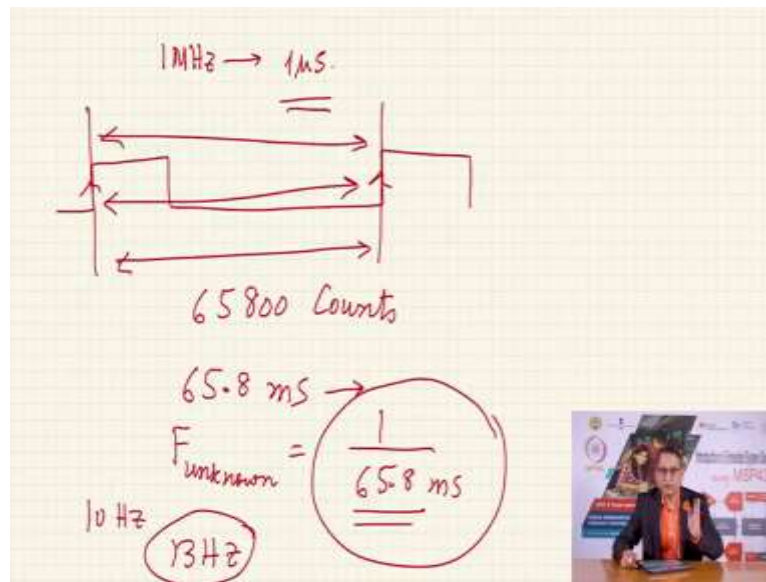
(Refer Slide Time: 13:24)



Now when you want to do period measurement we are going to use the timer in the capture mode and what are we going to do? Here is our reference signal this reference signal is an internal signal. The idea is that between this period of the unknown signal whose frequency we want to measure we do not want to measure frequency directly instead we want to measure the period of the signal and this can be done by counting the pulses of known frequency between the two rising edge of the unknown signal.

So, if I know how many pulses I have counted in that one period of the incoming unknown signal, if I know the period of the reference signal multiplying the count with the period I get the total period.

(Refer Slide Time: 14:16)



As an example, suppose the incoming the reference frequency inside reference signal that I have inside is let us say 1 mega Hertz that means the time period is 1 micro second. And let us say the incoming frequency I do not know the frequency, I do not know the period let us say I enable the counting process and in that counting process between the two rising edge of the incoming unknown frequency from here to here I accumulate a count of 65800 counts.

Since, I know the period of 1 clock signal of the reference which is 1 micro second therefore the total period is 65.8 milli seconds and from here I can estimate the frequency therefore the frequency of the unknown signal is equal to 1 upon 65.8 milli seconds.

Now, because we are doing all this in a embedded computer such as MSP430 and embedded computer can very easily perform this calculation. In the olden times when we did not have computers to perform this measuring period as a way of measuring frequency was left to just measure the period and display the period and leave it to the user to note the period and perform this calculation manually to estimate the frequency.

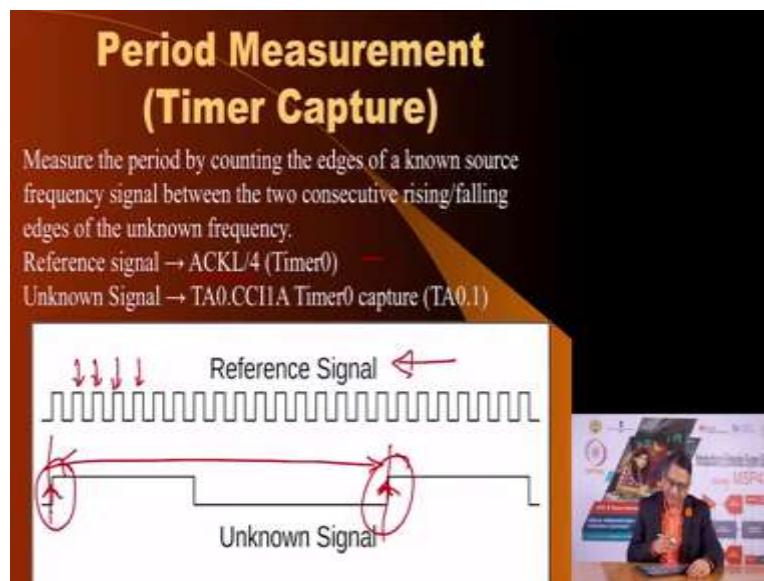
But today we do not have to do that, today we can measure the period and perform a mathematical calculation in the C program to estimate the frequency and we are forced to do this because the time period here is very-very low. See, if I have a 65 milli seconds and if I approximate to 100 milli second that means (the time) the frequency of is of this unknown signal is roughly a 10 Hertz.

In the case of 65 it will be about 13 Hertz. Now I want to measure the frequency of this unknown signal which is round about 13, 10, 20, 10 to 20 Hertz to a accuracy say of first digit. Then I would have to have a measuring window of 10 seconds so that I count 130 pulses divided by 10 to know whether it is 13.0 or 13.1 Hertz. 10 seconds is a very large window.

Now instead of doing that the period of this 13 Hertz signal is 50, 60 milli second I can easily which is a very small period in that I can easily count the number of clock signals of a known signal generated internally and then estimate the frequency to a second or third decimal place without spending 10 seconds that is the advantage of measuring the period as means of measuring frequency.

And the rule of thumb is if the frequency of the unknown signal is much lower than the reference frequency of your instrument you should measure period. If the frequency of the input signal is much higher than the reference frequency in your instrument then it is better to measure direct frequency. So we are going to in this first exercise we are going to measure the period of the unknown signal and from there estimate the value of the frequency.

(Refer Slide Time: 17:59)



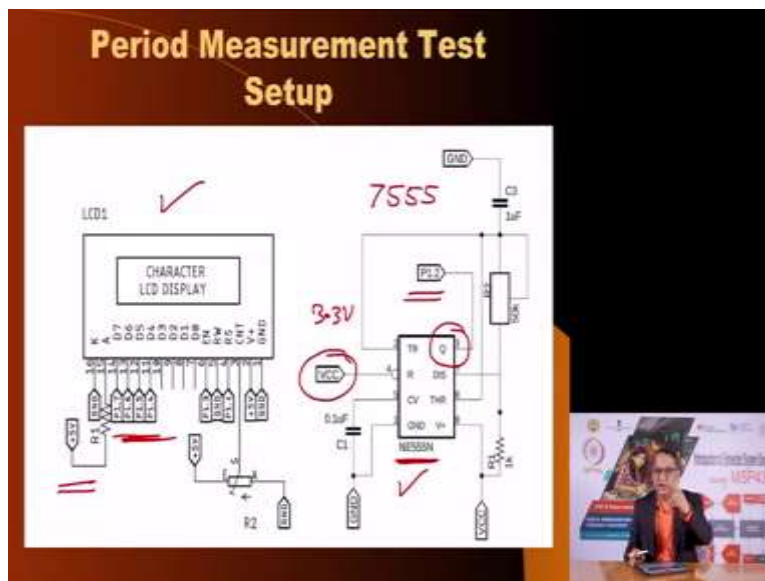
For that we are going to use timer 0 in the timer 0 the timer itself will be clocked by an internal signal derived out of the A clock, A clock is from the low frequency crystal oscillator because then it will be accurate as you know which is 32768 Hertz and we divide it by 4 so that we are

roughly getting 4 kilo Hertz, 4096 hertz frequency which is now we are going to be used as a reference frequency.

We are going to, we are going to count these pulses in the timer register. And the external signal whose frequency we want to measure the rising edge at the first rising edge we will capture the timer register value and on the second we will and we will note this value down in a variable and then when the second rising edge comes we will note the timer value again.

We know the timer is being clocked with ACLK by 4, so we know the period and once we know this second count we take the difference from the first count multiply it with the period of this reference signal which is 1 by 4096 Hertz because the frequency is 4096 Hertz so the time period is 1 by 4096 you will get the period of the unknown signal and then mathematically you can calculate the frequency also.

(Refer Slide Time: 19:28)




Now, to be able to measure obviously I need some source of unknown signal which I can verify so what we have done here is we will display the frequency and period on LCD we already covered how to interface the LCD to our MSP430 in this case we have used port 1 pins and to generate the unknown period or unknown frequency we have rigged up a triple 5 based astable multivibrator.

In fact this IC used here is 7555 so that it can work at 3 point VCC here is 3.3 volts. The VCC for this is 5 volts as we have gone through this earlier you remember the LCD works at 5 volts so the supply voltage for the 7555 is 3.3 volts and it is used to generate signal which is available on the output pin 3 and we are going to connect it to P1.2, why? Because, P1.2 will be the capture input capture compare input and that will be used to capture the value of the timer in to the capture compare register.

(Refer Slide Time: 20:44)

Selecting Timer Function on P1.2


PIN NAME (P1.x)	FUNCTION	CONTROL BITS AND SIGNALS ¹⁾				
		PISELx	PIELx	PIERLx	ADULTAKE ₀ (MCLR/P1)	CAPIEx
P1.0	PP1 x ASCII	1:0:0:1	0	0	0	0
TRACAP1	TRACAP1	0	1	0	0	0
ADCAP1	ADCAP1	1	1	0	0	0
ADP1	ADP1	0	1	0	0	0
CAPI1	CAPI1	0	0	1	0	1:0*2
CCP1	CCP1	0	0	1	0	1:0*2
PP1-Data	Capacitive sensing	0	0	0	0	0
P1.1	PP1 x ASCII	1:0:0:1	0	0	0	0
TRACAP1	TRACAP1	0	1	0	0	0
ADCAP1	ADCAP1	1	1	0	0	0
ADP1	ADP1	0	1	0	0	0
CAPI1	CAPI1	0	0	1	0	1:0*2
CCP1	CCP1	0	0	1	0	1:0*2
PP1-Data	Capacitive sensing	0	0	0	0	0
P1.2	PP1 x ASCII	1:0:0:1	0	0	0	0
TRACAP2	TRACAP2	0	1	0	0	0
ADCAP2	ADCAP2	1	1	0	0	0
ADP2	ADP2	0	1	0	0	0
CAPI2	CAPI2	0	0	1	0	1:0*2
CCP2	CCP2	0	0	1	0	1:0*2
PP2-Data	Capacitive sensing	0	0	0	0	0
P1.3	PP1 x ASCII	1:0:0:1	0	0	0	0
TRACAP3	TRACAP3	0	1	0	0	0
ADCAP3	ADCAP3	1	1	0	0	0
ADP3	ADP3	0	1	0	0	0
CAPI3	CAPI3	0	0	1	0	1:0*2
CCP3	CCP3	0	0	1	0	1:0*2
PP3-Data	Capacitive sensing	0	0	0	0	0




Now, since we are going to capture the you know the incoming frequency on P1.2 let us see which here is the P1.2 signal we want to select the capture compare register 1 it is an input so the direction of that is 0 and the rest of the PSEL and PSEL 2 bits are 1 and 0 that puts P1.2 that is the port 1 bit 2 as timer capture compare input that is what you have to write in your program.

(Refer Slide Time: 21:19)

```
1 #include <msp430.h>
2 #include <stdint.h>
3 #include <stdio.h>
4
5 #define ODD 0
6 #define DATA 1
7
8 #define LCD_OUT_PIN P1DIR
9 #define LCD_OUT_PIN P1DIR
10 #define LCD_OUT_PIN P1DIR
11 #define ODD 0
12 #define DATA 1
13 #define ODD 0
14 #define DATA 1
15 #define ODD 0
16 #define DATA 1
17 #define ODD 0
18 #define DATA 1
19
20 //write unsigned char count
21 //write unsigned int value, value, serial, // Global variables
22 //write mode from, time;
23
24
25
26 //write function for sending delay to 0.2 s, increments
27 //write 1 milliseconds to be delay
28 //write mode
29 //write mode
30
31 //void delay(int t)
32
33
34 void pulse(int)
35 {
36     digitalWrite(1, 1);
37     digitalWrite(1, 0);
38     digitalWrite(1, 1);
39     digitalWrite(1, 0);
40 }
41
42 void write(int t, int value, int mode)
43 {
44     digitalWrite(1, 1);
45     digitalWrite(1, 0);
46     digitalWrite(1, 1);
47     digitalWrite(1, 0);
48 }
```



```
1 //write function to send 0s after data is written
2 //write mode
3 //write mode
4
5 //void pulse(int)
6 {
7     digitalWrite(1, 1);
8     digitalWrite(1, 0);
9     digitalWrite(1, 1);
10     digitalWrite(1, 0);
11 }
12
13
14 //write function to write data/command to LCD
15 //write value value to be written to LCD
16 //write mode mode -> Command or Data
17 //write mode
18 //write mode
19
20 //void int_write(int t, int value, int mode)
21 {
22     if(mode == ODD)
23         digitalWrite(1, 1); // Set 0S -> 0S for Command mode
24     else
25         digitalWrite(1, 0); // Set 0S -> 0S for Data mode
26
27     digitalWrite(1, 1); // write high nibble first
28     pulse(1);
29     digitalWrite(1, 0);
30
31     digitalWrite(1, 1); // write low nibble next
32     pulse(1);
33     digitalWrite(1, 0);
34 }
35
36 }
```




```
17 /**
18  *Write function to print a string on LCD
19  *Write % pointer to the character to be written.
20  *Return void
21  **/
22 void lcd_print(char *)
23 {
24     while(*)
25     {
26         lcd_write("%d", 0000);
27         delay(100);
28     }
29 }
30
31 /**
32  *Write function to convert number into character array
33  *Write int integer number to be converted.
34  *Return void
35  **/
36 void lcd_printNumber(unsigned int num)
37 {
38     char buf[10];
39     char *str = buf+9;
40     *str = '\0';
41     do
42     {
43         unsigned long n = num;
44         num /= 10;
45         char c = (n % 10 + '0') + '\0';
46         *--str = c;
47     } while(num);
48     lcd_print(str);
49 }
50
```



```
18 /**
19  *Write function to move cursor to desired position on LCD
20  *Write row the cursor of the LCD
21  *Write col. Column cursor of the LCD
22  *Return void
23  **/
24 void lcd_gotoCursor(int r, int c)
25 {
26     const uint8_t row_offsets[] = { 0x00, 0x40, 0x80, 0xc0 };
27     lcd_gotoXY ( col = row_offsets[r], 0 );
28     delay(1);
29 }
30
31 /**
32  *Write function LCD
33  **/
34 void lcd_init()
35 {
36     LCD_PIN = (0x04-0x05+0x06+0x07);
37     LCD_CS = ~(0x04-0x05+0x06+0x07);
38
39     digitalWrite(LCD_CS, HIGH); // Pull CS pin up (15k)
40     lcd_write(0x01, 0); // Initialization Sequence 1
41     digitalWrite(LCD_CS, LOW); // Pull CS pin down
42     lcd_write(0x02, 0); // Initialization Sequence 2
43     digitalWrite(LCD_CS, HIGH); // Pull CS pin up
44
45     // All subsequent commands take 40 us to execute, except clear & cursor return (1.34 us)
46     lcd_write(0x03, 0); // 4 bit mode, 1 line
47     digitalWrite(LCD_CS, LOW);
48     lcd_write(0x04, 0); // Display ON, Cursor OFF, Blink OFF
49     digitalWrite(LCD_CS, HIGH);
50     lcd_write(0x05, 0); // Clear screen
51     digitalWrite(LCD_CS, LOW);
52     lcd_write(0x06, 0); // Set Display Cursor
53     digitalWrite(LCD_CS, HIGH);
54     lcd_gotoCursor(0,0); // Set Row 1 Column 1
55 }
56
```

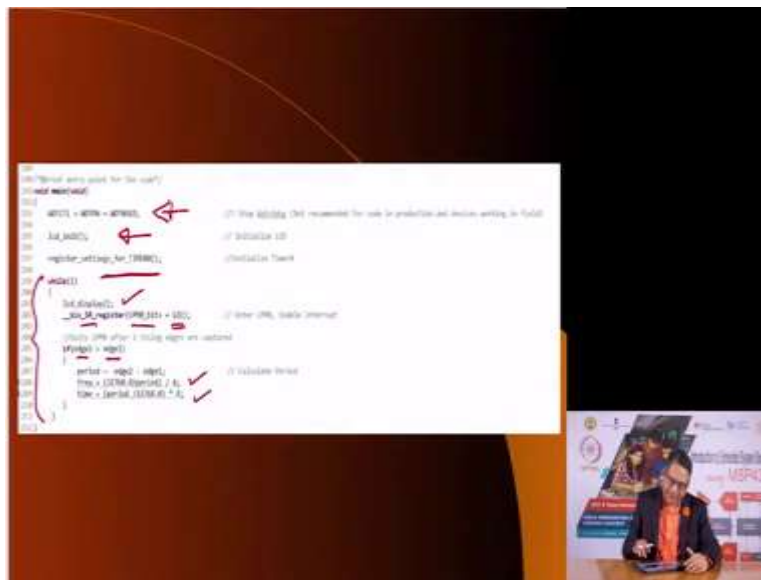


```
104 // Output Display on LCD
105 #define LCD_DISPLAY_ON_LCD
106 #define LCD_DISPLAY_ON_LCD
107 void lcd_display()
108 {
109     int part_time = time; // Integer part of calculated time
110     int actual_part_time = (time - (float)int_part_time) * 100.0; // Decimal part of calculated time
111     int part_freq = freq; // Integer part of calculated frequency
112     int actual_part_freq = (freq - (float)int_part_freq) * 100.0; // Decimal part of calculated frequency
113
114     lcd_write("TIME", 0, 0); // Clear screen
115     delay(200);
116     lcd.setCursor(0,0);
117     lcd_print("Period: ");
118     lcd_printNumber(int_part_time);
119     lcd_print(".");
120     lcd_printNumber(actual_part_time);
121     lcd_print("%");
122     lcd.setCursor(0,1);
123     lcd_print("Freq: ");
124     lcd_printNumber(int_part_freq);
125     lcd_print(".");
126     lcd_printNumber(actual_part_freq);
127     lcd_print("%");
128     delay(1000);
129 }
130
131 // Register settings for timer 0
132 #define REGISTER_SETTINGS_FOR_TIMER0
133 void register_settings_for_TIMER0()
134 {
135     PDIR0 |= 0x01; // Set P1.2 = Input
136     PDIR0 |= 0x02; // Set P1.3 = TMR0 Capture Pin
137     TMR0CTL0 |= CAP = 0x01 + CCI0A = 0x01 + CKDIV4; // Capture Mode, Rising Edge, Interrupt
138     TMR0CTL0 |= SYNCHRONIZE = 0x00; // Synchronize, Sync=0
139     TMR0CTL0 |= CLEAR_ON_CAPTURE = 0x01; // Clear = 0x01; Edge = 0x01; Clear = 0x01
140 }
```

Here is the entire program. I am not going to go through the program I have also I have already explained the crux of the program much of it is about the LCD which you already seen here are the generating the enable pin for the enable signal for the LCD this a function to write into the LCD. This is to print values which you have the data, the information that you want to print.

This is for printing numbers, this is to set the cursor, this is to initialize the LCD, this is to display the actual frequency values the period values and from there estimate the, calculate the frequency. This is important this is the register settings for timer 0 what we have done that we want to use P1.2 as a input in the capture mode and we want to use a rising edge, we want to have interrupt enabled and the source is (CCIA) CCI0A which is P1.2. And the clock for the timer is coming from ACLK divided by 4 this is the meaning of this line of code.

(Refer Slide Time: 22:35)

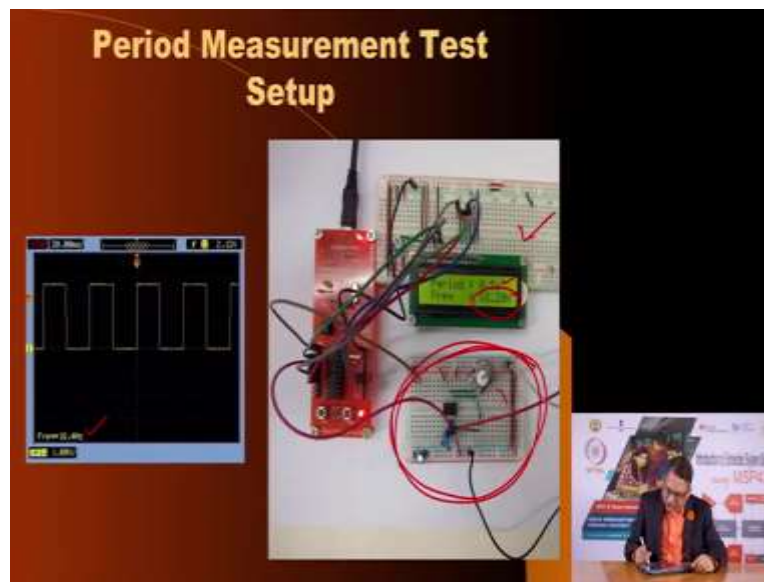


Here is a main program as usual we always disabled the watchdog timer, we have initialized the LCD, we have called the subroutine for the timer 0 initialization and now we are waiting in a infinite loop we are this is the infinite loop here.

In that we first display something in the LCD which maybe initially all wrong then we set the status register bits to enter in a low power mode because we only want to exist the low power mode whenever the interrupt is generated based on the rising edge of the incoming signal and we have enabled the global interrupt flag.

Now we are saying so this edge 2 and edge 1 are two variables into which the value of the timer registers are going to be stored. If they are, if edge 2 value is more than edge 1 now you can calculate the period and from there you can calculate the frequency and the time period like this.

(Refer Slide Time: 24:37)

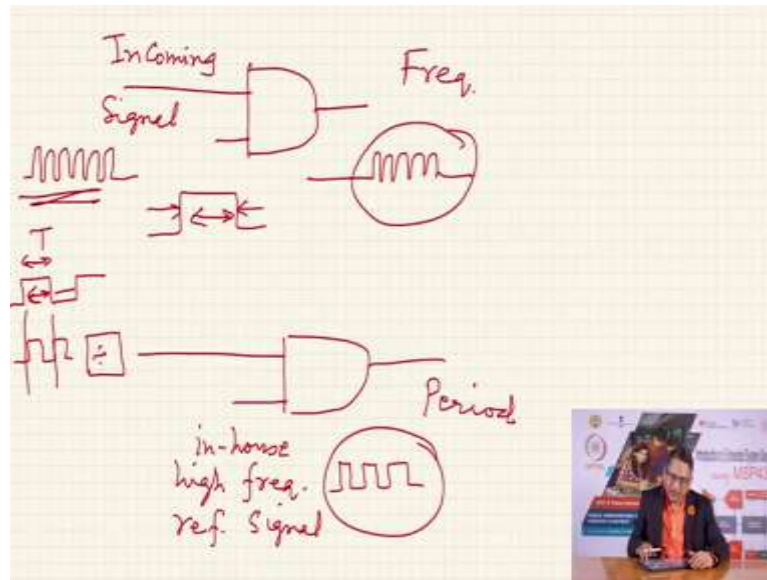


Here is the setup as you see here the frequency is very low as an example 18 Hertz and you see our now this is the information given by the oscilloscope which may not give very accurate values. Here we see the frequency is actually 18.28 Hertz ignore the period part because it is not printing the second decimal or third decimal in this case.

Here is the LCD the setup for the LCD and here is the setup for the 7555 to generate the unknown signal, here is a potentiometer so that we can vary the frequency and measure it on P1.2 and display the value here. This is the setup for the period measurement.

As I have discussed earlier if the frequency of the incoming signal is high then instead of measuring the period you can directly measure its frequency by enabling a mechanism to count the incoming pulses for 1 second period. Alright, let me go back here and may be you know recall this again.

(Refer Slide Time: 25:45)



So imagine that you have a AND gate, and here is your incoming signal which, whose frequency you want to measure. Imagine the second signal is a 1 second pulse. Now, because this is AND gate is going to allow incoming signals to pass here which could be high frequency like this and therefore you are going to get nothing and then high frequency for 1 second and so on. So, if this signal you feed it to a counter the counter at the end of for this 1 period is going to give you the frequency.

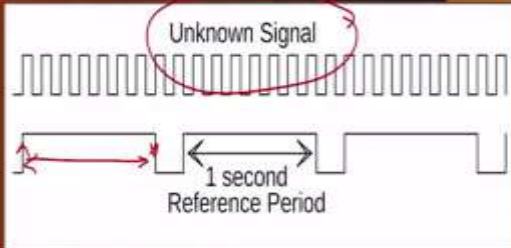
On the other hand, if the incoming frequency is very low then you do the following. You bring the incoming signal divide it by 2, why do I divide it by 2? So that I can get high for this and then low for the same period so in this one entire period of the unknown signal I measure a in house high frequency reference signal.

Now, what will happen? I do not know this period this is some value T , but I know the frequency of this which is being generated internally so if I count how many pulses I have accumulated in this time period I can estimate the value of this T . So, this is the period measurement and this is the frequency measurement. We have already seen the period measurement earlier now let us see how do we measure the frequency directly.

(Refer Slide Time: 27:38)

Direct Frequency Measurement (Timer as counter)

- The counter will count the number of unknown high frequency pulses during a period of known signal. Unknown Signal → TACLK (Timer0).
- 1 second Reference Period → Timer1



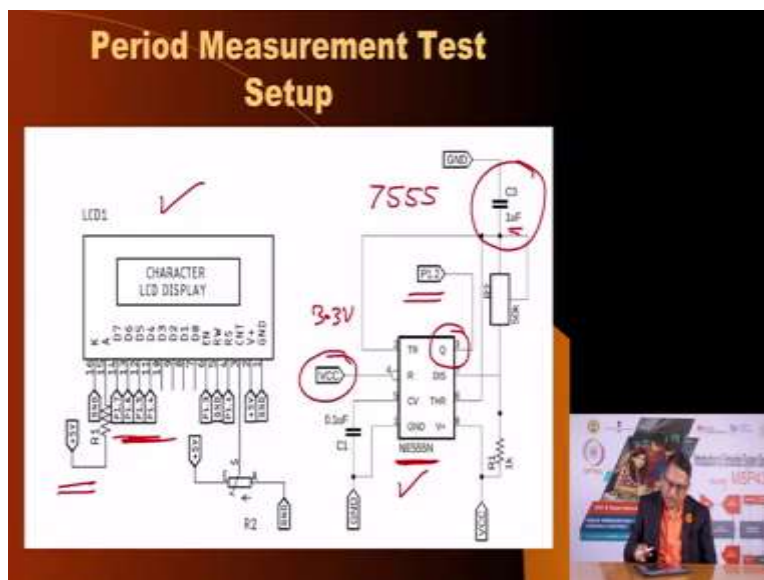
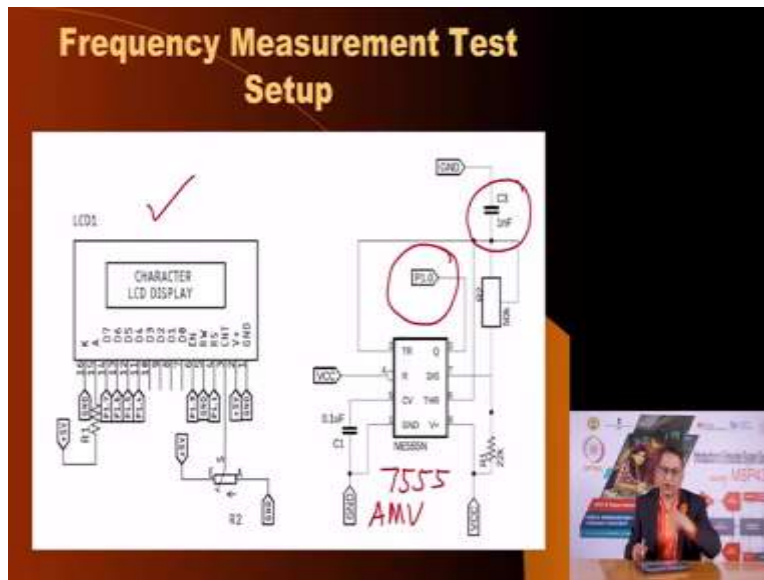
The diagram illustrates the timing for direct frequency measurement. It shows two waveforms: an 'Unknown Signal' (a high-frequency square wave) and a '1 second Reference Period' (a lower-frequency square wave). The unknown signal is shown as a series of pulses within the reference period. A red circle highlights the 'Unknown Signal' label.

We create reference internal reference signal of 1 second. Now, we cannot do both of these things using a single timer so for frequency measurement we are forced to use two timers. We are going to put the unknown signal on the input pin of the timer instead of an internal clock signal we are going to derive it out of an external pin and in that case it is TACLK we will see which pin it maps to and this is going to clock timer 0.

And we are going to generate 1 second reference signal on timer 1 that is the meaning of this setup. This is our unknown signal, this is our unknown signal here whose frequency we do not know but we know that internally we are able to generate 1 second reference period which we are going to generate from timer 1 and the incoming signal unknown signal we are going to feed to timer 0 into the timer signal directly into the timer.

And using we are going to generate interrupt here read the timer 0 value store it and when the second interrupt happens, when the second interrupt happens here this time in the falling edge we are going to capture the value of timer 0. Again, and the difference will be the frequency directly. Why? Because this period is 1 second.

(Refer Slide Time: 29:09)



Again for the setup we have created mechanism to display on LCD same as past and we have created a source of frequency which we can measure. But, instead of in the previous one the unknown signal being generated out of this 7555 IC it is in the astable multivibrator by the way both the cases. Here the frequency is higher and that you can how do I ensure that?

Let me go back to the previous setup here the period of that astable multivibrator is decided by this capacitor as you see this is 1 micro Farad. Compare this with the frequency with the capacitor of this timer 555 here it is thousand nano thousand pico Farad, 1 nano Farad. And so the frequency that will be generated here will be much higher than the previous case.


```

17 /**
18  *Serial function to write data to the LCD
19  *Write value to be written to LCD
20  *Write mode (0 = Command or Data)
21  *Return void
22  */
23 void lcd_write(int i, value, uint8_t mode)
24 {
25     if(mode == 0)
26         lcd.setCursor(0, 0); // Set 00 = 00 for Command mode
27     else
28         lcd.setCursor(0, 1); // Set 01 = 01 for Data mode
29
30     lcd.setCursor(0, 0); // Set 00 = 00 for Data mode
31     lcd.print(i);
32     delay(10);
33
34     lcd.setCursor(0, 1); // Set 01 = 01 for Data mode
35     lcd.print(i);
36     delay(10);
37 }
38
39 /**
40  *Serial function to print a string on LCD
41  *Write % pointer to the character to be written
42  *Return void
43  */
44 void lcd_print(char *)
45 {
46     while(1)
47     {
48         lcd.print("DATA");
49         delay(10);
50     }
51 }

```



```

17 /**
18  *Serial function to convert number into character array
19  *Write the integer number to be converted
20  *Return void
21  */
22 void lcd_printNumber(unsigned int num)
23 {
24     char buf[5];
25     char *str = buf+5;
26
27     *str = '\0';
28
29     do
30     {
31         unsigned long n = num;
32         num /= 10;
33         char c = (n % 10) + '0';
34         *--str = c;
35     } while(num);
36
37     lcd.print(str);
38 }
39
40 /**
41  *Serial function to move cursor to desired position on LCD
42  *Write row and Column of the LCD
43  *Write (x) Column Cursor of the LCD
44  *Return void
45  */
46 void lcd_goto(int x, int y, int col)
47 {
48     const uint8_t row_offsets[] = { 0, 1, 2, 3 };
49     lcd.setCursor(col + row_offsets[row], y);
50     delay(10);
51 }

```



```

11,**
12 // Global initialize LED
13 **
14 #include <led.h>
15
16 LED_PIN = (0A-05-06-07-08-09);
17 LED_PIN = (0A-05-06-07-08-09);
18
19 delay(100); // wait for power to 2.2V
20 led_write(LED1, ON); // Initialization Sequence 1
21 delay(100); // wait ( 0.1 sec )
22 led_write(LED1, ON); // Initialization Sequence 2
23 delay(10); // wait ( 100 ms )
24
25 // All subsequent commands take 40 us to execute, except clear & power return (2.00 us)
26 led_write(LED1, ON); // 4 bit mode, 2 lines
27 led_write(LED1, ON);
28
29 led_write(LED1, ON); // Write 0x, Green ON, Blue OFF
30 led_write(LED1, ON);
31
32 led_write(LED1, ON); // Clear screen
33 led_write(LED1, ON);
34
35 led_write(LED1, ON); // Auto Dependent Cursor
36 led_write(LED1, ON);
37
38 led_write(LED1, ON); // Set Line 5 Column 3
39
40,**
41 // Global Display on LED
42 **
43 #include <led.h>
44
45 led_write(LED1, ON); // Clear screen
46 led_write(LED1, ON);
47 led_write(LED1, ON);
48 led_write(LED1, ON);
49 led_write(LED1, ON);
50 led_write(LED1, ON);
51 led_write(LED1, ON);
52 led_write(LED1, ON);
53 led_write(LED1, ON);
54 led_write(LED1, ON);
55 led_write(LED1, ON);
56

```

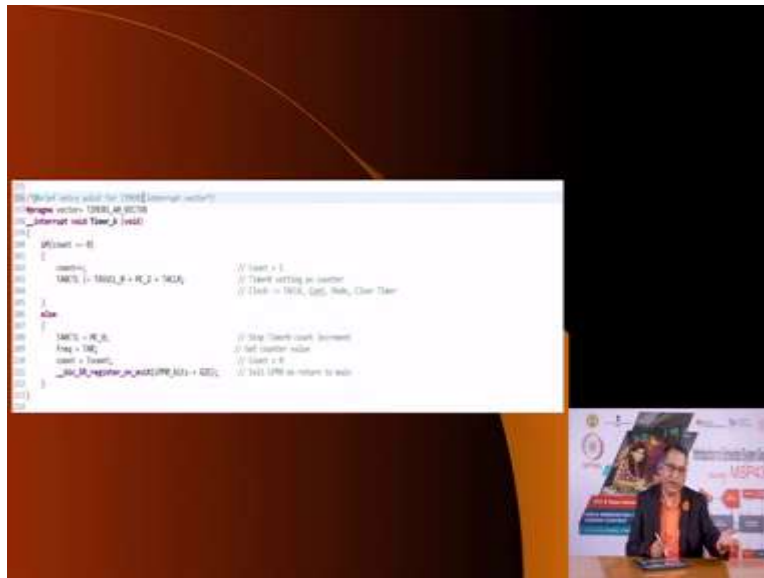


```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

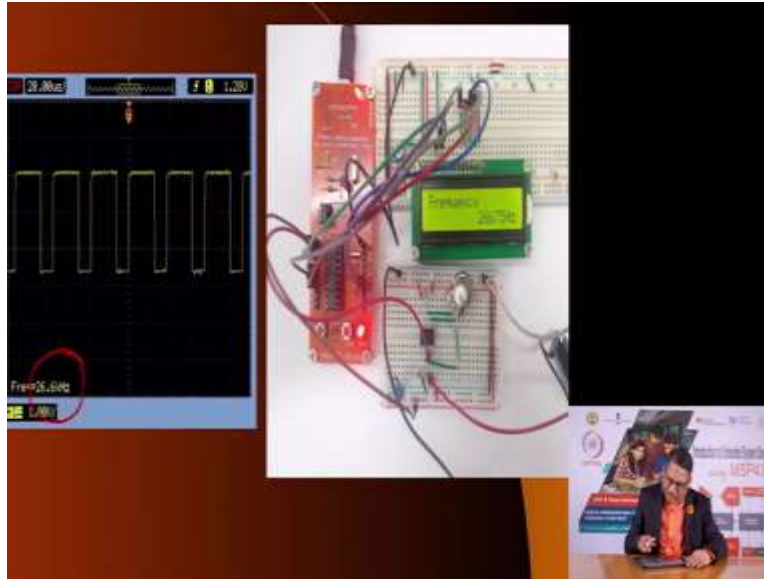




Here is the program for doing that again LCD part, the writing of the LCD, the setting of the cursor here, this is to initialize the LCD in four bit mode to display the value directly the frequency here and here is the register setting for timer 1 which is important because that is going to generate the 1 second period. This is the main program.

Since, we are generating the 1 second period from the, the low frequency crystal we have to wait for it to stabilize. Then bit 0 is the input P1.0 so that we have decided as input then we have initialize the LCD and we have call the function for timer 1 as we saw here and now we are waiting. Once the interrupt happens you are going to go in this interrupt subroutine and you are going to directly capture the value of the timer 0 value and use that as the count which is directly the frequency.

(Refer Slide Time: 32:10)



Here is a setup, as you see here the frequency of this unknown signal from the 555 is estimated to be 26.6 kilo Hertz. And the actual frequency down to 1 Hertz resolution measured by our setup is 26675 Hertz.

So, we have seen how the timer modules in the MSP430 microcontroller could be used for capturing external signals and using these modes you can measure frequency and period of unknown signals which is a very important activity often times in embedded applications. So, this completes the timer capture mode and I will see you very soon with a new lecture. Thank you.