

Introduction to Embedded System Design
Professor Dhananjay V. Gadre (NSUT) & Badri Subudhi
Indian Institute of Technology, Delhi
Lecture 30
Pulse Width Modulation, PWM Using Timer Capture

Hello, and welcome to a new session for this online course on Introduction to Embedded System Design. I am your instructor Dhananjay Gadre. In this session we are going to consider a very important aspect which is to be able to create analog waveforms using digital control. There are many methods of generating analog voltages with the help of a microcontroller. And 1 such method uses the timer building block that is a common peripheral in many microcontrollers and MSP430 is a very rich is very rich in this resource.

We have already introduced ourselves to the use of timer, how we can generate interrupts at predetermined intervals and periods using the timer. In this lecture, we are going to look at various methods of generating pulse width modulation signal, which is 1 method of generating analog voltages. Using a software approach as well as using a hardware approach which uses the timer.

So, the main topic of our discussion today is to be able to generate analog voltages, we will take two approaches, a software method which does not require any timer but we will tie down the CPU in just generating that analog voltage, it will not be able to do many other things simultaneously. If we want to do that, then we must rely upon the timer operation and many timer operations offer a PWM generation mechanism and we are going to explore that in this lecture today, so, let us begin.


(Refer Slide Time: 2:01)

Pulse Width Modulation

- PWM is a technique in which the width of a pulse is modulated to vary the average value of the signal.
- The width refers to the ON-time of the Rectangular pulse. The time period of the signal is kept fixed.

$$V_{AV} = V_{CC} \times D \quad 0 \leq D \leq 1$$

Duty Cycle = $\frac{T_{on}}{T_{on} + T_{off}} = D$



Pulse width modulation, as the name suggests, is a mechanism of modifying the width of modulating the width of a pulse waveform. Here by modifying the width, we are able to change the average value of the signal. Here is a sample signal we see there is a certain on time as you see here, the time for which the signal is high of the total time of time period of the signal. The ratio of the on time and the total time is defined as the duty cycle say D multiplied by the voltage the maximum voltage of that pulse waveform will give you the average voltage.

So, V average by changing the duty cycle of this waveform, which can go from 0 to 1, I can vary the average voltage from 0 to V_{cc} . This is a very good method of generating analog voltages. Now, the voltage itself appears rectangular, how do I get out of this the analog voltage which is proportional to the duty cycle? Well, you can do that by applying this signal to an appropriate low pass filter, we will see how to calculate the value of that filter and what sort of filter is used.

(Refer Slide Time: 3:38)

Pulse Width Modulation

- As the On-Time increases, the Duty Cycle of the Pulse increases and the average value of the signal increases.
- If a PWM signal is applied to an LED such that it is turned on and off at a very high rate that the human eye cannot detect the changes due to persistence of vision, the intensity of the led seems to vary.
- The output device operates on average value of this pulse.
- This mimics analog voltages through a digital output.

$$F_{PWM} = \frac{1}{T_{on} + T_{off}}$$

The slide also features a small inset image of a man in a suit sitting at a desk with a laptop, and a banner in the background that reads 'Introduction to Embedded System Design using MSP430'.

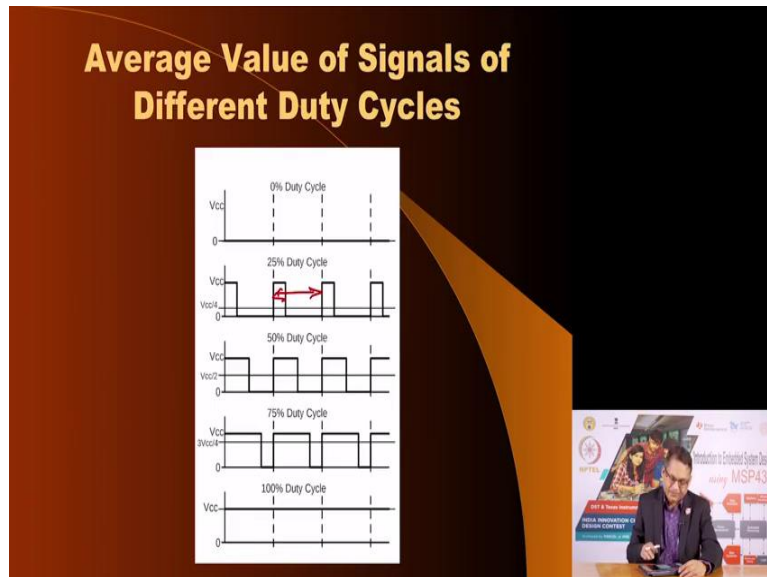
The property of pulse width modulation signal is that as you increase the on time, the duty cycle of the pulse width pulse increases and so does the average value. Now, if we apply the PWM signal to output device such as an LED and if we can keep frequency of the PWM signal frequency of the PWM signal F_{PWM} is equal to 1 by here the T_{on} plus T_{off} , T_{on} plus T_{off} .

If the frequency of the PWM signal is much higher than the rate at which the human eye can perceive on and off of a LED, then the eye will not perceive the LED as on and off instead, it will be able to only perceive intensity which lies between 0 and maximum, proportional to the duty cycle.

And this is because of the limitation of the human eye called persistence of vision. Apart from LEDs you can use a pulse width modulation signal to drive a DC motor and by changing the duty cycle you can change the average voltage applied to the DC motor and thereby you will be able to control the speed of the DC motor. We have discussed this in the past the speed of a DC motor is proportional to the voltage applied across the terminals.

And therefore, this is a good method by which we can mimic an analog voltage from a digital signal.

(Refer Slide Time: 5:23)



Here are examples of the average values of the PWM signal in this case, there is no waveform therefore, the average value is 0. Here the on time is 25 percent of the total time this is the total time the on time is 25 percent therefore, the average voltage is V_{cc} by 4 here it is 50 percent.

So, the average voltage is 50 percent V_{cc} by 2 in this case it is 3 fourth of V_{cc} because the duty cycle is 75 percent. And if the output is continuously high, the duty cycle is 100 percent and so you get full V_{cc} out.

(Refer Slide Time: 6:01)2

Software PWM

- Software PWM is implemented with the help of delays.

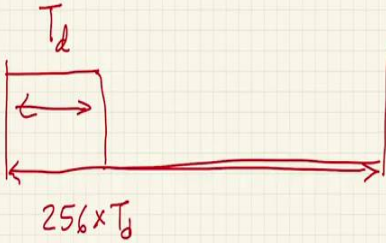

Step 1: Turn the Output Pin On using software by using PxOUT

Step 2: Give some delay (Ton)

Step 3: Turn the Output Pin Off using software by using PxOUT

Step 4: Give some delay (Toff)


- Software PWM is helpful when the pin cannot be configured as a Timer Output Pin.
- PWM Time Period = Ton + Toff.



$n \cdot T_d = '1'$
 $(256 - n) \cdot T_d = '0'$
 $0 \leq n \leq 256$

$\frac{n}{256} = \text{Duty Cycle}$

$\left(\frac{1}{256}\right) \times V_{CC}$



Now, as I mentioned we can generate PWM signal using a software loop, all you have to do is write a program in which an output pin any pin which can be converted which can be programmed as an output pin of a microcontroller can serve as a PWM output.

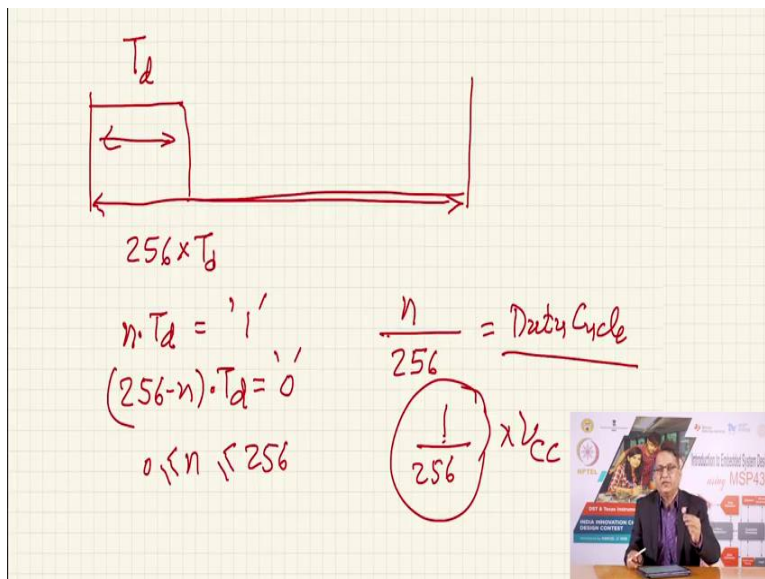
What you do is you find out you decide some amount of delay and that delay you should be able to replicate. So, let us say that I am able to let us say I have a total delay of T_d . Let us say the, this is the delay function I can write, this is the amount of delay that I can generate in a

guaranteed way. Now, what I will do is, I will have let us say 256, 256 periods of this T_d , into T_d .

Now, in this period of 256 into T_d , I can choose that the signal the pin is high for certain period and is low for the rest. Now, if this is some integer multiple n into T_d is equal to is kept at logic 1 and therefore 256 minus n into T_d is the time for which the output is 0. And therefore, I will be able to generate a duty cycle which is equal to n upon 256 is equal to the duty cycle. Now, by changing this n so, I can use increase in from this to 256.

As the value of n increases, you will see that the duty cycle increases. And because this is being done as a software program by writing an appropriate value of n , I will be able to change the duty cycle from any value of n equal to 0 to n is equal to 256 and therefore, I get a range of duty cycle which gives me a resolution that is I am able to resolve the duty cycle of 1 part in 256. This is the variation I cannot if I want to make output to analog voltage smaller than this resolution, this particular arrangement will not be able to do it, but 1 by 256 is the resolution into V_{cc} , into V_{cc} is the kind of minimum voltage that I can generate out of such a waveform.

(Refer Slide Time: 9:05)



Large Delay

```
1 #include <msp430.h>
2
3 #define RED BIT7 // Red LED -> P1.7
4
5 /**
6  *@brief This function provides delay
7  *@param unsigned int
8  *@return void
9  */
10 void delay(unsigned int t) // Custom delay function
11 {
12     unsigned int i;
13     for(i = t; i > 0; i--)
14     {
15         _delay_cycles(50); // _delay_cycles accepts only constants !
16     }
17 }
18
19 /*@brief entry point for the code*/
20 void main(void) {
21     unsigned int j;
22     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
23
24     P1DIR |= RED; // Set LED pin -> Output
25
26     while(1)
27     {
28         for(j = 0; j < 256; j++) // Increasing Intensity
29         {
30             P1OUT |= RED; // LED_ON
31             if (j != 0) delay(j); // Delay for ON Time
32             P1OUT &= ~RED; // LED OFF
33             if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
34         }
35         for(j = 255; j > 0; j--) // Decreasing Intensity
36         {
37             P1OUT |= RED; // LED_ON
38             if (j != 0) delay(j); // Delay for ON Time
39             P1OUT &= ~RED; // LED OFF
40             if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
41         }
42     }
43 }
44 }
```



```

// Custom delay function
void delay(unsigned int t)
{
  unsigned int i;
  for(i = t; i > 0; i--)
  {
    __delay_cycles(50);
  }
}

/*Brief entry point for the code*/
void main(void) {
  unsigned int j;
  WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

  P1DIR |= RED; // Set LED pin -> Output

  while(1)
  {
    for(j = 0; j < 255; j++) // Increasing Intensity
    {
      P1OUT |= RED; // LED ON
      if (j != 0) delay(j); // Delay for ON Time
      P1OUT &= ~RED; // LED OFF
      if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
    }
    for(j = 255; j > 0; j--) // Decreasing Intensity
    {
      P1OUT |= RED; // LED ON
      if (j != 0) delay(j); // Delay for ON Time
      P1OUT &= ~RED; // LED OFF
      if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
    }
  }
}

```

256x delay function



Let us see what kind of program we can write. so that we are able to generate this PWM signal, what we are going to do here is we are going to apply this software generated PWM signal on to the onboard LED of the lunch box. Therefore, you have to make any additional connections all you do is understand this code, download this code, rebuild it and download it into your lunch box.

When it starts working you will see that the intensity of the LED increases from minimum to maximum it goes up then it falls like this. Then it goes up again and so on and so forth. This is how it will appear. Let us go through this program. Now here to clarify, I have mentioned that there is large delay the reason for this large delay is so that when you see This PWM waveform being applied to the LED you can perceive you can easily see that the LED intensity is going gradually from minimum to maximum and then it is decrementing from maximum to minimum in the same at the same rate.

If we want to see this waveform on the oscilloscope we will have, will make some changes to the code by reducing the delay time period. So, let us see the first part of the code is that we have defined we have hash included the mandatory MSP430 header file. We have defined our LED to be on bit 1 point, P 1.7. Then we have created this delay function which is nothing but it goes through a certain amount of loop calling this sub routine where the variable is 50.

This gives you a certain amount of delay we will see how much delayed gives you and then this is the main part of the program that is all you require this entire code is able to generate software PWM in which it changes the duty cycle from 0 to maximum and then from maximum to minimum and it repeats it. And so, the end result is when you download this code on the lunch box, you will see that the LED on the lunchbox the intensity of the LED goes from minimum to maximum gradually and then in the same graduality it drops from maximum to minimum.

So, what we have done is we have as is mandatory when we are not using the watchdog timer we are turned off the watchdog timer. We have set the LED which is connected we have set the pin to which the LED is connected that is P 1.7. We have set the direction to be an output pin and then we enter into a infinite loop here. We have infinite loop. What are we doing in this? We have a variable J, which is going from 0 to 256.

And for that period for that loop, it turns the LED on here we are turning the LED on, then it goes to execute a delay subroutine equal to the value of J. So if the value of J initially is 0, it is not going to execute that delay subroutine. On the other hand, if j is any non 0 value, it is going to call the sub routine delay with the variable j. That means, if j is 1, it is going to go back into this and it is going to loop through this ones.

If j was 2 it is going to loop through it, loop through it twice, and so on. So it is going to turn the LED on for a certain period of time from time t equal to 0 to multiples of this basic time period into the value of j. And so what you will see is that the LED is for certain amount of time and for the rest of the time that is 256 minus this count is going to keep the LED off, we are turning the LED off here and then we are delaying for how much time 256 minus j and we are going through this loop by changing the value of j from 0 to 255.

So, what should you see that you will see that the LED intensity goes from minimum to maximum, then once this loop is over, it enters into the second loop here it decrements the value of j from 255 till it reaches 0. Now the duty cycle we will see goes from maximum to 0 and then this loop repeats. Now, based on the duty or the delay cycle here you would expect a certain amount of period.


The period of your PWM signal will be 256 into the delay period, delay time period, delay function. This is the basic, this is the basic period of the period of the PWM signal. Now, this is sufficient and suitable if you want to see the intensity of the LED, but if you want to observe it on the oscilloscope, this may be too slow and therefore, what we have done is we have simply recompile this code by changing the value of the delay cycle here from 50 to 1.

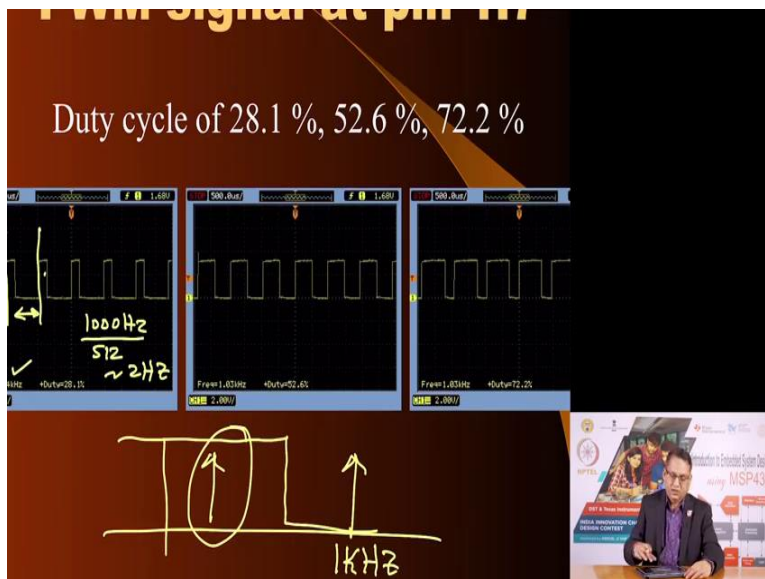
It is still going to call this function x number total number of 256 number of times for 1 period of which certain amount goes for the time when the LED is on and the rest the LED is off or in this case, pin is high and for the rest 256 minus that amount the pin is low.

(Refer Slide Time: 14:53)

Less Delay

```
1 #include <msp430.h>
2
3 #define RED BIT7 // Red LED -> P1.7
4
5 /**
6  *Brief This function provides delay
7  *Program unsigned int
8  *return void
9  */
10 void delay(unsigned int t) // Custom delay function
11 {
12     unsigned int i;
13     for(i = t; i > 0; i--)
14     {
15         __delay_cycles(1); // __delay_cycles accepts only constants !
16     }
17 }
18
19 /*Brief entry point for the code*/
20 void main(void) {
21     unsigned int j;
22     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
23     P1DIR |= RED; // Set LED pin -> Output
24
25     while(1)
26     {
27         for(i = 0; i < 256; i++) // Increasing Intensity
28         {
29             P1OUT |= RED; // LED ON
30             if (j != 0) delay(1); // Delay for ON Time
31             P1OUT &= ~RED; // LED OFF
32             if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
33         }
34         for(j = 255; j > 0; j--) // Decreasing Intensity
35         {
36             P1OUT |= RED; // LED ON
37             if (j != 0) delay(1); // Delay for ON Time
38             P1OUT &= ~RED; // LED OFF
39             if ((255-j) != 0) delay(255-j); // OFF Time = Period - ON Time
40         }
41     }
42 }
43
44 }
```





So, what we do is we recompile this code now, you see the delay cycles we are invoking is just 1. This will lead to a certain amount of time period and let me show you the PWM signal you see the time period of the PWM signal from, you receive from here, from here to here is as you see here it is 1 kilohertz.

Now in this 1 kilohertz waveform, what are we doing that this 1 kilohertz waveform we are repeating we are changing the value of duty cycle in every step from minimum to maximum in 256 steps and then from 256 to 0. So, therefore, there are 512 steps in the overall variation of the duty cycle from minimum to maximum and from maximum to minimum. Therefore, what kind of waveform of the signal that you expect the analog signal it will be 1000 hertz approximately divided by 512 approximately you should get 2 hertz.

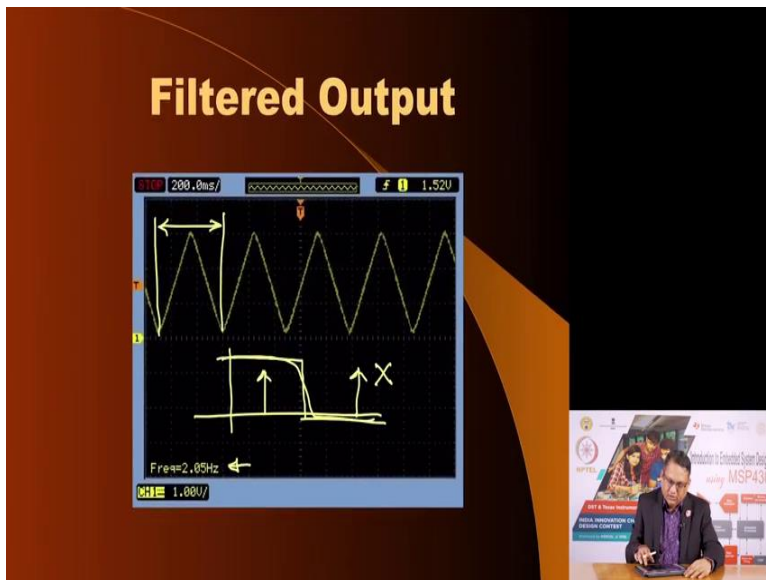
Let us see whether you get a 2 hertz signal or not. Now, as I mentioned to be able to get 2 hertz or whatever signal which is being used to modulate this duty cycle, the frequency of your PWM signal is here, this is 1 kilohertz and your message or the information which is modulating this is somewhere here and you want to extract this. So, what you do is you apply a low pass filter you pass this PWM signal through a low pass filter and the simplest low pass filter is a RC filter which is like this.

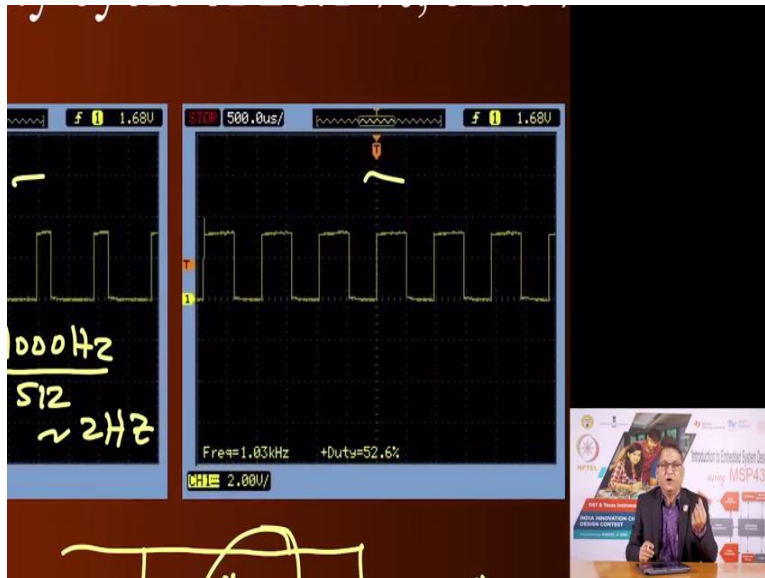
(Refer Slide Time: 16:37)

FILTER CONNECTION

The slide features a circuit diagram of an RC filter on the left, a photograph of the physical circuit on a breadboard in the center, and a calculation for the filter frequency at the bottom. The calculation is: $R = 1.5 \text{ Kohm}, C = 10 \text{ uF}$ and $\text{Filter Frequency} = 1/(2 \times 3.14 \times R \times C) = 10.6 \text{ Hz}$. A small inset video in the bottom right corner shows a person presenting the slide.

$R = 1.5 \text{ Kohm}, C = 10 \text{ uF}$
 $\text{Filter Frequency} = 1/(2 \times 3.14 \times R \times C) = 10.6 \text{ Hz}$





If I take R and C this is R then C the product of RC this frequency that comes out of here $\frac{1}{2\pi RC}$ is the 3 db frequency equal to here in this case for 1.5 and 10 micro farad you get 10 hertz. Which means of my, of my signal I have a something like this. So, this 3 db is 10 hertz here, but my carrier frequency that is the PWM carrier frequencies 1 kilohertz as you see it is much higher and my signal the signal which is modulating this is much lower as I mentioned, we estimate this signal to be 2 hertz. It is very well within the pass band of the low pass filter.

So, if I apply a low pass filter to the PWM signal at this point I should be able to see the signal going from 0 to maximum back to 0 and so on. And the period of that signal should be what we are predicted to be around towards that what we did was we took the output from P 1.7. Applied external RNC as you see here this is the resistor here and this is the capacitor. This is 10 kilo ohm, 1.5 kilo ohm in fact, as you see here 1.5 kilo ohm resistor and 10 micro farad in a electrolytic capacitor the resultant was this frequency and here is the filtered output.

And as you see here, the time period is indeed 2 hertz that is the time period from here to here this is the time period is 2 hertz and this matches exactly what we have been saying the requirement is that there should be large gap between the PWM carrier frequency that is the frequency of the PWM signal and the signal frequency which is your message and once that is there, you can easily filter out the PWM carrier waveform by passing it through a low pass filter.

If the difference is not large then you will need a much sharper low pass filter which means you will have to increase the order of this filter. We have already seen this how the duty cycle goes from in this case 28 percent and 52 percent and 72 percent but the period remains the same period remains the same, why? The sum of on time and off time is constant. Now, this is a method, which uses a program to generate a PWM signal if we do that, then the microcontroller is unable to do anything else.

And so, this is very useful only when this is the only thing your microcontroller has to do, but if the microcontroller has to do multiple things, while generating a PWM signal, then the only way you can achieve that is by outsourcing by offloading the job of generating the PWM signal to a peripheral and in this case, a timer is often very useful for doing that. In fact, MSP 430 microcontroller time have specialized features which allow it to generate PWM all you have to do is specify the duty cycle that you would want the PWM signal to operate at the frequency of the PWM signal and the resolution.

(Refer Slide Time: 20:25)

1bit
65536

Hardware PWM

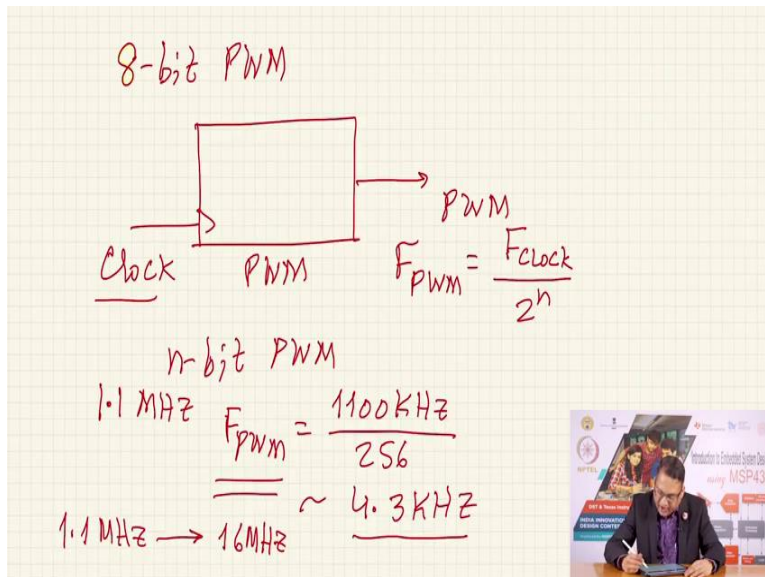
1. Duty Cycle
2. F_{pwm}
3. Resolution

← 16-bit →

Now you can control all these 3 things using the hardware PWM feature of MSP 430. Let me repeat what can you do you can specify duty cycle you can specify the P the time period that is the frequency of PWM. And 3 you can specify the resolution, resolution meaning what is the minimum variation in the duty cycle that you can achieve in the entire period which means if I have an 8 bit counter; my 8 bit counter has 256 possibilities.

So I can only change the duty cycle, 1 count in 8 bits, 1 count in 256 counts. On the other hand, if I have a 16 bit if I have a 16 bit time counter, 16 bit, then I can still vary 1 bit, I can change the duty cycle by 1 bit, but 1 bit out, out of how many 1 bit in 65,536. So, MSP430 allows you to go all the way from any arbitrarily small range to a large range of 16 bits. Now here is the impact of choosing or varying the resolution of the PWM signal.

(Refer Slide Time: 22:00)



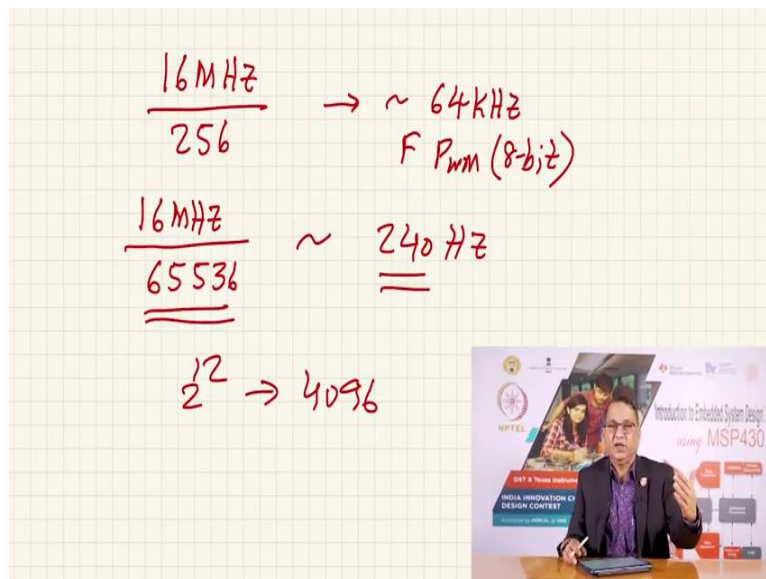
If you choose a 8 bit, if you choose 8 bit PWM signal, 8 bit PWM signal. What that does essentially is that, here is my PWM generator. PWM generator requires a clock the system clock or the clock that is being fed to this generator in this case a timer. It will generate PWM, the frequency of PWM will be equal to F clock that is the frequency of this clock signal divided by 2 raised to power n, where you are chosen a n bit PWM.

So, if n is 8, and let me take an example let us say the clock is as we know the default clock when you run your timer when you run your MSP430 the default clock at which the CPU starts working on the same clock can be fed to the timer is 1.1 megahertz then what happens is you have 1100 kilohertz divided by 256 is the frequency of the F PWM will be this and this will be roughly 4.3 kilohertz.

Now, the PWM frequency needs to be very high, why? So that it can keep you can keep a distance between the signal frequency, the frequency of the signal of interest, the message frequency and the carrier frequency. The frequency of the PWM signal is what we call as the carrier frequency. You want to have a large gap between the 2π , so that a reasonable filter can remove the carrier frequency and so that you can extract the analog voltage and therefore, it is very important that we should be able to have a high frequency of PWM signal.

But the PWM frequency is largely determined by this system frequency that is the frequency at which the timer is being clocked. And as you know, I can change the I can dynamically change or I can at reset I can decide to have an arbitrary value of the clock frequency, the default is 1.1 megahertz, but I can change that and I can use the DC and use the DC to generate a M clock and SM clock the 2 clocks which are used by the micro controller and I can go from 1.1 megahertz to 16 megahertz.

(Refer Slide Time: 24:51)



Handwritten calculations on graph paper:

$$\frac{16 \text{ MHz}}{256} \rightarrow \sim 64 \text{ kHz}$$

$F_{\text{PWM}} (8\text{-bit})$

$$\frac{16 \text{ MHz}}{65536} \sim \underline{\underline{240 \text{ Hz}}}$$
$$2^{12} \rightarrow 4096$$

The image also includes a small inset video of a presenter in a suit, with a background banner for "Introduction to Embedded System Design using MSP430" and "INDIA INNOVATION C1 DESIGN CONTEST".

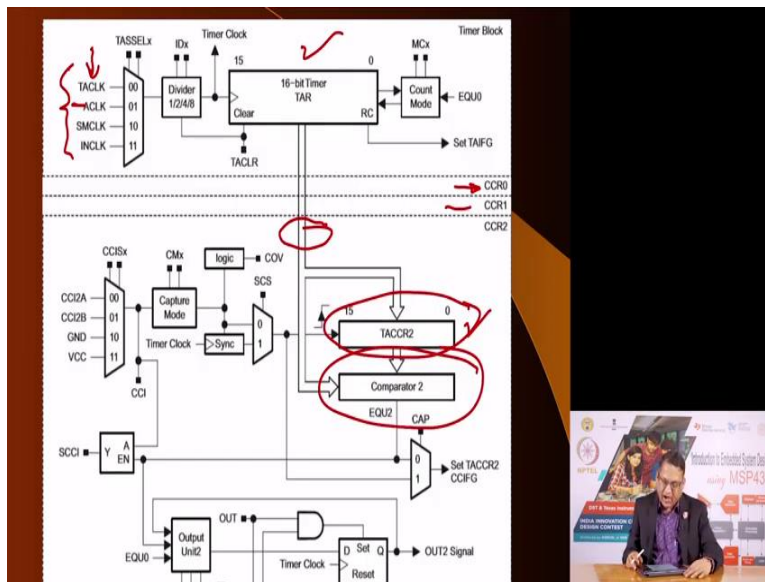
So, if I have if I choose to have a 16 megahertz clock and now if I choose to have a 8 bit PWM, what will I get? 16 megahertz divided by 256 will give you roughly 16 times more frequency than the 4 kilohertz that you have got, you are getting about 4 kilohertz when the signal clock frequency was 1.1 megahertz, now it has become 16 times more therefore, this will lead to roughly 64 kilohertz, PWM signal, f PWM for 8 bits.

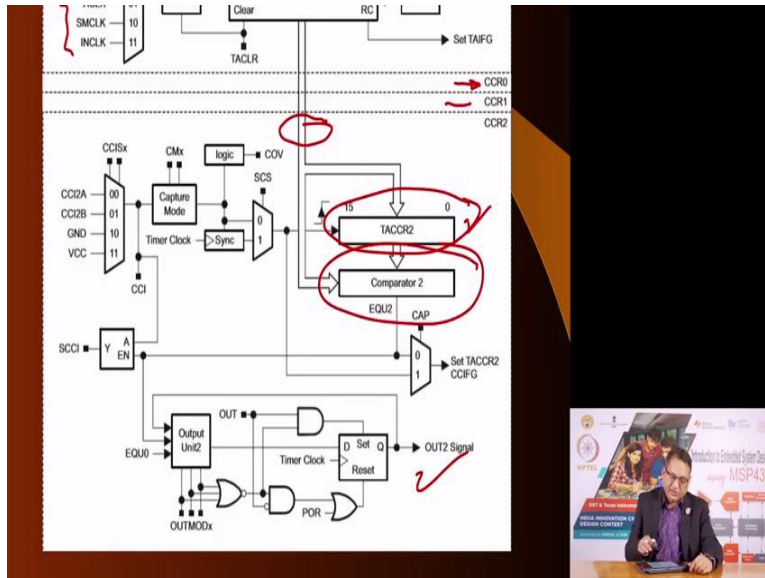
Now, if I want higher resolution, let us say I want a 16 bit resolution, then what I can do is I can still choose to have the highest because I cannot go beyond 16 mega hertz on the MSP 430, G 2553 and G Series of microcontrollers. But 16 megahertz divided by 65536, this will roughly give you, let me calculate here for you, it will give you roughly 240 hertz PWM carrier frequency.

And so, in many cases this could be sufficient. If not, you can trade the resolution you can choose to have a low resolution instead of 16, this is a 16 bit resolution, instead of 16 bit resolution you can go for it to 12 bit resolution 12 bit will give you 2 raise to power 12 which is 4096 and therefore, you will get roughly 16 times this frequency about 3, 3 and a half kilohertz PWM frequency.

So, you can using the keeping the constraint of the clock frequency in mind, the system clock frequency in mind and the resultant PWM frequency you can choose to have higher resolution or a lower resolution, but be aware that it is going to impact your carrier frequency of the PWM signal and if for a given application the carrier frequencies sufficient and suitable, why not, you can choose to have a higher resolution.

(Refer Slide Time: 27:24)





If not, you can give up the resolution choose a lower resolution and have a higher frequency of PWM signal. So, let us see how the timer functions, timer modes allow us to generate the PWM signal. For that, please look at this block diagram of the timer as we saw in a previous lecture on the timer operation, it consists of many parts the most important part is the timer, which in this case is a 16 bit timer MSP 430 timers are all 16 bit timers.

The source of the clock for these timers can be through many of the input signals input signal which is available from 1 of the pins or a clock which has which is internally generated as you see, you can choose the a clock Aclk or the SM clock to be the source for the timer the counting purpose. Now, you can you also know that the timer has a control register through which you can choose the mode of the timer in though there are 4 modes 1 you can stop the timer you can make it go in the up mode or you can go it in the up down mode or you can make it go in the continuous mode.

Now, based on that the count the timer will go from 0 to the maximum value. If it is the continuous mode the maximum value is the maximum value of a 16 bit number which is FFFF. On the other hand, in the other 2 modes, that is the UP mode and UP DOWN mode you have the maximum value that the timer will count up to will be determined by the control capture register are 0. Alright?.

So it is going to go count up to that and maybe it will come to 0 in the UP mode or from that value it will decrement to 0 1 bit at a time that depends on the mode of the timer. Now, that timer values available on the output bits of the timer 16 bits. They can be compared to a register whose value is in timer compare registers R2 or for that matter R1 or it could be captured into this or it could be compared with a value which is stored in this these registers


So, there are 3 channels, channel 0 is special, but channel 1 and channel 2 can be used to capture the value of the timer or the captured value could be compared with a value stored in those registers. And using that, in the last exercise when we looked at the timer what we saw that using this we were able to generate interrupts. So, that we knew we from the time that we started the timer we wanted an interrupt after a certain period of time.

And once that interrupt generated we restarted the timer and so on. Now, in this case, what we are going to do is that we can actually do more actions on the compare functions. Once the value is compared, you can decide what you want to do, and we want to send the compared output to output pins and this is the way that we will be able to generate a PWM signal. Let us see how that works.


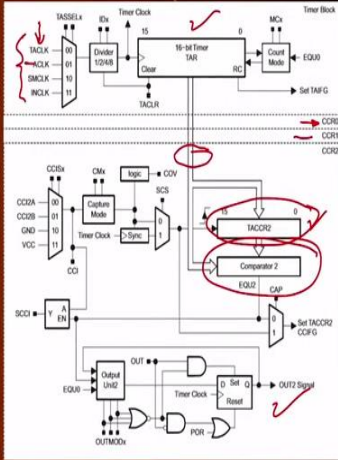
(Refer Slide Time: 30:46)

TIMER_A Output Unit

- Each capture/compare block contains an Output Unit.
- The compare mode is selected when bit CAP = 0 in TACCTLx register.
- When TAR value equals value in TACCRx, an internal signal EQUx is set, which affects the output according to output mode.
- The output unit is used to generate output signals such as PWM signals.
- Each output unit has eight operating modes that generate signals based on the EQU0 and EQUx signals.
- The output modes are defined by the OUTMODn bits in TACCTLx register



TIMER_A Block Diagram



So, the timer A output unit has a capture compare block which contains an output unit, here is the output unit, this part this is the output unit of the timer the compare mode is selected when in the timer control register you have you set the value of CAP equal to 0. If CAP is 1 then it goes in the capture mode, we are going to consider the capture mode in a subsequent lecture. When the timer value is equal to value in the TACRRx, which is 1 of the internal signals, the EQUx meaning if this is the TACCR1 then you are going to talk about EQU1 will be set which affects the output according to the output mode.

Similarly, the output unit is used to generate output signals such as PWM using another mode. Each output unit has 8 operating modes that generate signals based on the EQU0. On the EQU0 and EQUx signals EQUx here is either 1 or 2. The output modes are defined in the out mode bits in the timer control register.

(Refer Slide Time: 32:09)

Timer Modes

Timer's operation mode can be selected by configuring 'MCx' bits in the Timer Control TACTL register

MC	Mode	Description
00	Stop	Timer is stopped. ✓
01	Up ✓	Timer repeatedly counts from zero to value stored in TACCR0 Register. ✓
10	Continuous	Timer repeatedly counts from zero to 0xFFFF hex.
11	Up/Down	Timer repeatedly counts from zero to value stored in TACCR0 and then back to zero. ✓

The slide also features a video thumbnail of a presenter in a suit, with a background showing a presentation slide titled 'Introduction to Embedded System Design using MSP430'.

Let us see what are those output modes before that as I mentioned, the timer itself can be operated in 4 modes and this is programmed in the timer control register you can stop the timer you can make it function in the up mode in the mode the timer counts from 0 to a value stored in TACCR0.

So, TACCR0 is a very special register, therefore, that is that cannot be used in PWM signal generation R1 and R2 in some cases R2 can also be used. TACCR0 is used to store the value with which the timer value will be compared. In the continuous mode it will go from 0 to a maximum value that a 16 bit allows and in the up down mode it will go from 0 to the value stored to this and then from here it will decrement to 0.

And so, in the up mode, the waveform is more like this in the up down mode it is like this. So you get a saw tooth waveform of comparison in the mode and in the up down mode you get a triangle waveform.

(Refer Slide Time: 33:19)

TIMER_A Output Modes

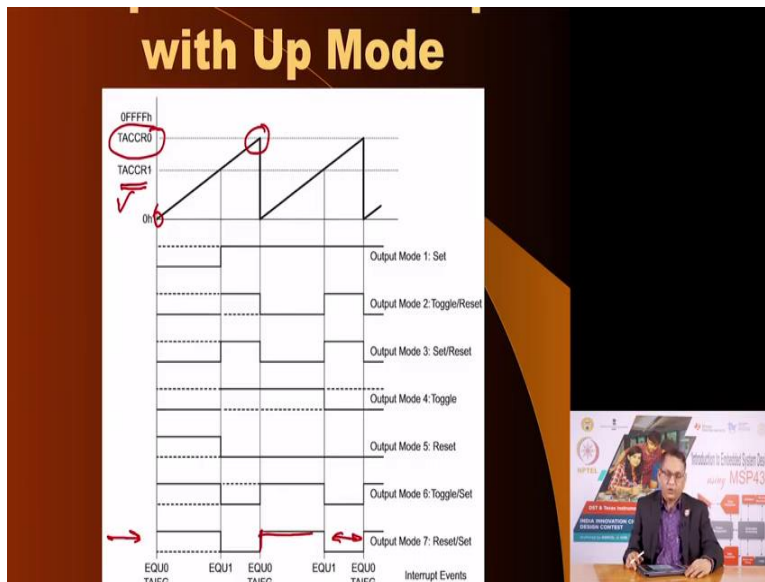
OUTMOD	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer counts to the TAxCCRn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer counts to the TAxCCRn value. It is reset when the timer counts to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer counts to the TAxCCRn value. It is reset when the timer counts to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer counts to the TAxCCRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer counts to the TAxCCRn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer counts to the TAxCCRn value. It is set when the timer counts to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer counts to the TAxCCRn value. It is set when the timer counts to the TAxCCR0 value.

MOD	Mode	Description
0	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
1	Set	The output is set when the timer counts to the TAxCCRn value. It remains set until of the timer, or until another output mode is selected and affects the output.
0	Toggle/Reset	The output is toggled when the timer counts to the TAxCCRn value. It is reset when timer counts to the TAxCCR0 value.
1	Set/Reset	The output is set when the timer counts to the TAxCCRn value. It is reset when counts to the TAxCCR0 value.
0	Toggle	The output is toggled when the timer counts to the TAxCCRn value. The output pe double the timer period.
1	Reset	The output is reset when the timer counts to the TAxCCRn value. It remains reset another output mode is selected and affects the output.
0	Toggle/Set	The output is toggled when the timer counts to the TAxCCRn value. It is set when counts to the TAxCCR0 value.
1	Reset/Set	The output is reset when the timer counts to the TAxCCRn value. It is set when counts to the TAxCCR0 value.

Here are the output modes if the output mode bits are 000 this is called the output mode and the out signal will be immediately available on that pin. You can have the set mode you can have toggle reset, but the 1 that we are interested in is the combination which is 111, in which the bit is reset to 0 and then is set to 0.

When is it reset? The output is reset when the timer counts to the value stored in this and this is for example, TACCR1 and it is set when the timer counts to value stored in TACCR0. So using TACCR1 and TACCR0 using this mode, which is the reset set mode you will be able to generate a PWM signal.

(Refer Slide Time: 34:12)



So let us look at that what happens. So you are you are chosen the up mode in which the value is going to go from 0 to the value stored in TACCR0.

When it matches the value of TACCR1 here, this is the seventh mode, the output toggles and then when it reaches this value it is set. So it is going to remain high for the time for the count stored in TACCR1, and for the rest of the time it is going to become 0. Therefore, by changing the value of the count stored in TACCR1 of any number from 0 to TACCR0, you are able to change the duty cycle from 0 to 100 percent.

The TACCR0 itself can be modified and if you modify it, it will change the period overall period of the PWM signal and that is what 1 means when you say that you are able to change the resolution. So, if I choose TACCR0 value to be the maximum value that is 65536 there is a resultant resolution will be 16 bits. On the other hand, if I said TACCR0 to 256 that means the resolution of the PWM signal will be 8 bits.

We are going to use this mode we are going to load values in the TACCR0 and TACCR1 register appropriately to generate similar waveforms that we generated in the software PWM mode.

(Refer Slide Time: 35:49)

PWM Using Timer

Step 1: To implement a Timer output, a pin which has timer output functionality has to be chosen. Let us choose P1.6 as TA0.1 output of Timer0.

DVCC	1	20	DVSS
P1.0/TA0CLK/ACLK/A0/CA0	2	19	XIN/P2.6/TA0.1
P1.1/TA0.0/UCARXD/UCASOMI/A1/CA1	3	18	XOUT/P2.7
P1.2/TA0.1/UCA0TXD/UCASIMO/A2/CA2	4	17	TEST/SBWTCK
P1.3/ADC10CLK/CAOUT/VREF-/A3/CA3	5	16	RST/NMI/SBWTDIO
P1.4/SMCLK/UCBSTE/UCACLK/VREF+/A4/CA4/TCK	6	15	P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/C
P1.5/TA0.0/UCB0CLK/UCAS0TE/A5/CA5/TMS	7	14	P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6
P2.0/TA1.0	8	13	P2.5/TA1.2
P2.1/TA1.1	9	12	P2.4/TA1.2
P2.2/TA1.1	10	11	P2.3/TA1.0

NOTE: ADC10 is available on MSP430G2553 devices only.
NOTE: The pull-down resistor of port P1 should be enabled by setting P1RDN = 1

lity has to be chosen. Let us choose P1.6
Timer0.

DVCC	1	20	DVSS
0/TA0CLK/ACLK/A0/CA0	2	19	XIN/P2.6/TA0.1
RXD/UCA0SOMI/A1/CA1	3	18	XOUT/P2.7
TXD/UCA0SIMO/A2/CA2	4	17	TEST/SBWTCK
/VREF-/VREF-/A3/CA3	5	16	RST/NMI/SBWTDIO
+/VREF+/A4/CA4/TCK	6	15	P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/C
/UCA0STE/A5/CA5/TMS	7	14	P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6
P2.0/TA1.0	8	13	P2.5/TA1.2
P2.1/TA1.1	9	12	P2.4/TA1.2
P2.2/TA1.1	10	11	P2.3/TA1.0

Input and Output Pins

You can find this information on Page 16 of the datasheet:

INPUT PIN NUMBER			DEVICE INPUT SIGNAL	MODULE INPUT NAME	MODULE BLOCK	MODULE OUTPUT SIGNAL	OUTPUT PIN NUMBER			
PW20, N20	PW28	RHB32					PW20, N20	PW28	RHB32	
P1.0-2	P1.0-2	P1.0-31	TACLK	TACLK	Timer	NA				
			ACLK	ACLK						
			SMCLK	SMCLK						
PinOsc	PinOsc	PinOsc	TACLK	INCLK						
P1.1-3	P1.1-3	P1.1-1	TA0.0	CC0A	CCR0	TA0	P1.1-3	P1.1-3	P1.1-1	
			ACLK	CC0B			P1.5-7	P1.5-7	P1.5-5	
			V _{SS}	GND			P3.4-15	P3.4-14		
			V _{CC}	V _{CC}						
P1.2-4	P1.2-4	P1.2-2	TA0.1	CC1A	CCR1	TA1	P1.2-4	P1.2-4	P1.2-2	
			CAOUT	CC1B			P1.6-21	P1.6-21	P1.6-21	
			V _{SS}	GND			P2.6-27	P2.6-26	P2.6-26	
			V _{CC}	V _{CC}			P3.5-19	P3.5-18		
P3.0-9	P3.0-7	P3.0-7	TA0.2	CC2A	CCR2	TA2	P3.0-9	P3.0-7	P3.0-7	
PinOsc	PinOsc	PinOsc	TA0.2	CC2B			P3.6-20	P3.6-19		
			V _{SS}	GND						
			V _{CC}	V _{CC}						



Now, if you want to use the timer for PWM, the output signal can be routed on any given pin. In this case we will route the pin on this P 1.6 that is TA 0.1. And let us see how we do that. These are the pins which are associated with the timer A0 input and output pins we are interested in this.

We are going to use CCR 1 here we are going to store the compare the value which will determine the frequency of the PWM signal and this will give us output on P 1.6. You can also route it to P 2.6 or you can route it to P 1.2 you can do that.

(Refer Slide Time: 36:49)

MSP430G2553 Timer1_A Input and Output Pins

You can find this information on Page 16 of the datasheet:

INPUT PIN NUMBER			DEVICE INPUT SIGNAL	MODULE INPUT NAME	MODULE BLOCK	MODULE OUTPUT SIGNAL	OUTPUT PIN NUMBER			
PW20, N20	PW28	RHB32					PW20, N20	PW28	RHB32	
-	P3.7-21	P3.7-20	TACLK	TACLK	Timer	NA				
			ACLK	ACLK						
			SMCLK	SMCLK						
-	P3.7-21	P3.7-20	TACLK	INCLK						
P2.0-8	P2.0-10	P2.0-9	TA1.0	CC0A	CCR0	TA0	P2.0-8	P2.0-10	P2.0-9	
P2.3-11	P2.3-16	P2.3-12	TA1.0	CC0B			P2.3-11	P2.3-16	P2.3-15	
			V _{SS}	GND			P3.1-8	P3.1-6		
			V _{CC}	V _{CC}						
P2.1-9	P2.1-11	P2.1-10	TA1.1	CC1A	CCR1	TA1	P2.1-9	P2.1-11	P2.1-10	
P2.2-10	P2.2-12	P2.2-11	TA1.1	CC1B			P2.2-10	P2.2-12	P2.2-11	
			V _{SS}	GND			P3.2-13	P3.2-12		
			V _{CC}	V _{CC}						
P2.4-12	P2.4-17	P2.4-16	TA1.2	CC2A	CCR2	TA2	P2.4-12	P2.4-17	P2.4-16	
P2.5-13	P2.5-18	P2.5-17	TA1.2	CC2B			P2.5-13	P2.5-18	P2.5-17	
			V _{SS}	GND			P3.3-14	P3.3-13		
			V _{CC}	V _{CC}						




If on the other hand you are using timer 1 for PWM, then you have the timer CCR 1 available on P 2.1 or P 2.2. Here it is P 1.6 which is pin number 14 and these are for 20 pin ICs which is the ones we are using on our lunch box.

(Refer Slide Time: 37:14)

To select the timer function of that pin, PxDIR, PxS and PxSEL2 Registers are used.

P1.6

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾					
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.y+1 ⁽²⁾	JTAG Mode	CAPD.y
P1.6/		P1.x (IO)	I/O: 0: 1	0	0	0	0	0
TAD.0/		TAD.0	1	1	0	0	0	0
UCB0CLK/		UCB0CLK	from USCI	1	1	0	0	0
UCA0STE/		UCA0STE	from USCI	1	1	0	0	0
AS ⁽³⁾ /	5	AS	X	X	X	1 (y=5)	0	0
CAS		CAS	X	X	X	0	0	1 (y=5)
TMS		TMS	X	X	X	0	1	0
Pin Osc		Capacitive sensing	X	0	1	0	0	0
P1.6/		P1.x (IO)	I/O: 0: 1	0	0	0	0	0
TAD.1/		TAD.1	1	1	0	0	0	0
UCB0SOM/		UCB0SOM	from USCI	1	1	0	0	0
UCB0SCL/		UCB0SCL	from USCI	1	1	0	0	0
AS ⁽³⁾ /	6	AS	X	X	X	1 (y=6)	0	0
CAS		CAS	X	X	X	0	0	1 (y=6)
TDWTCLK/		TDWTCLK	X	X	X	0	1	0
Pin Osc		Capacitive sensing	X	0	1	0	0	0



Now, to be able to tell the microcontroller because you see you are going to use P 1.6 which is otherwise input or output pin export 1 bit 6 you want to use it for timer function that is PWM function. So, you must tell the control registers associated with the port 1 that you want to use this pin as a PWM signal and for that, what we are going to do is for this pin, we are going to invoke this function so, in the P 1 register it is 1 so that it is an output. P 1 select we will write 1, P 1 select 2 we will write 0 and for the rest of them we will write 0.

And so, you must program the ports associated with P 1 point port 1 in this fashion so that P 1.6 can act as a PWM output signal associated with timer 0.

(Refer Slide Time: 38:20)

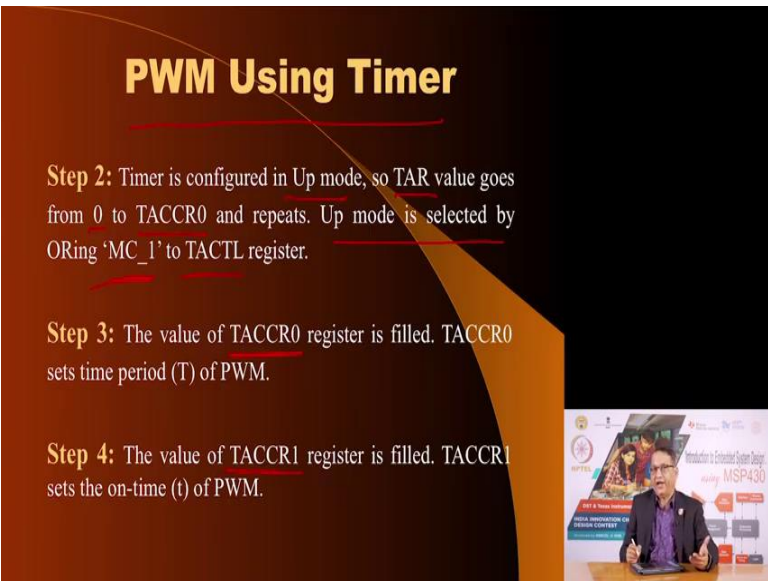


TIMER_A Registers

1. Timer_A control - TACTL
2. Timer_A counter - TAR
3. Timer_A capture/compare control 0 - TACCTL0
4. Timer_A capture/compare 0 - TACCR0
5. Timer_A capture/compare control 1 - TACCTL1
6. Timer_A capture/compare 1 - TACCR1
7. Timer_A capture/compare control 2 - TACCTL2
8. Timer_A capture/compare 2 - TACCR2
9. Timer_A Interrupt vector - TAIV

These are the registers associated with timer A we have already seen that in the previous lecture, so, I am not going to repeat that you will see it when we look at the code.

(Refer Slide Time: 38:31)



PWM Using Timer

Step 2: Timer is configured in Up mode, so TAR value goes from 0 to TACCR0 and repeats. Up mode is selected by ORing 'MC_1' to TACTL register.

Step 3: The value of TACCR0 register is filled. TACCR0 sets time period (T) of PWM.

Step 4: The value of TACCR1 register is filled. TACCR1 sets the on-time (t) of PWM.

Now, to use the PWM signal or using a timer, you have to do various steps. You have to configure the timer in up mode. The timer value goes from 0 to the value stored here and it repeats. In up mode the up mode is selected by ORing this, this is a mask to the timer control

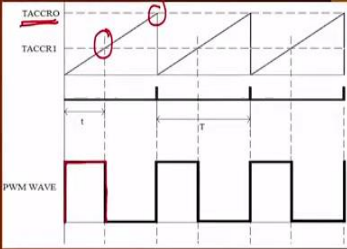
register another step the value of this is filled and this will determine the frequency or the period of the PWM signal. And then you write a value in the CCR 1, which will decide the on time that is the duty cycle of the PWM signal.

(Refer Slide Time: 39:16)

PWM Using Timer

Step 5: Output mode 7 is selected by writing 'OUTMOD_7' to TACCTL1 register : `TACCTL1 = OUTMOD_7;`

- Output resets when timer (TAR) value counts to TACCR1 value.
- Output value sets when timer (TAR) value count reaches TACCR0 value.

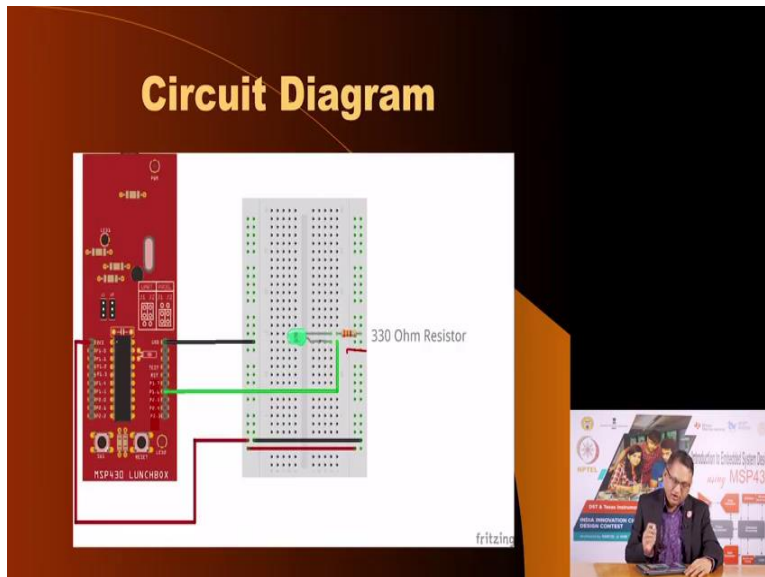


The diagram illustrates the relationship between the timer counter (TAR) and the PWM output. The TACCRO register defines the period of the PWM signal, while the TACCR1 register defines the pulse width (duty cycle). The PWM WAVE shows a high pulse followed by a low pulse, repeating periodically.

And last but not the least, you must tell the port that you want to route the output signal, you want to select the output mode 7 and you want to route it into the P 1.6 pin. Now, here is an example what will happen that when the counting starts the output will become high and it will remain high till the counter value reaches TACCR1, then it turns low, and it remains low till it reaches the value of TACCR0, and then the cycle repeats.

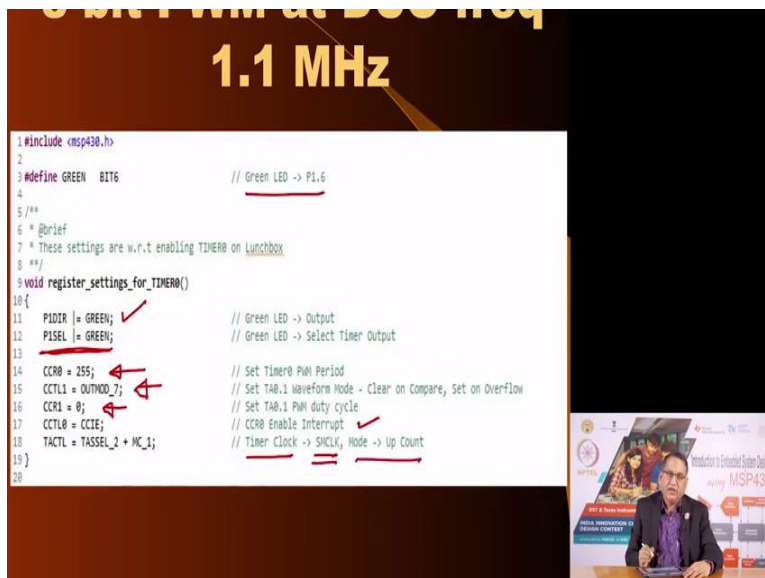
Now as you can see, if I increase the TACCR0 value, it is going to increase the period that but it will give me higher resolution. If I reduce it, it will reduce the period meaning it will increase the frequency but it will also reduce the resolution.

(Refer Slide Time: 40:10)



In our lunch box, we do not have any LED on the PWM in on P 1.6. As you know we have LED on P 1.7. But unfortunately we cannot route the PWM output on P 1.7. So, we have to connect an external LED we with a series resistor to P 1.6.

(Refer Slide Time: 40:32)



Here is the code we have decided we have connected a LED to P 1.6. These are the resistors, this is a subroutine which we are going to call for timer 0. We are selecting the LED as output and

we are selecting the timer output on this. And then we are setting a value in CCR 0 equal to 255 which means we are choosing a 8 bit resolution.

We are choosing mode 7, output mode is mode 7. The value in CCR1 register is initially set to 0 which means initially the duty cycle will be 0 and the as interrupt is enabled. So, that after every compare, after every P time P 1 period of PWM signal will have an opportunity to either change the value of CCR1 or CCR0 if you like. So, we have enabled the interrupts and we are using the timer clock, getting the timer clock from SM clock and the mode is in the up mode.

(Refer Slide Time: 41:45)

```
21 /**
22 * @brief
23 * Entry point for the code
24 **/
25 void main(void) {
26
27     WDCTL = WDTMW | WDTWLD; // Stop watchdog timer
28
29     register_settings_for_TIMER0();
30
31     _bis_sr_register(CCR1); // Enable CPU Interrupt
32
33     while(1)
34     {
35     }
36
37 }
38
39 /**
40 * @brief
41 * Entry point for TIMER0_interrupt vector
42 **/
43 #pragma vector = TIMER0_VECTOR
44 __interrupt void TIMER0_A(void)
45 {
46     CCR1 = CCR1 + 1; // Increment CCR1
47     if(CCR1 == 256)
48     {
49         CCR1 = 0;
50     }
51 }
52
```

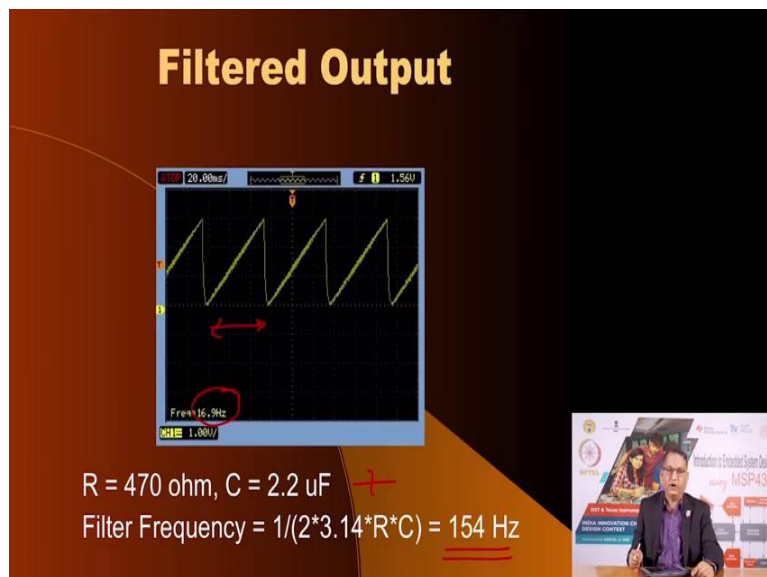
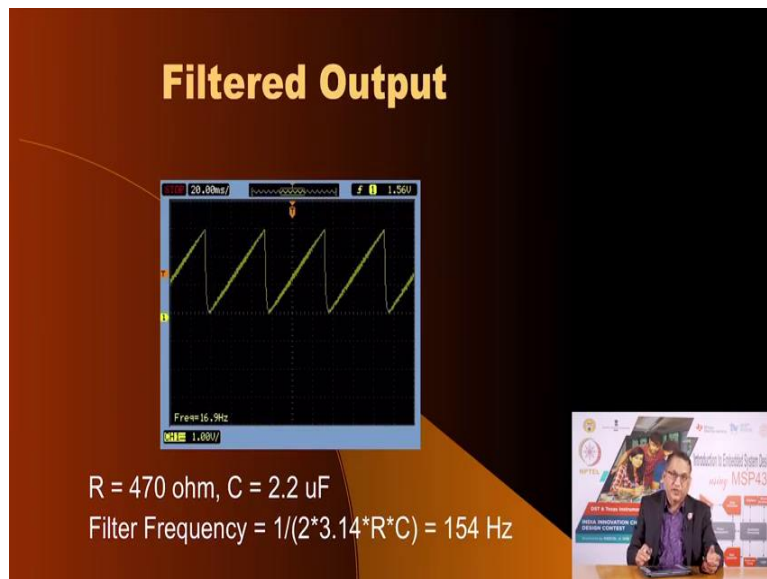
Here is the main program, we have simply turned the watchdog timer off, we have called that function which we have just discussed and we have enabled the interrupts and we are just waiting in a infinite loop. Now, every time the timer overflows that is when it reaches the TACCR0 it will generate an interrupt and it will go into this. Where what we are doing is we are simply incrementing the value of CCR1.

And so, CCR1 as we saw CCR1 started with 0, it will keep on incrementing because of this but the moment it reached 256 we want to make it 0. So, if after incrementing value as 256 make the value of CCR1 equal to 0, which means the value of CCR1 will go from 0 to 255. And when it goes when it is 0 the duty cycle will be 0, when it is 255 the duty cycle will be 100 percent. And

once you run this program you will see that the intensity of the LED goes from minimum to maximum abruptly going to 0.

So, that the intensity will give you a saw tooth waveform because of this.

(Refer Slide Time: 42:57)



Here is the filtered output. Since we know that we are the DC frequency is 1.1 mega hertz and we are choosing the 8 bit resolution the resultant frequency will be about; here is the PWM single, it will be about 4.3 mega, kilo hertz which we just calculated. And because there are 256


levels, so, 4.3 kilo hertz divided by 256 will give you the resultant signal which as you see here is 16 hertz, 16, 17 hertz.

And this value has been obtained, this signal has been obtained by filtering the PWM signal. This is what being generated by microcontroller by itself, this is the result of the filtered output and the filter characteristics is that it has a frequency of, cut off frequency of 154 because of these values involved and this is much lower than the PWM signal frequency. Therefore, the PWM signal will be filtered out and it is much higher than the message that is the signal frequency. Therefore, that saw tooth waveform will flow easily out of it and that is what you see here.

(Refer Slide Time: 44:17)

Hardware PWM Code 2 16 bit PWM at DCO freq - 16 MHz

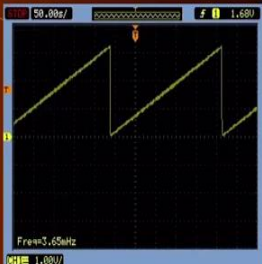
```
1 #include <msp430.h>
2
3 #define GREEN BIT6 // Green LED -> PL1.6
4
5 /**
6  * @brief
7  * These settings are w.r.t enabling TIMER0 on LUX8060
8  */
9 #void register_settings_for_TIMER0()
10 {
11     PDIR |= GREEN; // Green LED -> Output
12     PSEL |= GREEN; // Green LED -> Select Timer Output
13
14     CCR0 = 65535; // Set Timer0 PWM Period
15     CCTL1 = OUTMOD_7; // Set TMR0.1 waveform mode - Clear on compare, Set on overflow
16     CCR1 = 8; // Set TMR0.1 PWM duty cycle
17     CCTL0 = CCIE; // CCR0 Enable Interrupt
18     TACTL = TASSEL_2 + MC_1; // Timer Clock -> SHCLK, Mode -> up count
19 }
20
```




Here is another program in which certain things are done, 1, the DCO frequency has been jacked up from 1.1 mega hertz to 16 mega hertz, rest everything is changed apart from instead of 8 bit PWM we have chosen 16 bit PWM. I strongly recommend you go through this program and see the output.

(Refer Slide Time: 44:33)



Filtered Output

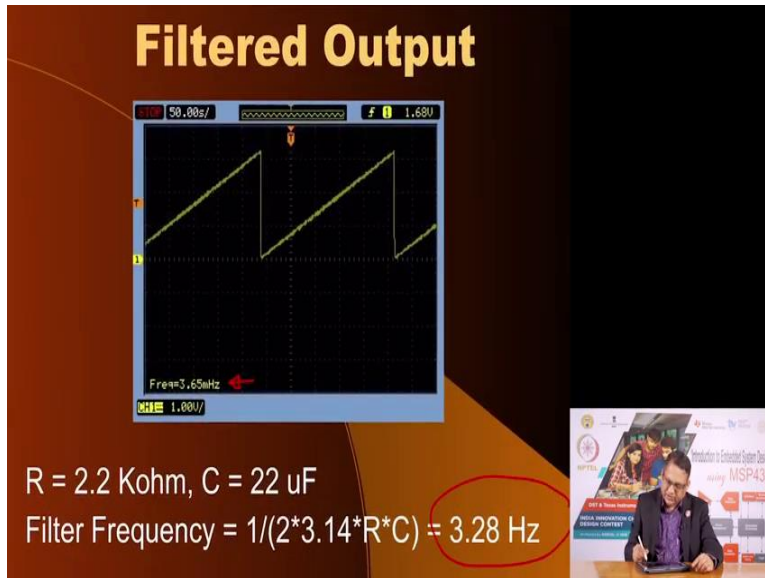


$R = 2.2 \text{ Kohm}$, $C = 22 \text{ uF}$
Filter Frequency = $1/(2 * 3.14 * R * C) = 3.28 \text{ Hz}$



Duty cycle of 24.5 %, 50 %, 75 %

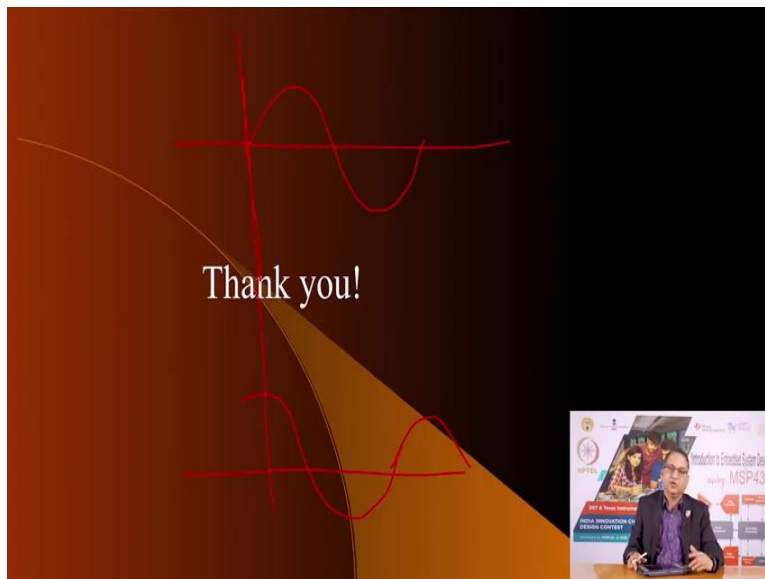
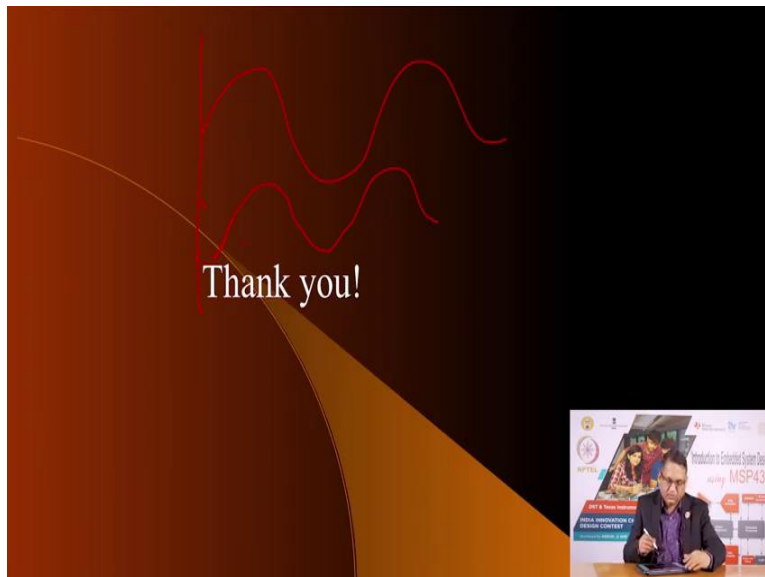




Now, you see because the 16 mega hertz frequency is used divided by 65536 will give you PWM, basic PWM frequency of just 236 hertz and this if you further by, if you are going to change the duty cycle all the way from 1 to 65536 will give you a frequency which is 234, 236 hertz divided by 65536 will give you a very very small frequency and that what you can see here. The frequency is 3.56 mille hertz and this has been obtained by putting a RC filter, low pass filter of just 3.28 hertz.

So, I strongly recommend that you go through this program and understand how a high resolution PWM based on the hardware feature has been generated using this microcontroller. I thank you for listening to this, I hope it opens your mind as to how a digital signal of the type called PWM can be used to generate all of waveforms. You can use the same feature to generate arbitrary waveforms, you can create sine waves, you can create cosine waves, you can create sine and cosine together.

(Refer Slide Time: 46:00)



Something like this, suppose I want a sine wave here and I want cosine wave say something like this. So, this is 0 and this goes let me try it again. So, let us say this is a sine wave and you wanted another waveform which is synchronized with this but something like this. So, you can easily create two waveforms which are matched locked to each other used under the control of the microcontroller you can easily do that using a PWM function.

I am going to put up some exercises for you so that you are able to generate such waveforms using the PWM function. Thank you very much.