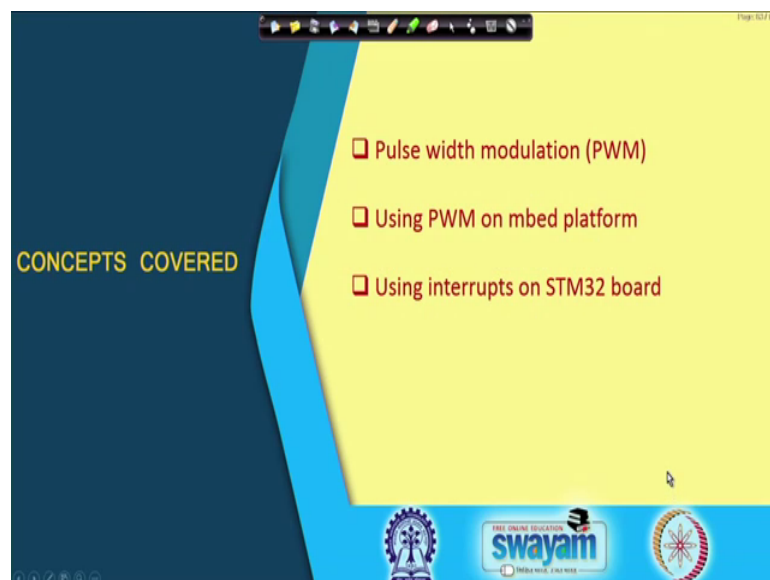


Embedded System Design with ARM
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
PWM and Interrupt on STM32F401

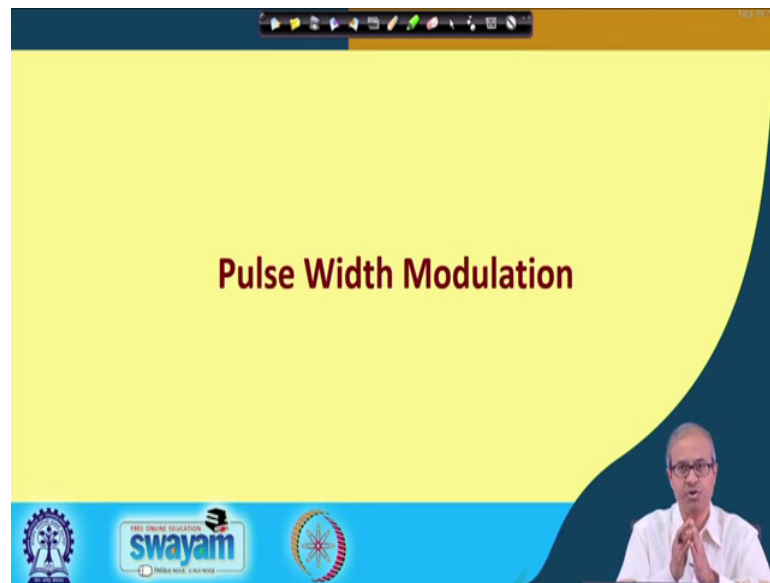
In this lecture, we shall be talking about some of the unique you can say input output features that are available in the STM 32 board which you saw in our last lecture. The title of this lecture is PWM and the Interrupts on the STM32F401 board.

(Refer Slide Time: 00:43)



So, in this lecture we shall be first talking about pulse width modulation which in short we call PWM, then how to use PWM on this specific particular board, and lastly how to use interrupts on this board.

(Refer Slide Time: 01:08)



So, first let us talk about pulse width modulation, what it is exactly. Now, pulse width modulation is a technique where you can encode the value of you can say some kind of digital value into an equivalent analog kind of a value, but not directly there is an indirect way of encoding it. We shall be understanding it as this lecture proceeds.

(Refer Slide Time: 01:40)

A presentation slide with a yellow background and a dark blue curved border on the right. The title "What is Pulse Width Modulation (PWM)?" is centered in a dark red font. Below the title is a list of bullet points. To the right of the first bullet point is a diagram of a rectangular digital waveform with "ON" and "OFF" labels and a period "T". At the bottom, there are logos for "swayam" and other educational institutions. A small video inset in the bottom right shows the same man as in the previous slide, now with his hands open and gesturing.

First let us see exactly; what is pulse width modulation. The first thing is that just using PWM we can generate some kind of rectangular digital waveform which means something like this a periodic waveform where a rectangular waveform will be going up

and down, and there will be a particular time period this will be the time period let us say T. So, this will continue.

Now, the main variability here is that where PWM helps us in bringing some kind of you can say feature which is proportional to some parameter we are trying to encode that comes from the ON-OFF behavior like when this waveform is high we say it is ON and when it is low we say it is OFF. So, the matter of interest is that for how long our waveform is ON and for how long it is OFF, right. So, this output signal will alternate between ON and OFF durations with a specific time period; of course, this ON time and OFF time will also be fixed for a particular case, alright.

And, this PWM we shall see it can be used in a variety of applications. Well, we can of course, use for communication data communication and also for control applications which we would be more interested in for embedded system design. And for communication applications you can also use PWM to encode some information, but for our lecture our scope of discussion here we do not need to know that, fine.

(Refer Slide Time: 03:46)

How it works?

- The period is normally kept constant, and the pulse width (or ON time) is varied.
- **Duty Cycle:** It is defined as the proportion of time the pulse is ON, expressed as a percentage.

$$\text{Duty Cycle} = (\text{pulse ON time}) / (\text{pulse period}) * 100\% = t_{on} / T * 100\%$$

The diagram shows a square wave with a period T and pulse width t_{on} . A dashed horizontal line indicates the average value. Handwritten annotations include a circled T and a circled t_{on} .

swayam

Now, let us see how exactly PWM works. Well, in this waveform which is being generated by PWM the time period is kept constant. So, we specify the time period. So, once we specify this time period let us see the time period is capital T this will be kept constant let us say. Now, depending on the parameter we want to encode there is something we are trying to vary. So what is, that it is the pulse width pulse width or the

on time let us call it t_{on} ; this t_{on} is something which we are varying. These are varying in proportion to some parameter that we want to encode, alright.

And in this respect we define something called the duty cycle of this rectangular waveform. The definition of duty cycle is it is the proportion of the time the pulse is on expressed as a percentage. So, actually duty cycle is defined as the total time for which your pulse is on divided by the total time period or the pulse period multiplied by hundred. Now, in this diagram the pulse on time is t_{on} divided by capital T is the time period multiplied by 100 that will give you the duty cycle.

So, this duty because t is the maximum value of t_{on} if you can have so, the value of duty cycle this will be a fraction or if you multiply it by 100 it will be a some kind of a percentage varying from 1 to 100, right.

(Refer Slide Time: 05:51)

- Whatever duty cycle a PWM has, there is an average value, as indicated by the dotted line.
 - If the ON time is small, the average value is low; if it is large, the average value is high.
 - By controlling the duty cycle, we can control the average value.

- Average value of the signal = $\frac{1}{T} \int_0^T f(t) dt = t_{on} \cdot V_H + (1 - t_{on}) \cdot V_L$
- In general, V_L is taken as 0V for ease of calculation.
 - Average value becomes $t_{on} \cdot V_H$

Now, another thing to note with respect to PWM is that well, whatever duty cycle you have you have some particular duty cycle for any kind of waveform you can define some average value. What is the average or DC value of the waveform for those of you who know Fourier transform. If you take the Fourier transform of this waveform you will get some amplitude at 0 level that will be the DC component of this waveform.

Now, let us assume this voltage is varying from 0 volts up to some maximum let us say 5 volts, 0 volts to 5 volts. So, the average value shown by this dotted line in between this

will be some value between 0 and 5. Now, this average value will very much depend on how much time the pulse is on. So, more the on period this dotted line will be moving up, but if the pulses are very narrow very small value of t_{on} then this dotted line will move down, ok. So, if the on time is small the average value will go down if it is large average value will be high.

So, essentially by controlling the duty cycle by adjusting the duty cycle we are essentially adjusting the average value of the waveform, ok. Now, if you simply take an integral and take the average you can compute the average value of the waveform over the time period T . So, integral over the time 0 to T , let us say one time period $f(t)$ let us see this waveform dt . So, so it is high for certain time t_{on} and off for the remaining time. So, if you take the average and if we just calculate you will get a value like this, where V_H is the high level of voltage V_L is the low level of voltage. So, it is high for t_{on} period of time it will be low for t_{off} which is $1 - t_{on}$ period of time, because we are dividing by T . So, actually you can write t_{on} , t_{off} here.

And, for most cases the low level of voltage is taken to be the ground voltage 0 volt. So, if V_L is 0 the second term will disappear this will become 0. So, your average value will be only t_{on} multiplied by V_H . So, by adjusting t_{on} the average value will be becoming proportional to the on period or the duty cycle indirectly, right. If the time period is constant duty cycle is proportional to t_{on} which in turn is proportional to the average value.

(Refer Slide Time: 08:50)

How to Extract the Average Value?

- The average value can be extracted from the PWM stream using a low-pass filter.
- If the PWM frequency and the values of R and C are appropriately chosen, V_{out} becomes an analog output.
 - Can be used in place of a digital-to-analog converter.

In practice, the filter is not always required.
Many physical systems have response characteristics that act like low-pass filters.

The slide features a circuit diagram of an RC network. The input voltage V_{in} is a square wave pulse. It is connected in series with a resistor R . The output voltage V_{out} is taken across a capacitor C connected in parallel to ground. The slide also includes a small inset video of a man speaking and logos for 'swayam' and 'THE ONLINE EDUCATION' at the bottom.

Now, from this PWM waveform, how can we extract the average value? Well, the simplest way is you can use a low pass filter. The simplest kind of low pass filter is an RC circuit a resistance in series and a capacitor in parallel. So, if you apply a pulse waveform. So, you will be getting an output voltage which will be the DC value; provided the RC value the RC time constant is chosen in a suitable way that only the lowest frequency of components will be passed without any attenuation the higher frequency will get attenuated. So, you will be getting approximately the DC value of the voltage in the output, right.

Now, the interesting point to notice that this kind of a circuit with PWM you can use in place of a digital to analog converter; now, a digital to analog converter which we shall be discussing later is a circuit which takes a digital input and produces a proportional analog voltage at the output. So, by controlling the digital input we can adjust the output analog voltage.

Now, PWM also provides you with a similar feature like if you change the duty cycle of this waveform that you can say is some kind of digital input you are adjusting t_{on} and you are getting a voltage in the output which is proportional to that t_{on} . So, this circuit also works something like a digital to analog converter just instead of applying a digital value, in the input I am applying a pulse width I am adjusting the pulse width of the PWM waveform, right.

Now, the point to notice that, suppose I use this circuit to control some device. Well, it can be a heater; it can be light where I am applying a pulse waveform to provide with the power supply. Most of the circuits have a built-in resistance and capacitance effect inside it. There will be some electronic circuit, there is input impedance and there is often a built-in low pass filter in the input stage. So, this low pass filter often is not required for most cases. This PWM pulse waveform you can directly apply to the device you want to control. And the low pass filter in the input stage will automatically extract the average value and based on the average value whatever you want will be done, right.

(Refer Slide Time: 11:39)

Some Typical Applications

1. Control of DC motor.
 - The voltage supplied to the motor is proportional to the duty cycle.
2. Controlling the brightness of LED.
 - The duty cycle of the voltage source determines the brightness.
3. Control the temperature (heater).
 - Switch ON and OFF the heater with an appropriate duty cycle.
4. Many more ...

So, some typical applications of PWM: well, here I am listing only a few you can think of other applications, as well there are many applications you can think of. Suppose you are trying to control a DC motor. So, for controlling a DC motor there is often a control signal with the help of the control signal you can adjust the power supply you are applying to the motor.

So, the power supply that is applied if the PWM waveform you are applying directly then the average value will be the equivalent power supply. So, by adjusting the duty cycle you are effectively adjusting the power supply thereby you are adjusting the speed of the motor, ok. This is one application you think of. Suppose any light well I am considering a led light emitting diode it can be any other lamp. So, if I apply a PWM

waveform as the power supply of that lamp. So, I can adjust the brightness level of the lamp this is another application.

Think of a gadget like heater or even you can think of microwave oven. So, when you adjust the power of the microwave oven what actually happens? There as there is a PWM mechanism which will be switching the microwave mechanism ON and OFF and the duty cycle is adjusted. Same is the case of an automatic heater control system. You turn ON and OFF the heater, the time you are turning ON the heater that will proportional to the duty cycle of the PWM waveform. And there are many other similar applications you can think of.

(Refer Slide Time: 13:30)

PWM on the STM32F401

- The *PwmOut interface* is used to control the frequency and duty cycle of a digital pulse train.
- The *Arduino Interface* supports up to **6 PWM outputs**, although these *PwmOut ports* share the same period timer.

Now, I mean if you revisit that board we saw in the last lecture the STM32F401 board here if you look at you note the Arduino connector because the same connections are available on the detail connectors also. But, if you look at the Arduino connectors you will see that there are several PWM pins which are mentioned. There are six such PWM pins which are mentioned. So, there is a standard PWM out interface that we shall be using some kind of software development platform called embed this would shall be seeing later under embed this PWM out is a standard library which is provided it helps us in developing applications that use a PWM.

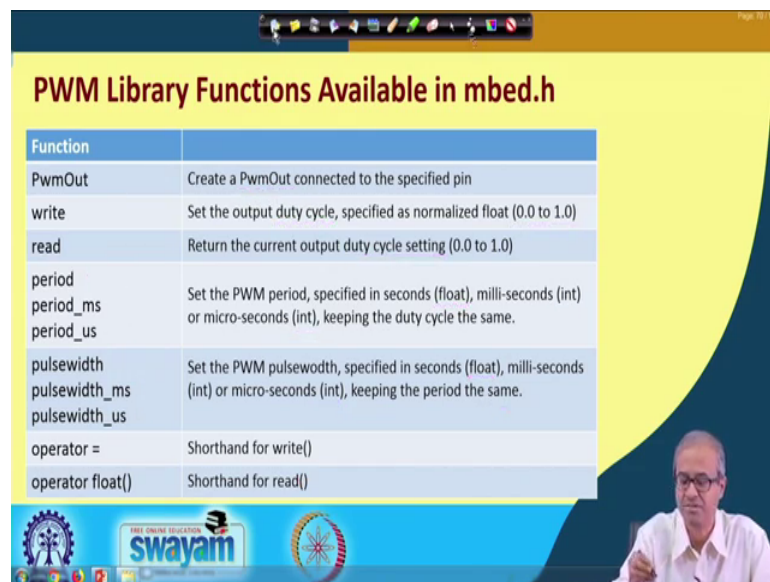
Now, this interface has some functions which can be called to adjust the duty cycle of the PWM waveform that has been generated, ok. Now, this Arduino interface as I have just

now shown Arduino interface supports up to 6 PWM outputs, but if you have access to these detail pins there are many more. So, you can have more than six PWM output pins also.

Essentially this PWM signals ON and OFF these are generated using some timers. Well, you can put the output to one load the timer with a value and with a clock the timer will be down counting you wait till it goes 0. Like for example, this ON period can represent a count value of 10 and this OFF period can represent a count value of 18 let us say. So, I will be loading the counter or the timer with that appropriate value and let it go on counting down till it reaches 0 and as soon as it reaches 0. I change the value of the signal and this repeats 10, 18, 10, 18, 10, 18.

And, when I want to change the duty cycle the total I will keep 28, but only I will be changing the first part. Suppose the first part I make 5 the second part will automatically get adjusted to 23. This sum will be 28 because period will remain same. So, these adjustments are done automatically internally as I adjust the duty cycle.

(Refer Slide Time: 16:19)



Function	
PwmOut	Create a PwmOut connected to the specified pin
write	Set the output duty cycle, specified as normalized float (0.0 to 1.0)
read	Return the current output duty cycle setting (0.0 to 1.0)
period	
period_ms	Set the PWM period, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the duty cycle the same.
period_us	
pulsewidth	
pulsewidth_ms	Set the PWM pulsewidth, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the period the same.
pulsewidth_us	
operator =	Shorthand for write()
operator float()	Shorthand for read()

So, let us look at the functions that are available as part of the PwmOut library. So, I am just talking very briefly about it this PwmOut is you can say the class this is an object oriented concept. This is the class which will be creating a PwmOut interface connected with a specified pin because on the Arduino interface will be seeing there are many pins which are called d, d 1, d 2, d 3, d 4, d 5. So, whatever PWM pin you will see it has

written PWM slash d 3. So, here you can specify that d 3 that you are trying to use the d 3 pin as your PwmOut, ok. I will take some example.

And, there are some functions called write and read. Using the write function you can specify the duty cycle of the waveform. Duty cycle is specified as a fraction from 0 to 1. So, means it is not multiplied by 100 just on period divided by the total period, it is specified as a fraction. So, if you write it as 0.5. So, it will be equal ON period and equal OFF period, ok. They will be symmetrical.

Similarly, read: well, you can use read to read the current value of the duty cycle setting. Suppose, in a program you do not remember; what was the duty cycle you are said, you can call this function read and know that well my duty cycle was 0.5, let us say, ok. And, we can set the time period there are three functions available period, period underscore ms, period underscore us. The first one specifies the time period in seconds, second one specifies in milliseconds, third one in microseconds. The only point to notice that when you specify it in seconds, your parameter will be a floating point number, you can write 2.5. So, it will be 2.5 seconds. But, when you specified milliseconds and microseconds the parameters will be integer. So, no fractional parts are allowed you should remember this.

So, you can set your period you can set by write the duty cycle. So, your PWM will be generated, but instead of specifying the duty cycle there is another way of having the control yourself with yourself you can specify the pulse width exactly what is the on time t on. So, here also you can specify in seconds, milliseconds or microseconds. Again in seconds it will be floating point, million microseconds there will be integers.

Now, write there is a shortcut if you straightaway give the name of that object and equal to something. So, that is equivalent to write you can make a shortcut and just name of the object if we use in an expression that is equivalent to read, these are shortcuts. So, here we will take some examples how to use them.

(Refer Slide Time: 19:39)

The slide displays two code snippets for generating a 100 Hz pulse with a 50% duty cycle. The first snippet uses `PwmOut PWM1(p21);` and `PWM/D9` for pin specification, while the second uses `PWM1 = 0.5;` as a shortcut. A waveform diagram shows a square wave with a period of 100 μs. A small diagram on the right shows a pin header with pins numbered 1 to 5.

```
#include "mbed.h"
PwmOut PWM1(p21); // Pin 21
                // (PWM/D9)

int main() {
    PWM1.period(0.01);
    PWM1.write(0.5);
}
```

```
#include "mbed.h"
PwmOut PWM1(p21); // Pin 21
                //
                (PWM/D9)

int main() {
    PWM1.period(0.01);
    PWM1 = 0.5;
}
```

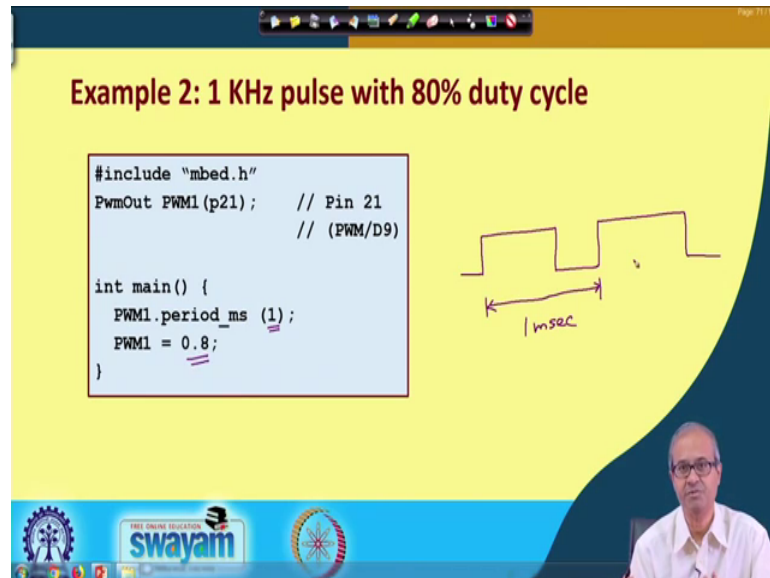
This is a simple example. Suppose, we want to generate a 100 hertz pulse with 50 percent duty cycle. So, what is the meaning of 100 hertz? Suppose, this is my pulse which has been generated, this is my time period. 100 hertz means 1 by 100 second will be the time period. 1 by 100 is how much? 0.01 second. So, you see the first alternative. So, I have I am showing here two alternate codes. The first alternate says I have I have to include this mbed dot h header because many of the functions I am using they are all included in this mbed header. This PwmOut is the PWM object I told you. So, we will have to create an instance of this class you create an object you call it PWM 1 this is a name you are giving and p 21 is the name of the pin you are giving; p 21 is the pin name.

So, you know p 21 is with respect to the original detailed connector where you see there will be one connector on either sides. So, the convention is the pins are numbered like this 1, 2, 3, 4, 5, 6 like this that pin number 21 is a PWM port or if you use the Arduino connector it is written PWM slash D 3. So, instead of p 21 you can also write D 3 straight away or D 9 sorry p 21 is D 9, you can write D 9 here right same thing

And, in the main function this is a c code I am writing. So, I am calling the two functions period and write PWM dot period is 0.01 which means 100 hertz, 1 by 100 seconds and write I am specifying the duty cycle 0.5. The second code is the same one where instead of calling write I am using the shortcut. If I write straight away PWM one equal to 0.5

this means actually we are writing the duty cycle 0.5, this is just a shortcut. So, this is how you can generate a PWM waveform.

(Refer Slide Time: 22:14)



The slide displays the following code in a light blue box:

```
#include "mbed.h"
PwmOut PWM1(p21); // Pin 21
// (PWM/D9)

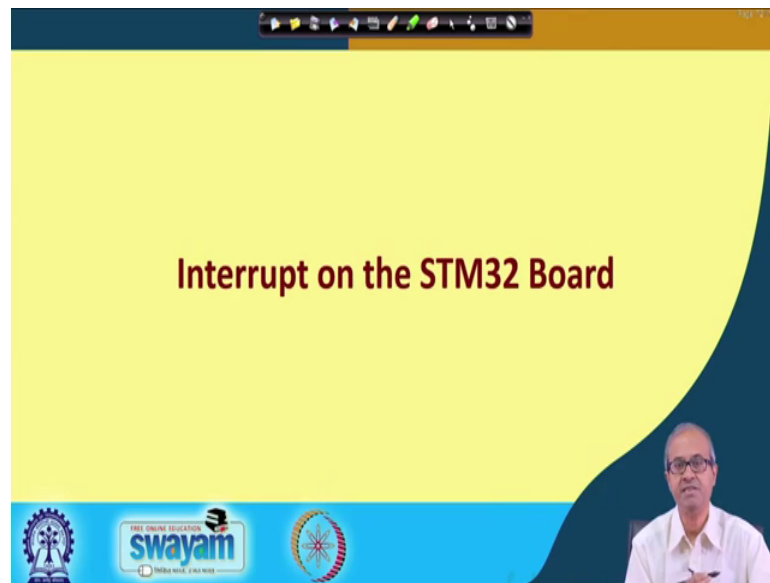
int main() {
    PWM1.period_ms (1);
    PWM1 = 0.8;
}
```

To the right of the code is a hand-drawn waveform diagram of a square wave. A horizontal double-headed arrow below the first cycle is labeled '1 msec', indicating the period. The pulse width is approximately 80% of the period.

At the bottom of the slide, there is a blue banner with the 'swayam' logo and other educational icons. A small video inset in the bottom right corner shows a man speaking.

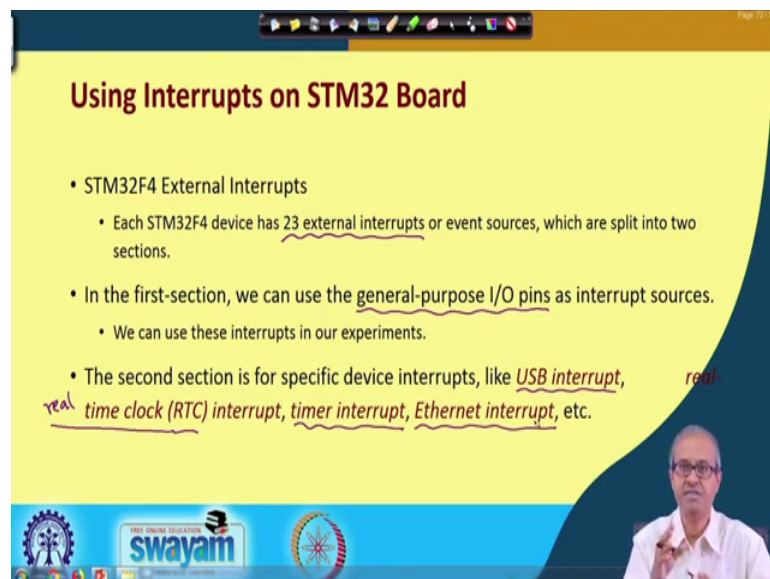
Just another example. This is a one kilohertz pulse with 80 percent duty cycle. So, here the waveform will look something like this. First it is 1 kilohertz; 1 kilohertz means the time period will be 1 by 1 k which is 1 milliseconds and 80 percent will be the duty cycle. So, in the same way you specify a PWM pin, you specify the period in milliseconds I am using your ms 1. So, it is 1 milliseconds and PWM equal to shortcut I am using duty cycle 0.8. So, it will be generating a waveform like this. So, this we shall be seeing in the demonstration how we can use this PWM control to generate various kinds of waveforms.

(Refer Slide Time: 23:12)



Now, let us see; what are the kinds of interrupt facilities that are available on the STM 32 board, because you know that in many applications wherever you are interfacing some devices there will be some devices which can be generating some interrupt signals from outside. So, I mentioned that in the ARM processors there are many interrupting source; in fact, any of the port lines you can use as an interrupt input, right.

(Refer Slide Time: 23:39)



So, let us look into a little more detail.

For the STM 32F4 general family of boards; talking about external interrupts there are 23 external interrupts that are supported, right. Among this 23 external interrupts, these are split into two sections. In the first section we have the general purpose IO pins like the port A, port B, port C I talked about. I will show it in the next slide again in the first part you have the general purpose IO pins which can be used as interrupt inputs.

So, in the actual experiments you shall be using these pins only to connect interrupting sources, but there is another section another part where some specific devices are generating interrupts. Like your USB interface can generate an interrupt, your real time clock which is there; real time clock which maintains the time of day clock that can generate an interrupt, some timer which are using that can generate an interrupt. If you are having a network interface like Ethernet that interface can also generate an interrupt. So, these are device specific interrupts which fall in the second category, but first category is more general purpose. So, in our experiments we shall be using the first category.

(Refer Slide Time: 25:17)

The slide is titled "General-Purpose I/O (GPIO) Pin Interrupt Lines". It contains the following text:

- The 16 interrupt lines are referred to as Line0 to Line15.
- Three 16-bit GPIO ports: PA, PB and PC.
- Line0 is connected to PA0, PB0, and PC0.
- Line1 is connected to PA1, PB1, and PC1.
- LineZ is connected to PAZ, PBZ, and PCZ, and so on.
- We cannot use PAX, PBX, PCX as interrupt sources together for the same value of x.

The slide also features a presenter in the bottom right corner and logos for "swayam" and "MHRD" at the bottom.

Now, in the first category these are called general purpose IO or GPIO pin interrupt lines. Now, under this category broadly speaking we say that there are 16 interrupt lines. These interrupt lines are referred to as Line0 up to Line15. Now, you should remember that in this ARM board this familiar board that we are using, there are three general purpose IO

ports each of them are 16-bits these are called port A, port B and port C. These are number from port A0 to port A15 port B0 to 15 port C0 to 15, ok.

Now, the point to notice that this 16 interrupt lines they are not independently connected to all the port lines rather Line0 is connected to PA0, PB0 and also PC0. So, whenever when you are trying to use Line0, you can connect your interrupt input to either PA0 or PB0 or PC0; this also means you cannot use PA0 and PB0 as two separate interrupt inputs because they actually mean the same line with respect to interrupt.

Similarly, Line1 is connected to PA1, PB1, PC1. Line2 is connected to Line2 connected to PA2, PB2, PC2, and so on, right like this the things are getting connected. So, as I said you cannot use PAX, PBx, PCx for the same value of x as interrupt input because PAX, PBx, PCx all refer to the same interrupt input Line x, ok. This is something you have to remember. There are 16 interrupting sources you can use, at a time, but only 48 port inputs you can use for the connection.

(Refer Slide Time: 27:28)

Function	Description
InterruptIn ✓	Create an InterruptIn connected to the <u>specified pin</u>
int read() ✓	Read the input, represented as 0 or 1 (int)
operator int()	Shorthand for read()
void <u>rise</u> (* func)	Attach a function to call when a rising edge occurs on the input
void enable_irq() ✓	Enable IRQ
void disable_irq() ✓	Disable IRQ

Now, for the interrupt the library functions that are available to us are as follows. Well, interruptin is the class you create an object of type interrupt in. So, here also you specify which pin you are trying to connect to just like PWM. And these are the functions that are available. Read means you can read the status of the interrupt input whether it is 0 and 1 currently, if you call read you can read the input it will come as either 0 and 1 of type integer.

Now, if you just use the name of the object you are creating that is a shortcut for read that that is also you can use and there is a function called rise, whereas a parameter you give a pointer to a function; this means if there is a rising edge on the interrupt input automatically that function will be called. So, essentially that function is like your interrupt service subroutine or the interrupt handler. So, when you are writing a C program you can also specify for a particular interrupt input this will be your interrupt handler. And, you can enable and disable the interrupts by calling the functions enable irq, disable irq, ok. This will enable and disable interrupts if you want.

(Refer Slide Time: 29:04)

The slide displays the following C code for toggling an LED:

```
#include "mbed.h"
InterruptIn switch (D3);
DigitalOut led (D2);

void toggle() { // Interrupt handler
    led = !led;
}

int main() {
    switch.rise (&toggle); // Install interrupt handler
    while (1) {
    }
}
```

The circuit diagram shows a switch connected to pin D3. One terminal of the switch is connected to a +5V supply through a resistor R. The other terminal is connected to ground. Pin D2 is connected to an LED through a resistor R, and the other terminal of the LED is connected to a +5V supply.

- Switch connected to the interrupt input D3.
- A LED connected to output pin D2.

Let us show a simple example. So, on the left I am showing the code of a simple example using interrupts. Now, the instance that I am saying is that suppose this is your microcontroller board. So, on the line D3 let us say on line D3 I have connected a switch. So, how I have connected a switch? Typically we connect, switch is like this, we connect a resistance, then you connect a switch like this, on the other side we connect 5 volts and here we have resistance. So, if this switch is open high voltage or logic 1 will come to D3, if it is pressed it will be shorted to ground 0 will come.

And, on the other side in D2 we are connecting a light emitting diode. So, how we are connecting let us say we are connecting it like this. We are connecting a resistance, then you are connecting a light emitting diode, then we are connecting it to 5 volts. So, whenever D2 is 0, there will be a current flowing and the led will glow if D2 is 1, there

will be no current flowing because both sides will be a same voltage led will not glow. This is how I have made the connections and what I want is that every time this switch is placed pressed the glowing status of the LED should change; that means, on off on off like that with every press of the switch the status of the LED should change.

Let us say how you have written the program. First as usual we have included this mbed dot h, then we have created an object of type interrupting we have given the name switch. So, you can give any name and we have connected it to D3. D3 is the pin number to which you have connected the interrupt input and this LED is a digital output pin. There is another object that we use for that it is called DigitalOut for all digital output port lines we use this object of type DigitalOut and LED is the name of this connection and here I have used D2 port number D2.

And, this is my main function. Let us see what we have done in the main function this one. Here we have first made a call to this rise, switch is the object you have created. switch dot rise means I am calling this function in connection with the object switch and I have given ampersand toggle means address of toggle; toggle is the name of a function which means I told you whenever there is a rising edge on that interrupt input automatically the function toggle will get called and after that there is a dummy loop while one this is an infinite loop your main program is waiting.

And, every time there is a switch press means an interrupt is coming what will happen? Toggle will just do led equal to not LED. If LED was 0, it will become 1, if it is 1 it will become 0. So, for every switch press toggle will be called and the status will change ok, just like this will go on.

So, with this we come to the end of this lecture. Here, we have tried to tell you about the PWM and the interrupt interfaces that are available on the STM 32 board and how to use them.

Thank you.