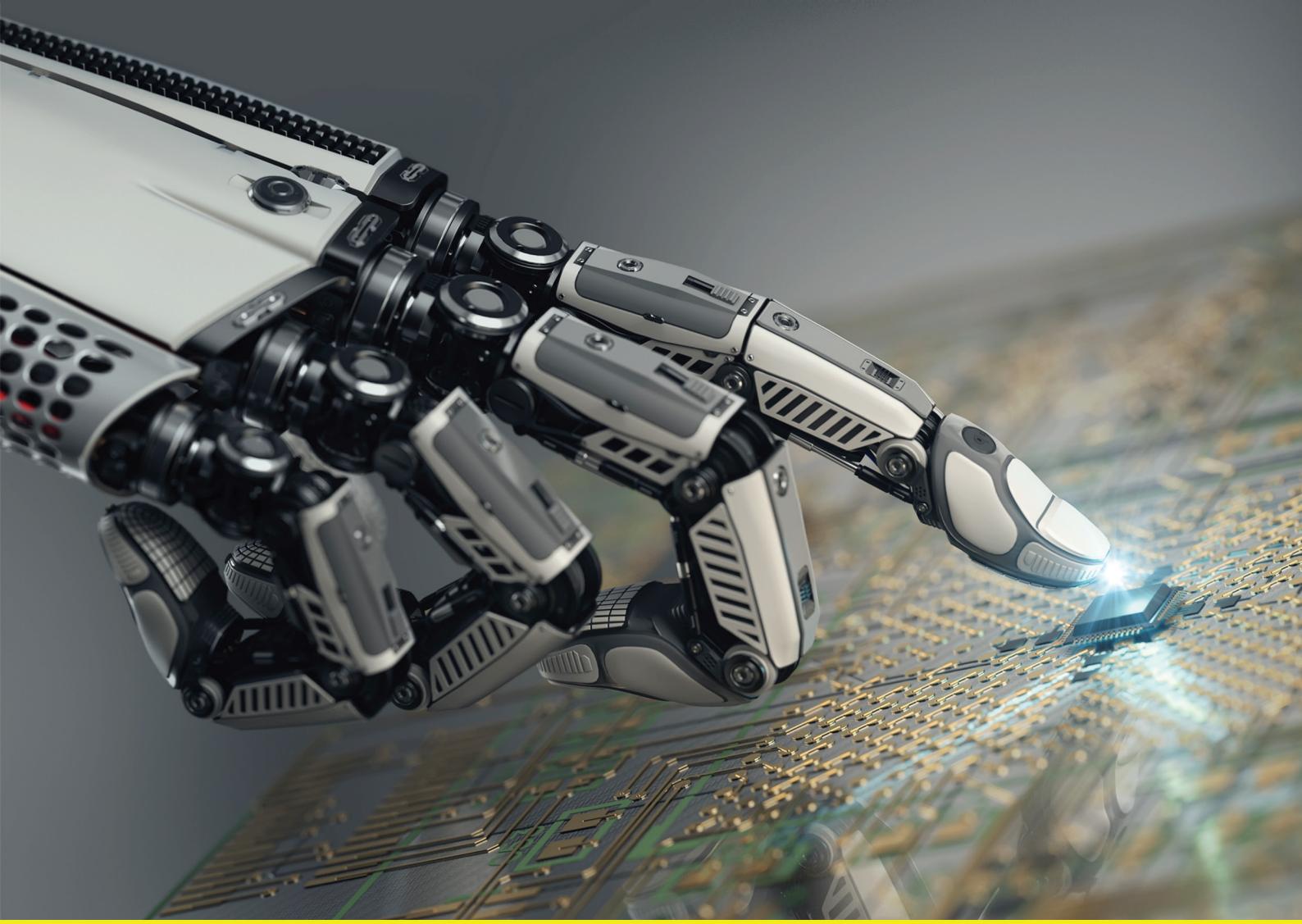


EMBEDDED SYSTEM DESIGN WITH ARM



Prof. Indranil Sengupta
Prof. Kamalika Dutta
Computer Science and Engineering
IIT Kharagpur



INDEX

S. No	Topic	Page No.
	<i>Week 1</i>	
1	Introduction To Embedded Systems	1
2	Design Considerations Of Embedded Systems	14
3	Microprocessors And Microcontrollers	27
4	Architecture of ARM Microcontroller (Part 1)	41
5	Architecture of ARM Microcontroller (Part 2)	53
6	Architecture of ARM Microcontroller (Part 3)	67
	<i>Week 2</i>	
7	ARM Instruction Set (Part 1)	82
8	ARM Instruction Set (Part 2)	94
9	ARM Instruction Set (Part 3)	104
10	About The STM32F401 Nucleo Board	115
11	PWM And Interrupt On STM32F401	128
	<i>Week 3</i>	
12	Digital to Analog Conversion	143
13	Analog to Digital Conversion (Part 1)	161
14	Analog to Digital Conversion (Part 2)	175
15	Output Devices, Sensors and Actuators (Part 1)	187
16	Output Devices, Sensors and Actuators (Part 2)	203
17	Output Devices, Sensors and Actuators (Part 3)	214
	<i>Week 4</i>	
18	Microcontroller Development Boards	224
19	Mbed C Programming Environment	235
20	Interfacing With STM32F401 Board	248
21	Interfacing With Arduino Uno	262
22	Interfacing 7-Segment LED and LCD Displays (Part 1)	278
23	Interfacing 7-segment LED and LCD displays (part 2)	290

Week 5

24	Serial Port Terminal Application (Coolterm)	300
25	Experiment With Temperature Sensor	310
26	Experiment With LDR Light Sensor (Part 1)	320
27	Experiment With LDR Light Sensor (Part 2)	329
28	Experiment With Speaker	337
29	Experiment With Microphone	349

Week 6

30	Design of Control System	358
31	Experiments With Relay	369
32	Experiments on Speed Control of DC Motor	381
33	Experiment With Multiple Sensors And Relay	398

Week 7

34	Introduction to Internet of Things	406
35	GSM And Bluetooth	419
36	Design of a Home Automation System	430
37	Design of a Simple Alarm System Using Touch Sensor	443

Week 8

38	Accelerometer	451
39	Experiment Using Accelerometer	456
40	Experiment Using Bluetooth	465
41	Experiment With Gas Sensor	487
42	Summarization of the Course	495

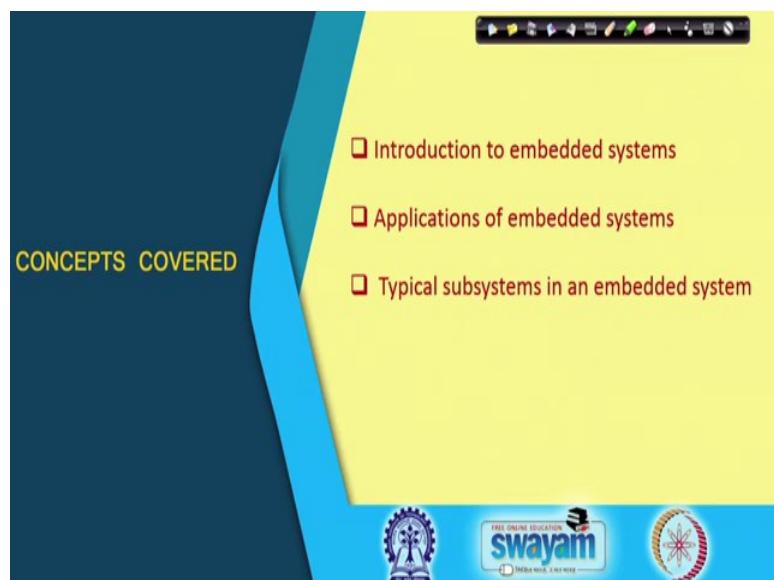
Embedded System Design with ARM
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 01
Introduction to Embedded Systems

Let me welcome you to this course that will be conducted jointly by myself and Dr. Kamalika Datta. In this course we shall be talking about various aspects of Embedded System Design, in particular we shall be emphasizing on some hands-on design examples. As the vehicle of demonstration we shall be primarily working with two different embedded system platforms; one is based on an ARM based board and the other one will be a standard Arduino Uno based board.

Now before we proceed with the actual experimentation, it is always good to have some theoretical background and some motivation behind why we need an embedded system, what are the main characteristics of an embedded system and what are the design alternatives. So, in this context the first lecture of this course, is titled Introduction to Embedded Systems.

(Refer Slide Time: 01:40)



Here we shall be primarily talking about embedded systems, what it is basically and some typical applications, and inside an embedded systems what kind of typical

subsystems we normally get to see. These are a few things that we shall be discussing as part of this course.

(Refer Slide Time: 02:05)

Introduction

- We have been brought up in the age of computing.
 - Computers are everywhere (some we see, some we do not see).
- Types of computers we are familiar with:
 - Desktops and Laptops
 - Servers
 - Mobile phones
- But there's another type of computing system that is often hidden.
 - Far more common and pervasive...
 - Hidden in the environment.

Embedded Systems

The first thing I want to emphasize is that, we have been very lucky that we have been born and brought up in an age of computing. Since our birth many of you have seen computer systems around you, but of course, the scenario was not the same maybe 4 or 5 decades back.

There has been a fantastic proliferation of electronics, computing systems, low power devices, and there are various kinds of advancements that have taken place over the years. The first thing I want to emphasize is that computer systems are everywhere today. If you look around you; well I am not talking about in a technical environment, not in your laboratory, not in your college well even in your residence when you wake up from your sleep.

If you look around you, you will find several instances of some computational devices that will be sitting somewhere. This is the kind of environment where we have all grown up. Traditionally speaking, we have become familiar with computers that are either desktops, laptops, servers etc. that are more conventional in some sense. Desktop computers are typically machines where you have learnt your computing on. Today laptops have become very much affordable, most of us own our personal laptops. So,

laptops are replacing the role that desktops used to have earlier. Talking about mobile phones this is a very underestimated computing device.

Whatever is inside a mobile phone today, if you think of the computational capability of the computers that existed 30 to 35 years back, believe me the processes that are inside a mobile phone are of the tune of 1 million times more powerful as compared to what you used to see in the large computer systems in those days. But mobile phones we use for some specific applications, for its use as a phone, sending messages. And of course, for recreational purposes we can watch video, we can browse internet, and there are so many new applications that you can run on mobile phones today.

But what I want to say is that, there is another kind of computing system that is often hidden from us. Hidden means we cannot see those computing systems. Like we know desktop can compute, a laptop can compute, a mobile phone can also compute in some sense. But if you look at a refrigerator, if we look at our air conditioning machine, do they look like a computing machine? No. But inside those machines there is some computing brain hidden, that is what I am referring to as this hidden thing. These are far more common in our daily life and pervasive, as they are everywhere, and the point I want to make is that, they are hidden in the environment for which they were created.

Now, such systems are traditionally referred to as embedded systems. We mean these are computing systems, but they are embedded inside the environment. By the term environment we mean the surroundings or actually the scenario for which the system was designed. Let me take the example of an air conditioning machine again. An AC machine is supposed to be housed inside a room normally. Now this computing system that is sitting inside an AC machine, for that computing system the AC machine is the environment. It does not interact with anybody outside that AC machine.

It is responsible for controlling the AC machine, it is responsible for responding to whatever you are pressing on your remote control, you are given giving some commands to the ac machine and so on. These are the kind of commands that those computing machines inside are responsible to respond to.

(Refer Slide Time: 07:47)

What are Embedded Systems?

- Computers are embedded within other systems:
 - What is "other systems"? – Hard to define.
 - Any computing system other than desktop / laptop server.
 - Typical examples:
 - Washing machine, refrigerator, camera, vehicles, airplane, missile, printer.
 - Processors are often very simple and inexpensive (depending on application of course).
 - Billions of embedded system units produced yearly, versus millions of desktop units.

Talking about embedded systems again, what are these embedded systems? I already gave an example. So, now it must be somewhat clear to you. Embedded systems are computers they are embedded within other systems. Now on the right side I have given some pictures. You see these examples, here you see a washing machine, this is a refrigerator, this is a laser printer, this is a digital camera, this is an automobile and here you can see a missile that is flying through the sky. These are all examples of embedded systems, they are fairly powerful computing systems or devices inside them of course, the power of these computing systems depend on the specific application.

For some applications you do not need too much computation power, but for a system like missile lot of computations need to take place in real time. As the missile flows in its trajectory, there is continuous feedback from the environment and there are some corrective actions that need to be taken such that the missile can follow the correct path and hit the target in a precise way.

So, it depends on the environment. Now this "other systems" can refer to anything, it is very hard to define in a very specific way. But broadly speaking you can say that these other systems are any computing system, which are not our conventional computers like desktop, laptop, servers etc.

Any computing system that is not one of these, they are sitting somewhere and working with the environment, they can be classified as embedded systems. Some typical

examples are given here. And again just as I said depending on the application, processor can be very simple and inexpensive. Because there are many devices, let me take an example again. Suppose you have gone to the market to buy a microwave oven, they are available for prices ranging for Rs. 3000 and more.

They are pretty cheap in comparison. There are embedded processors inside micro ovens also. Now if I say that I need a very powerful Pentium processor to be sitting inside a microwave oven, the price of that Pentium processor itself will be 5000 rupees including all accessories and peripherals, then this will be economically not viable. They have to be very cheap depending on the applications.

Today a very large number of embedded systems are being used all over the world in various applications, billions of them versus millions of conventional computing units. So, the number of such embedded computing devices far outnumber the number of conventional computing devices today.

(Refer Slide Time: 12:13)

Common Features of Embedded Systems

- They are special-purpose or single-functioned.
 - Executes a single program, possibly with inputs from the environment.
 - Imagine a microwave oven, a washing machine, an AC machine, etc.
- Tight constraints on cost, energy, form factor, etc.
 - Low cost, low power, small size, relatively fast.
- They must react to events in real-time.
 - Responds to inputs from the system's environment.
 - Must compute certain results in real-time without delay.
 - The delay that can be tolerated depends on the application.

The slide also features images of a microwave oven, a washing machine, and an air conditioning unit.

At the bottom, there are logos for the Indian Space Research Organisation (ISRO) and the Swayam platform, along with a photo of a speaker.

There are some common features that are present in most embedded systems, let us look at some of them. First thing is that they are special purpose or single functional. Like an air conditioning machine, there is a processor inside, the sole purpose of that processor is to ensure that the ac machine is working properly, and nothing else. So, it is not trying to help you in calculation in doing some maths, and so on.

So, it is very special purpose in that sense. It typically executes a single program related to the application for which it was built, and it can take inputs from the environment. You think of an ac machine; the inputs from the environment means, well the user can send some commands via the remote control. The ac machine itself can have some sensors inside it, can sense what is the current room temperature, what is the current humidity level. Accordingly it can activate the various subsystems inside in a very optimum way.

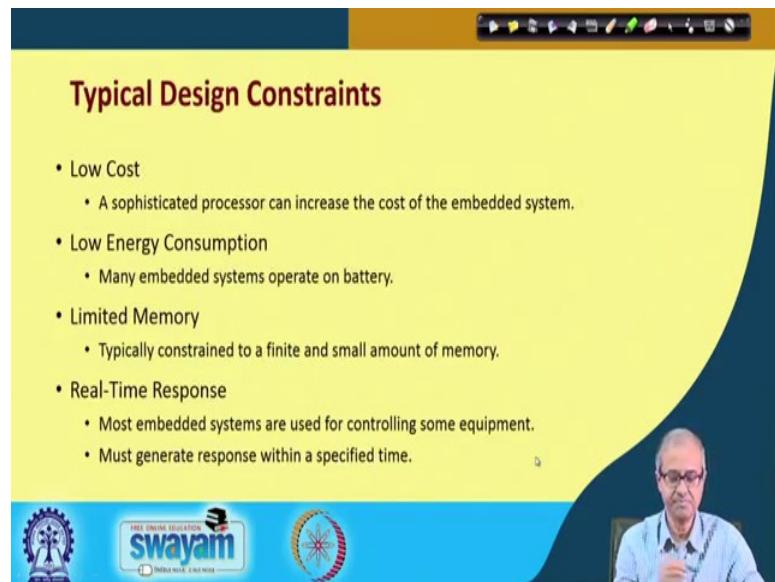
You think of a refrigerator there normally there are some control panels. You can set the temperatures and other factors there. It depends again on the application, through the environment you can send some commands to the processor, and the processor will respond accordingly. Some examples I have given, like microwave oven, washing machine, ac machine.

And for most of these embedded systems there are very tight constraints on cost, energy and also form factor. The processor has to be low cost otherwise the system will not be economically viable. The processor must not consume too much energy from the power source or from the battery wherever it is working. Because the processor is hidden from you and it is expected that the user should not worry too much about energy. Form factor; you think of an ac machine nowadays that are very sleek, now you cannot afford to have a computer inside which is very bulky and big because that will make your system also bulky and big.

So, depending again on the system, your computing system must be made very compact and should fit inside without any appreciable increase in area. So, low cost, low power, small size, relatively fast these are some of the characteristics, and another important thing is that it should have real time response. You think of ac machine, suddenly due to some reason the temperature of the room has gone up. It must turn on the compressor at high power to bring down the temperature. So, it must respond to these external inputs in real time, it cannot say that well I shall respond to this after 10 minutes.

Real time means, whenever an input comes, within some specified time the system should respond to that input. Most embedded systems respond in real time. They take inputs from the system environment and should carry out the computations; the tolerable delay varies from application to application. So, these are some of the common features.

(Refer Slide Time: 17:19)



Typical Design Constraints

- Low Cost
 - A sophisticated processor can increase the cost of the embedded system.
- Low Energy Consumption
 - Many embedded systems operate on battery.
- Limited Memory
 - Typically constrained to a finite and small amount of memory.
- Real-Time Response
 - Most embedded systems are used for controlling some equipment.
 - Must generate response within a specified time.

Now, there are some design constraints as well; some of these are pretty obvious. The processor should be a low cost thing. You should not make it too expensive because that will also make your system inside which it is embedded, it will make that also quite expensive. You cannot afford to use a very sophisticated processor inside such a machine; like inside a microwave you cannot afford to put a sophisticated processor like the Pentium, because the cost of the Pentium itself is pretty high.

Low energy consumption is another property which is pretty common. Well even for machines which operate from electric power, like a refrigerator if you put it on and go away, it is expected that it will consume very low power. And secondly, there are many gadgets which also operate on battery. For them obviously, energy consumption is a big issue. And because these processors are very simple, the amount of memory they have is also pretty small. So, whatever program you write, whatever computation you do they must fit inside that limited memory space. I mentioned many applications demand real time response; within a specified time, the system should respond to the inputs.

(Refer Slide Time: 19:09)

How to define an Embedded System?

- It is a microcontroller-based system that is designed to control a function or range of functions, and is not meant to be programmed by the end user.
 - The user may make choices concerning the functionality but cannot change them.
 - The user cannot make modifications to the software.
 - Can you *"program"* your washing machine or refrigerator or car?
 - Not today ... but not very sure of the near future.

Block Diagram of an Embedded System:

```
graph TD; UI[User Interfaces] --> EC[Embedded Computer]; EC[Embedded Computer] --- SW[Software]; EC[Embedded Computer] --- HW[Hardware]; UI --> EC; EC --> IO[Input/Output]; EC --> LO[Link to Other Systems];
```

The diagram illustrates the architecture of an embedded system. It consists of an **Embedded Computer** block which contains **Software** and **Hardware**. This central block receives **Input Interfaces** and provides **Output Interfaces**. Additionally, it has a **Link to Other Systems**.

Now, how can we define an embedded system based on whatever we talked about so far? Well embedded system can be loosely defined as a microcontroller based system. This is true for most of the embedded systems. The processor that is inside has to be low cost it has to be low power, it has to be small in size. There is some kind of processor called microcontroller that we shall be talking about throughout this course, they satisfy all these criteria.

Most of the embedded systems have microcontrollers inside them, they are like a complete computer in a single chip. Whatever you need, processor, memory, IO everything is inside a single chip and therefore, it is very low cost. So, it should be a microcontroller based system and not general purpose. It should control a function or a range of limited set of functions. And another point is that, this is not meant to be programmed by the user. Like a laptop or a desktop you can write a program, well you can write a program to compute the factorial of a number, to find the sum of n numbers etc. But whenever you think of an embedded system inside a device like your ac machine. You normally do not use it for your day to day computation, you cannot program it yourself. It comes to you factory programmed, you cannot modify the program, it is a fixed program system -- you are only using that program.

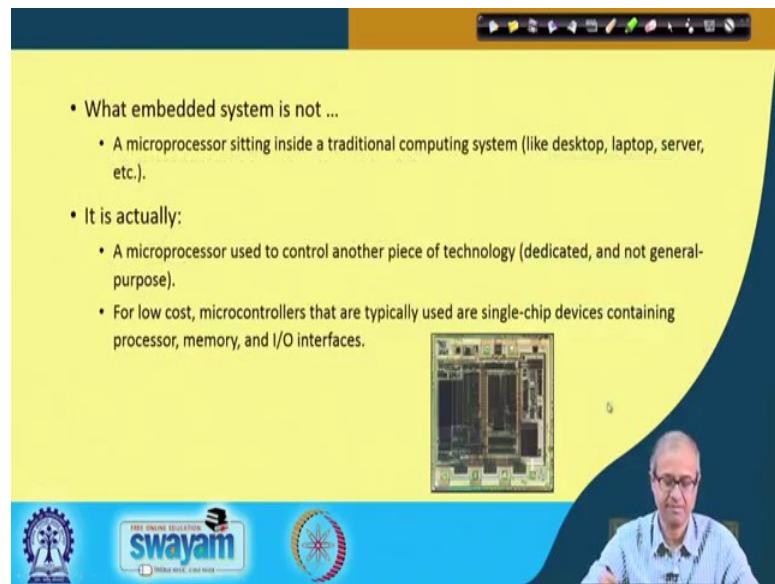
The user can give inputs for example, through the remote controls, but cannot change the functionality. You cannot change an ac machine to something else like a room heater for

example. The user normally cannot make any modifications to the software, but of course, nowadays systems are becoming more sophisticated; you think of a set top box that you use with your TV sets. Set top box are also embedded systems, there are quite powerful processors inside them. Now sometimes you may have noticed that while watching some programs or when you are switching it on, it says downloading updates. Well the software that is running inside the set top box also gets periodically updated by the service provider. That facility is provided inside that set top box. Just like your standard operating systems like windows that frequently needs some updates to be installed.

That means, you need some modifications to your existing software to make it better in some sense. So, normally you cannot program your washing machine or refrigerator or a car, but maybe tomorrow with the next generation embedded systems coming, you may be able to program them also. Like when you drive a vehicle you may program it and customize it according to your need, but as of today the systems come with very limited choice to the users; maybe a few things user can specify, but not all.

This is a schematic diagram of an embedded system. The embedded computer is sitting in the middle, which has some hardware, the microcontroller and some program which is running on its software. It takes inputs through some sensors, and it can control something through some outputs, and there are typically some user interfaces like some switches, some small keyboards, like some touch sensors maybe a remote control, these are all user interfaces and it can also have some links to other subsystems. This is a general schematic of an embedded system.

(Refer Slide Time: 24:09)



- What embedded system is not ...
 - A microprocessor sitting inside a traditional computing system (like desktop, laptop, server, etc.).
- It is actually:
 - A microprocessor used to control another piece of technology (dedicated, and not general-purpose).
 - For low cost, microcontrollers that are typically used are single-chip devices containing processor, memory, and I/O interfaces.

Now, what embedded system is not? Well it is not a microprocessor sitting inside a traditional computing system. We call it a computer, we do not call it an embedded system. Well it is a microprocessor used to control another piece of technology like ac machine, like microwave, like refrigerator and so on, this is what an embedded system is.

It is embedded inside some other system. Now as I had said for low cost microcontrollers, that they are single chip devices. You see here I have shown a picture of a microcontroller that belongs to the PIC family. This is a PIC microcontroller, this is a layout diagram where processor, memory, IO subsystems everything is embedded inside the same chip.

(Refer Slide Time: 25:17)

Applications of Embedded Systems

- **Limited by imagination.**

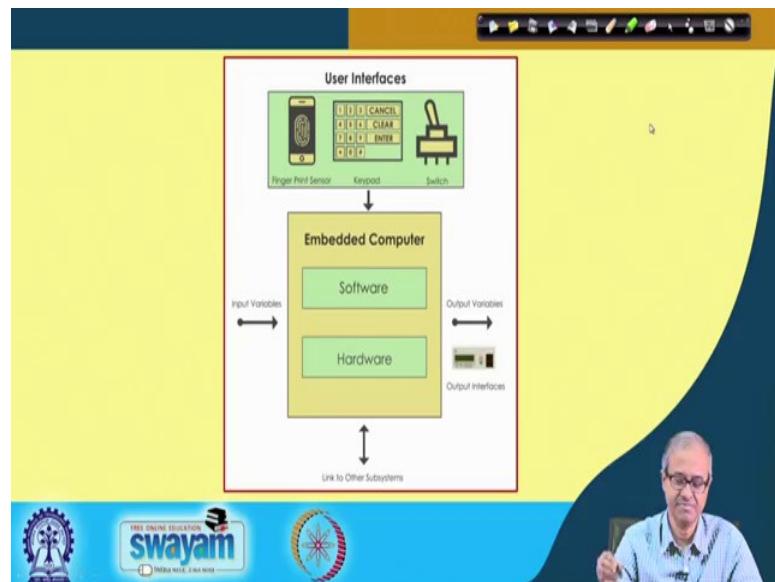
- Consumer Segment:** Refrigerator, washing machine, A/C machine, camera, microwave oven, TV, security system, etc.
- Office Automation:** Printers, Fax machines, photocopying machines, scanners, biometric scanner, surveillance camera, etc.
- Automobiles:** Air bags, anti-lock braking system (ABS), engine control, door lock, GPS system, vehicular ad-hoc network (VANET), etc.
- Communication:** Mobile phones, network switches, WiFi hotspots, telephones, MODEM, etc.
- Miscellaneous:** Automatic door locks, automatic baggage screening, surveillance systems, intelligent toilet, etc.

Now talking about some applications of embedded systems, you can summarize some of them, but they are limited by imagination, there is no limit to what you can imagine. And the number of such applications will only increase day by day. Today we are talking about IoTs, tomorrow IoT will be everywhere, maybe some of those devices will be inside your body, you will be carrying those devices along with you.

In the consumer segment you can think of refrigerator, washing machine, AC machine, camera, this already we have talked about. Even in your office you use printers and fax machines, photocopying machines, scanners, biometric scanner, cameras for surveillance, they are all examples of embedded systems. You think of automobiles; today all automobiles are intelligent in some sense, there are some computational devices that try to utilize the infrastructure inside those automobiles in an efficient way.

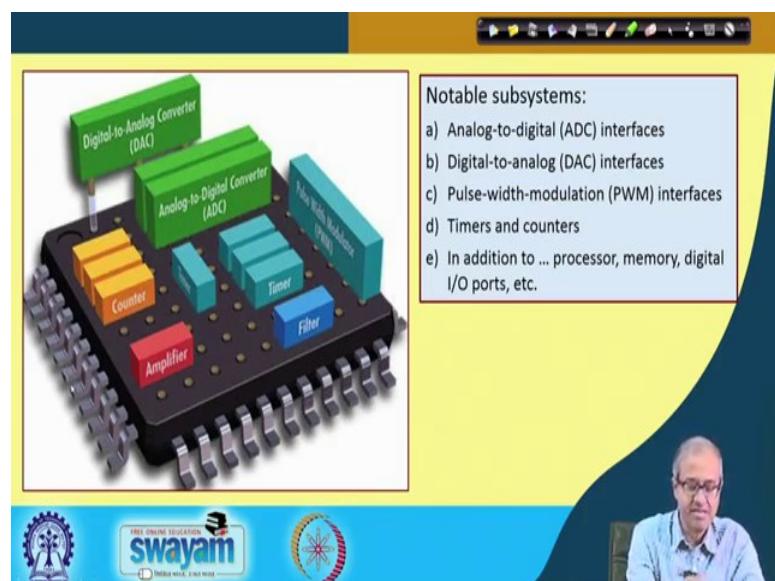
So, airbags, anti-lock braking systems, engine control, door lock, GPS, everything here these are controlled by embedded processors. For communication the mobile phones is the biggest example; you think of the network switches that we use for communication routers, switch, Wi-Fi hotspots, telephones there are processors inside each of them today. Then you can have automatic door locking systems we see it many places; automatic baggage screenings in airports and railway stations, surveillance systems, intelligent toilets, where there are some embedded systems inside toilets that can respond to users' needs automatically these are all examples of embedded systems.

(Refer Slide Time: 27:29)



Again talking about that diagram, this is what embedded system looks like. We have to have an embedded computer with some hardware, you have to write some software to do it, and of course there has to be a proper user interface to make it usable. And user interface has to be simple enough, it should not be a full big large keyboard where you can type anything and everything. It will be very specific to a particular application.

(Refer Slide Time: 28:02)



This is just like a cartoon I am showing. This rectangular box is a processor. Inside the processor you can see so many things. You can see a digital to analog converter, you can

see analog to digital converter, counters, timers, pulse width modulator, so many subsystems. In addition to processor, memory, IO ports etc. a typical microcontroller today that forms the heart of an embedded system can contain analog to digital interfaces, because most of the inputs from the outside world are analog in nature, they are continuous in nature, like temperature, humidity, pressure, etc.

With this we come to the end of this lecture. We shall be continuing with our discussion in our next lecture.

Thank you.

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

Lecture – 02
Design Considerations of Embedded Systems

In our previous lecture, we talked about what an embedded system really is and what it is not. In this lecture we shall be talking about some of the design considerations in embedded system design. When we design an embedded systems, what are the things we should look at, and what are the design consideration and other tradeoffs that you should be aware of. These are the things that we shall be talking as part of this lecture.

(Refer Slide Time: 00:52)



We shall broadly be talking about the design challenges, and we shall also be trying to understand some of the more important design tradeoffs. Design trade off means there can be some conflicting parameters; you can try to improve one of the parameter, but it might degrade some other parameter. You cannot improve everything together, this is called trade off. And there are some cost metrics that are quite important, like non-recurring unit cost matrix. Well, we shall also be talking about these.

(Refer Slide Time: 01:31)

Design Challenges

- Primary design goal:
 - An implementation that realizes the desired functionality.
- The main design challenge is ...
 - To simultaneously optimize several design metrics.
 - Often mutually conflicting.
- What is a design metric?
 - Some feature of an implementation that can be measured and evaluated.

Venn Diagram: A Venn diagram with three overlapping circles labeled 'Speed', 'Quality', and 'Cost'. The regions are labeled: A (Speed only), B (Quality only), C (Cost only), and D (Overlaps of all three: Speed, Quality, and Cost).

Let us look at some issues relating to design challenges first. When you are designing an embedded system, your primary design goal will be to design a system that realizes the desired functionality. Suppose, you are trying to build an embedded system for a washing machine, so your embedded system should be able to function properly in that environment. It should be able to control the washing machine in a proper way, so that as a user you would be happy with the performance.

So, desired functionality is the keyword, some implementation that realizes the design functionality. But in order to have this implementation, you may have some design challenges. There are several design metrics we shall be talking about, and you may have to optimize several of them. Like, user may say I want something which is low cost, small in size, very powerful and also very rugged. But you see if you try to reduce the cost, naturally your performance will also go down, your ruggedness can also be compromised. So, you cannot have best of all worlds; this is something we refer to as tradeoff. These are mutually conflicting in most cases.

I have given a small example where the three circles refer to three parameters or design metrics: speed, cost and quality. Well, in most designs, we want to improve on all these three, but as I had said these three are often mutually conflicting. If you try to reduce the cost you will be sacrificing on the speed and quality and so on.

(Refer Slide Time: 04:26)

Common Design Metrics

- **Non Recurring Engineering (NRE) Cost:** One-time initial cost of designing a system.
- **Unit Cost:** The cost of manufacturing each copy of the system, without counting the NRE cost.
- **Size:** The actual physical space occupied by the system.
- **Performance:** This is measured in terms of the time taken or throughput.
- **Power:** The amount of (battery) power consumed by the system.
- **Flexibility:** The ability to change the functionality of the system.

Let us talk about some of the common design metrics with respect to an embedded system design. First important parameter is called non recurring engineering cost, this is some kind of initial cost.

Suppose you have a design idea, you are going to manufacture the system. But before you start the manufacturing process, you will have to install some machines in your factory that will help you in the manufacturing. And that initial installation will incur some initial cost; this is what is referred to as non recurring expenditure. This will be required only once at the beginning. Once we have that infrastructure ready, you can manufacture the systems as many you want; you can manufacture 10, 20, 100, 1000 as many you want. The non recurring cost is the one time initial cost.

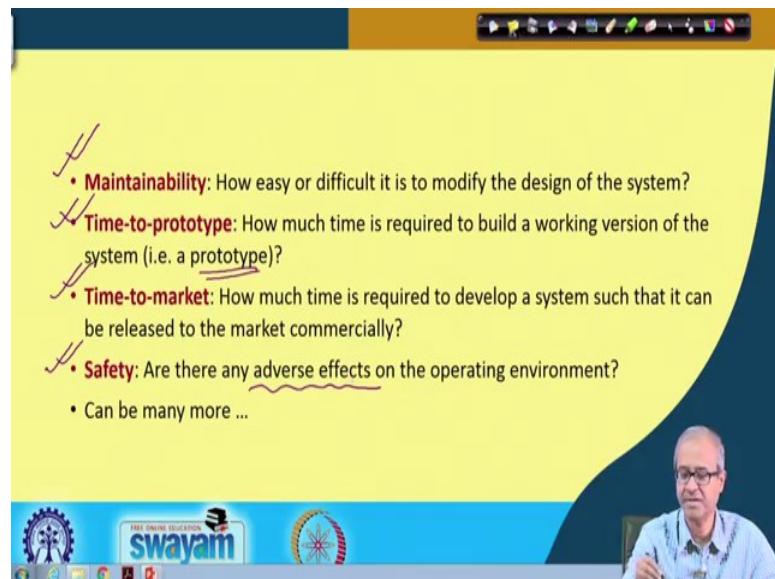
And once you have this NRE cost covered, you talk about unit cost. After we have that infrastructure, what is the cost of manufacturing every copy of the system, which you define as the unit cost. The unit cost will also consider the cost of the raw materials, labor cost, and so on. Size is an important parameter I talked earlier. Most of the embedded systems need to conform to a very small form factor, so that it can fit nicely inside the environment for which it is meant.

Performance again depends on the application. For some application you are may not be that much aware about the performance, you need very little computational capability, but there are some applications where performance is important. You need to ensure that

performance is assured. Power consumption, this is important. Most of the systems we see today, they run on battery. They need to consume very low power. Well, our mobile phone is a very classic example of an embedded system in that respect.

Flexibility means the ability to change the functionality of the system. The system was initially designed for certain application. Is it possible to modify it, so that you can also use it for some other application? Well if that flexibility is there, then possibly for the second development your total cost would be much less. So, flexibility is also an issue that needs to be considered.

(Refer Slide Time: 07:57)



Maintainability says that I have already purchased an embedded system as part of a larger system. Let us say I have purchased an AC machine. Tomorrow the company says that we have come up with a better AC machine that has a Bluetooth interface, which can interface with your mobile phones and computers and laptops. So, you can control the AC machine even from your mobile phone, even from your laptops by installing some apps.

So, is it possible for the older system to adapt to this new technology? Can you modify this design so that this added functionality can be incorporated? This is what we mean by maintainability. But of course, maintainability is possible only to a limited extent. Just for the example I cited, for having Bluetooth you need to have a Bluetooth interface. So, if it is not there in the first place, you cannot have Bluetooth.

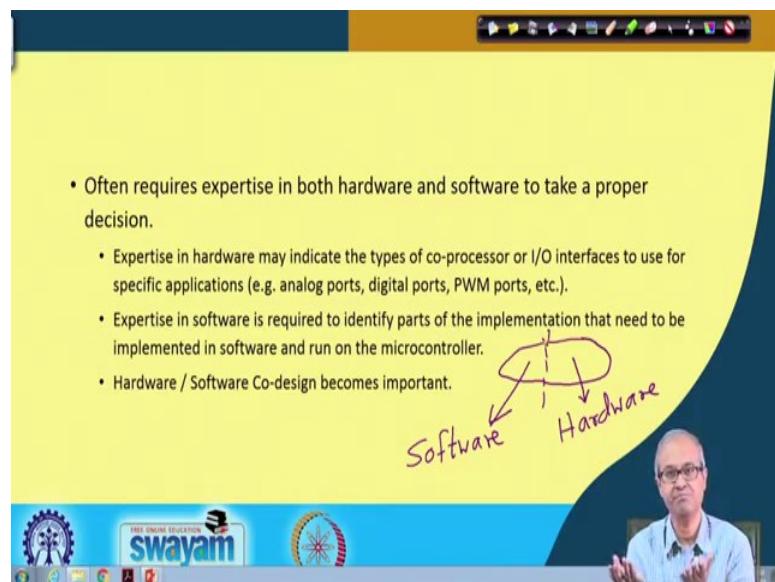
Then time to prototype. How much time is required to build the first working version of the system that is called a prototype? How much time is required to have the first version ready so that you can test? And after the prototyping is done, comes time-to-market. How much time it is required to build a finished product that can be sold in the market? Well, prototyping and finished products are two entirely different things. Finished products have to have durability, finishing, quality everything in place. And of course, safety is an important issue for many applications. You will have to be sure whether there are any adverse effects on the environment because of use of that system. And there can be many more such things, I have only listed a few of them.

(Refer Slide Time: 10:15)

Talking about design tradeoffs, there can be several design requirements that are mutually conflicting. Here in this diagram, I am showing four such metrics, power consumption, performance, non recurring expenditure, and size. Now, these four can be mutually conflicting; if you pull one, the others will be pulled in the reverse direction, means if you want to improve one maybe the others might get degraded. So, you will have to take a holistic view. This is the task of the designer. You cannot improve everything at once.

Like for example, the very slim laptops that are being built. Well you do not expect that those laptops will become powerful in all respects. To make it slim somewhere there is some compromise that has been made. These are examples of design tradeoffs.

(Refer Slide Time: 11:32)



And another important thing is that talking about design tradeoffs, for embedded systems you need a different kind of trained professionals. You think about the conventional computing days or traditional computing environment, where you have a set of engineers, who develop the hardware, the Pentium processor is developed by Intel by a set of highly qualified hardware engineers. There are some other groups who develop software for those computer systems that use those chips.

For example, in Microsoft there are some software engineers who developed the operating system, who develop utilities like Microsoft Office, and so on. So, there are hardware experts, there are software experts. Their role is more or less independent of each other. The hardware expert need not know software very well and software experts need only know very little about the hardware. But now when you are designing an embedded system, you are talking about a very small system where both hardware and software will be there. And talking about the design trade off, you will have to play with both of them.

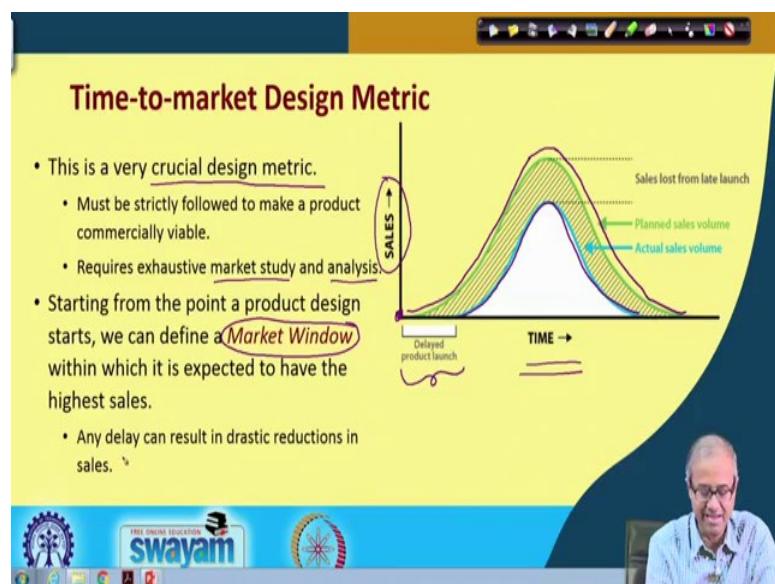
So, now you need to have the expertise of knowing the hardware, how to use it and also software how to program it in an efficient way. So, both hardware and software expertise are required this is what I wanted to say here. You may need expertise in hardware to identify what kind of coprocessors or IO interfaces you will be requiring for a certain applications, because you may have several choices. There can be some analog ports,

digital ports, pulse width modulation ports, but which of them will be best for a certain application? So, unless you are a hardware expert, you really cannot take a good decision here.

Similarly, you also need to have expertise in software, because you need to decide which parts of the implementation you need to implement in software, and which part is already implemented by some hardware. You talk now about something called hardware software co-design, meaning you are designing both hardware and software subsystems together.

Suppose you have a large system to be designed. You will have to decide that well I make a demarcation, one of the parts will be implemented in software, this will be running as a program on a microcontroller, and the other part will be implemented by some specialized hardware circuit. So, where this boundary will be is a matter of design tradeoff, the designer will decide how to shift this dotted line, so that desired performance and other criteria are satisfied in the best possible way.

(Refer Slide Time: 15:29)



There is an important thing here, time to market design metric. Let us try to explain this. See a company whenever it manufactures some products the ultimate goal will be to generate some profit out of it. So, marketing is the most important issue for any company. The quality of the product becomes secondary. Sometimes many companies compromise on the quality of product in order to bring the product to the market earlier,

so that they can reap greater profit out of it. Time to market is a very important or crucial design metric.

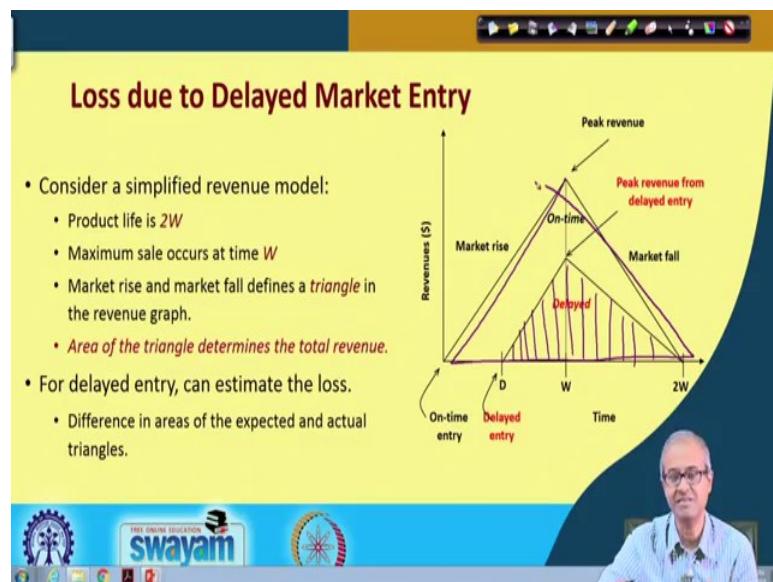
This must be strictly followed to make a product commercially viable. If you delay in the launch of a product, you may incur a big loss. Maybe some of your competitors already have hit the market with a similar product, and people will buy naturally from them and not from you. And this requires very careful market study and analysis. Now, this we are illustrating with a diagram like this. Here the x-axis shows the time and y-axis shows the sales.

Suppose your origin indicates the point where you are expected to bring the product in the market. So, what happens when a product comes to the market, well the users or the customers need some time to learn about the product. The sale increases slowly over time. And there will be a point where this sale will saturate and beyond that point again this sale will start dropping down. And there will be a point where the product will become obsolete, no one is buying it anymore. This is a typical curve, sale versus time.

And suppose there is a delay in the product launch. So, instead of this curve, you are now following this inner curve. There was an initial delay. So, you see the curve will be similar again, but the peak or the maximum sale can that you can achieve is becoming much less. The total area under this curve will denote your total revenue, how much total sale you have been able to do. The total area under the curve will indicate your total sales over that period of time. This curve area under the curve is defined as the market window. Any delay can result in a drastic reduction in sales.

Let us make a simple calculation with some approximation. Well the curve looks like this, but to a first approximation we can assume that these are straight lines, you can assume that this is straight line, this is also a straight line.

(Refer Slide Time: 19:10)



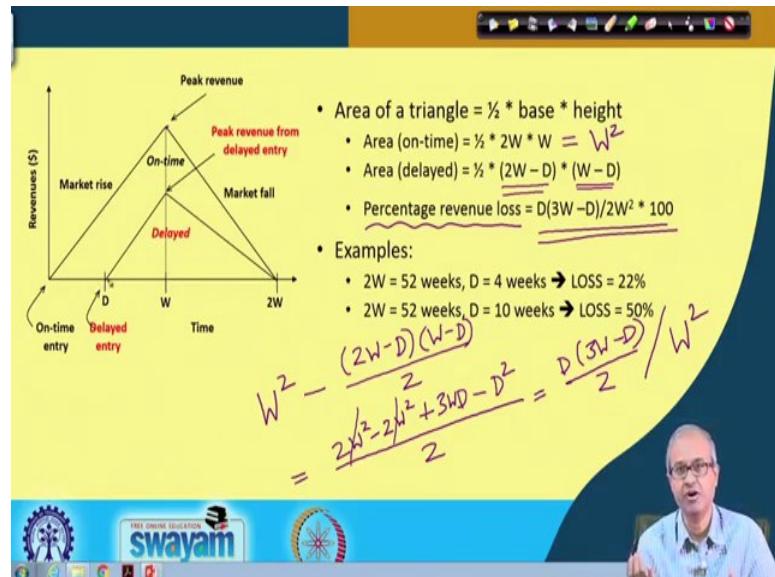
With that assumption let us try to estimate how much loss we are expected to incur if there is a delay in entering the market. We considered a simplified revenue model, you can modify it very easily to mean whatever is more realistic in your specific case. Here what we assume is that as this graph shows that the product life is $2W$. So, from 0 up to $2W$, this is your total product life. After $2W$, no one will be buying your product. And let us assume that the maximum sale occurs in the middle of it, at point W the sale becomes maximum. Market rise and market fall we approximate it straight lines that means we define a triangular structure.

And as I mentioned the total area under this triangle will determine the total revenue that you can generate. Now, if we have a triangle you know how to measure the area of the triangle. Now, for delayed entry as I had said in the previous diagram, suppose there is a delay in the entry by this much. So, now, your triangle starts from here at this point D , but it will again reach the peak at the same point because depending on market dynamics beyond a certain point interest in that product will start to go down.

So, the fall will start happening at the same point W . So, from then again it will continue up to $2W$ beyond which there will be no interest in that product anymore, but your initial delay is actually shifting the left edge or the left vertex of your triangle to the right, this is what is happening. There will be a difference in the area of the expected and actual triangles. Like your actual triangle area will be only this much, but your expected area

was the area of this whole larger triangle. It is naturally much less in this case because of the delayed launch.

(Refer Slide Time: 21:46)



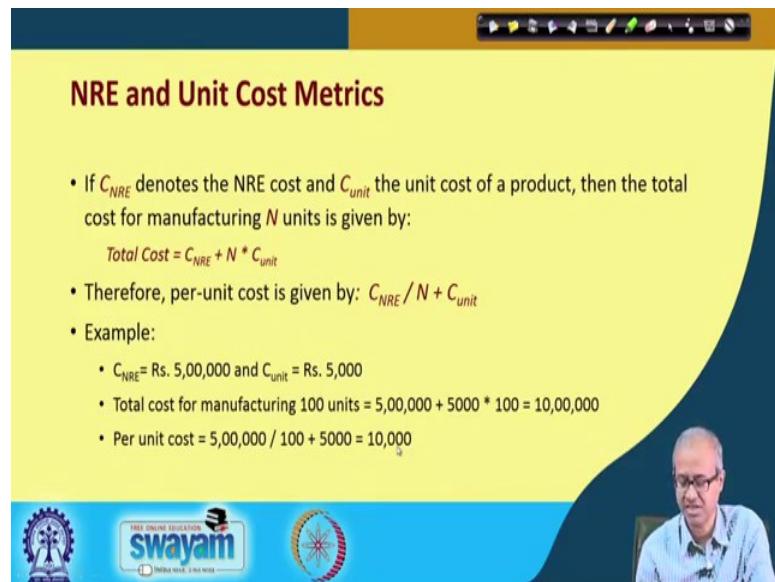
Let us make a calculation. We show that same plot on the left. We know that the area of a triangle is defined as half base into height. For the on-time case, that means, the larger triangle area will be half of base, which is $2W$, and height is W let us assume this is an equilateral triangle, it is rising at 45 degree slope. So, height let us assume this is also W . The area becomes W^2 .

For the delayed case your base becomes $2W - D$. And your height will become $W - D$ as there was a delay. The area calculation is shown.

The percentage revenue loss calculation is also shown.

As an example, suppose your window size is 1 year = 52 weeks. If your delay is 4 weeks, your loss becomes 22%. But if you are delayed by 10 weeks, your loss becomes as large as 50%. So you see that your loss increases very rapidly, and this delay becomes very costly for the financial viability of a product. This is a very important consideration.

(Refer Slide Time: 25:28)



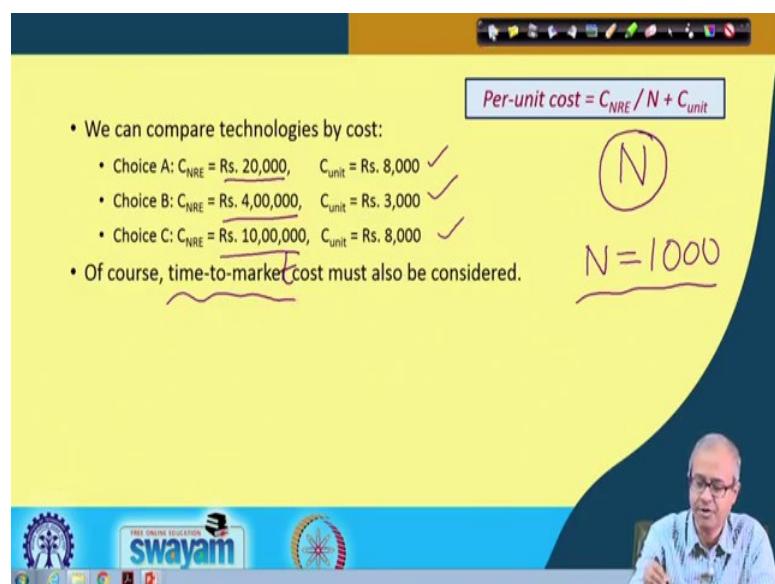
NRE and Unit Cost Metrics

- If C_{NRE} denotes the NRE cost and C_{unit} the unit cost of a product, then the total cost for manufacturing N units is given by:
$$\text{Total Cost} = C_{NRE} + N * C_{unit}$$
- Therefore, per-unit cost is given by: $C_{NRE} / N + C_{unit}$
- Example:
 - $C_{NRE} = \text{Rs. 5,00,000}$ and $C_{unit} = \text{Rs. 5,000}$
 - Total cost for manufacturing 100 units = $5,00,000 + 5000 * 100 = 10,00,000$
 - Per unit cost = $5,00,000 / 100 + 5000 = 10,000$

Now, talking about the NRE and unit cost, if C_{NRE} denotes the non recurring engineering cost, and C_{unit} denotes the unit cost. Then for manufacturing N units of the product the total cost will be $C_{NRE} + N * C_{unit}$. Now, if you try to calculate how much cost we have incurred for every unit, the per unit cost, you divide the total cost by N . So, it $C_{NRE} / N + C_{unit}$.

If your NRE cost is Rs. 5 lakhs and unit cost is Rs. 5000, then for manufacturing 100 units the total cost becomes Rs. 10 lakhs. So, per unit cost will be Rs. 10000.

(Refer Slide Time: 27:04)



Per-unit cost = $C_{NRE} / N + C_{unit}$

- We can compare technologies by cost:
 - Choice A: $C_{NRE} = \text{Rs. 20,000}$, $C_{unit} = \text{Rs. 8,000}$ ✓
 - Choice B: $C_{NRE} = \text{Rs. 4,00,000}$, $C_{unit} = \text{Rs. 3,000}$ ✓
 - Choice C: $C_{NRE} = \text{Rs. 10,00,000}$, $C_{unit} = \text{Rs. 8,000}$ ✓
- Of course, time-to-market cost must also be considered.

$\text{N} = 1000$

Just a very simple calculation I am just illustrating. Suppose you have several choices; like to manufacture a product you see that there are several different kind of equipments you can purchase to manufacture them. And the initial costs of the equipments are also quite different. Suppose you have three choices for the first choice the non-recurring costs is 20000, second one 4 lakhs, third one 10 lakhs. And the unit costs for the three cases are coming to this, this and this.

So, you can calculate the per unit cost for some particular value of N; well I leave it as an exercise for you. Suppose I tell I want to manufacture 1000 units of product, then using this formula you calculate the per unit cost for choice A, choice B, and choice C; which one of them is better. Not only that you will also have to consider the time to market cost, because maybe means one choice is good in terms of cost, but you are taking a long time to start that machine, because installing, training and just using that machine is requiring much more time that also you also have to need to consider. Thus not only the cost, but also the time to market.

(Refer Slide Time: 28:46)

Performance Design Metric

- Most widely used, but can also be most misleading.
 - Must be careful in the evaluation.
- Some of the measures:
 - Clock frequency/MIPS -- not very good for comparison
 - Latency (response time)
 - Throughput
- Measure of speedup among design alternatives.

And in addition you have some performance design metrics that also need to be considered. Normally, these are traditionally used for evaluating processor performances. The same consideration you can try to use for an embedded system, but there are a few things you need to remember. This performance design metrics tell you that which processor is faster than the other in some respect. These are quite widely used, but it is

also true that if you do not understand the meaning of these metrics they can be the most misleading.

Most of the computer manufacturers try to mislead the customers by quoting some figures with respect to benchmarking and speeds, which if you dig deeper and try to understand, you will actually feel that for your particular environment those numbers make no sense. You may require something else, which will be best suited for your application. So, you must be very careful in the evaluation.

Some of the typical measures that are used again are shown. Clock frequency, well you see someone is saying 1 GHz, 2 GHz, 3 GHz does faster clock frequency means a faster computer? Not necessarily. There are many other things that determine the actual performance of a computer system. There are some measures like MIPS; this also does not mean anything. What kind of instructions are they? Simple instructions or complex instructions? Your MIPS figure will vary drastically among them. So, it does not give any fair measure. Only with respect to some specific application, you should be able to measure how much time it is taking to run my application. That is something that would be most beneficial in your assessment.

Latency response time: you give an input after how much time you are getting back the output? This can be of course one good measure for many real time applications. Throughput may not be that good for an embedded system. It means number of computations that can be carried out per unit time. Normally, we talk about throughput for conventional computer systems, but for an embedded system that is meant for a very specific application, we normally do not talk about throughput.

And if you have several design alternatives, you should have a way to compare their speeds. These assessments are not easy. The best way is to run the application you want to run on a real computing machine and see how much time it takes. Whatever the manufacturer says you take them with a pinch of salt, do not trust them 100%.

With this we come to the end of this lecture where we talked about some of the design metrics that are relevant in embedded system design, and some of the implications with respect to the cost and revenue.

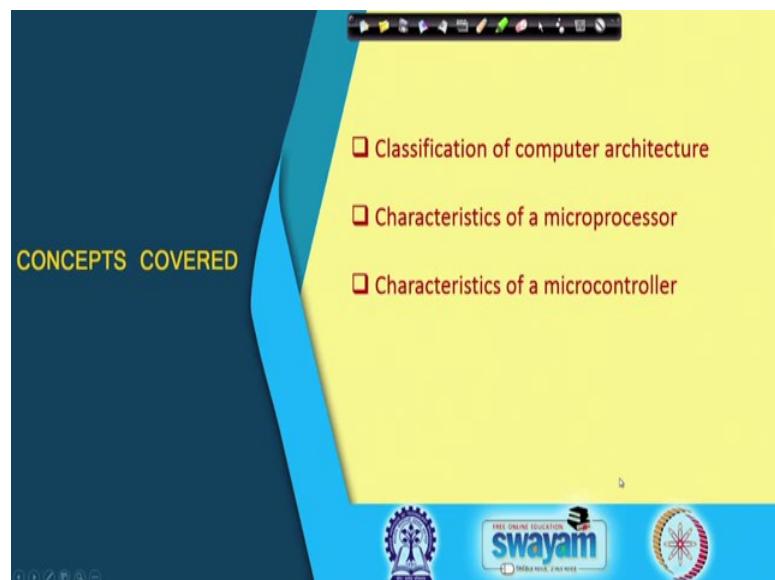
Thank you.

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

Lecture – 3
Microprocessors and Microcontrollers

In this lecture we shall be talking about some basic concepts of microprocessors, microcomputers and microcontrollers, because these are the brain or the processing power behind any embedded system that we see around us. It is always good to know what are the main differences between these.

(Refer Slide Time: 00:49)

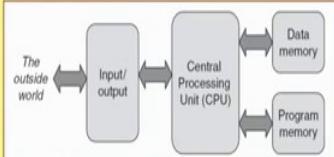


We shall be talking about some classification of computer architectures, followed by the main characteristic features of microprocessors and microcontrollers.

(Refer Slide Time: 01:06)

Basic Operation of a Computing System

- The central processing unit (CPU) carries out all computations.
 - Fetches instructions from the program memory and executes it; may require access to data in data memory.
- The input/output block provides interface with the outside world.
 - Allows users to interact with the computing system, and also observe the output results.

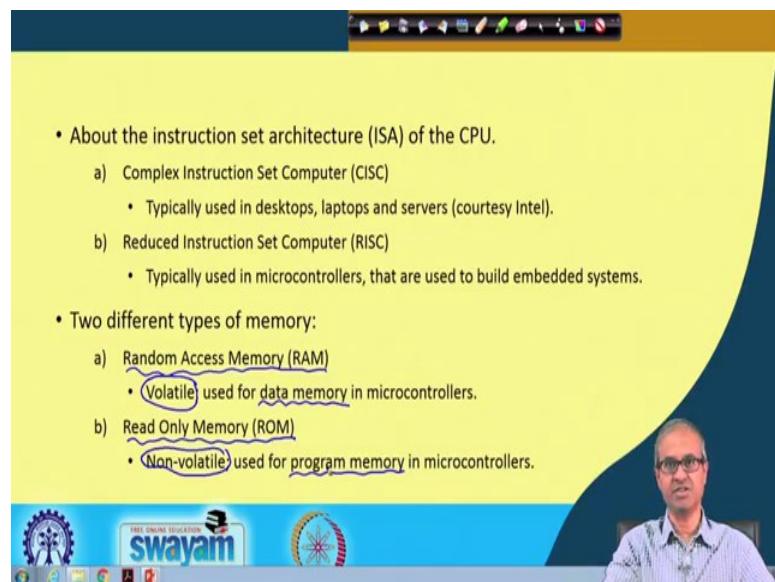


Let us start with how a computer system works basically. With respect to a computer system, so you may know that there is something called CPU that carries out all computations or calculations. You can see this schematic diagram, the CPU is sitting in the middle and there are some memory where program and the data are stored. What does the CPU do? It fetches instructions from the program memory one by one, executes them, and during execution it may require to transfer some data between the data memory, either reading a data or writing the data back into the data memory.

In addition to processing, the computer system often needs to interface or communicate with the outside world. The outside world is typically the user who is using a computer system for conventional systems like a desktop or a laptop. The IO subsystem allows users to interact with the computing system.

The outside world may not always be a human being, it may be environment, it senses some temperature, pressure, humidity -- some parameters of the environment, and those will be the inputs. And similarly it can take some corrective action like it can turn on a heater, turn on the compressor, and so on; these are the output devices.

(Refer Slide Time: 03:08)



- About the instruction set architecture (ISA) of the CPU.
 - a) Complex Instruction Set Computer (CISC)
 - Typically used in desktops, laptops and servers (courtesy Intel).
 - b) Reduced Instruction Set Computer (RISC)
 - Typically used in microcontrollers, that are used to build embedded systems.
- Two different types of memory:
 - a) Random Access Memory (RAM)
 - (Volatile) used for data memory in microcontrollers.
 - b) Read Only Memory (ROM)
 - (Non-volatile) used for program memory in microcontrollers.

Talking about the instruction set architectures this is one way in which you can classify computer systems. What kind of instructions the CPU is capable of executing depending upon the broad classification with respect to instruction sets? We can classify computers as either a CISC architecture or a RISC architecture. CISC is the short form for Complex Instruction Set Computer. The most common example of such CISC architecture are the Intel classes of processors which dominate the desktop, laptop and server markets.

More than 90% of the processors are made by Intel or some clones of those made by Intel. These are called CISC, the instruction can be fairly complex with lot of flexibilities and features.

There is another philosophy called reduced instruction set computers or RISC. Here the instruction set is made very simple, and so it is much easier to implement the computer system in hardware. In some sense it is more efficient with respect to execution of the instructions. Most of the microcontrollers that we see today they are based on the RISC architecture, because of primarily this reason, they are easy to implement. Not only microcontrollers, most of the modern processors at some level are based on the RISC philosophy of instruction execution because it makes the instruction execution unit much more efficient.

So, even the Intel processors I am talking about those are based on CISC; internally these complex instructions are broken up into simpler instructions or micro instructions, they

look more like a RISC instruction set, which are then executed efficiently using a controller. Talking about the memory system broadly speaking there can be two kinds of memory; one which is called random access memory where you can both read and write, and the other is read only memory when you store something permanently you can only read from them.

RAM are typically volatile, volatile means they retain the values only during the time the power is switched on, as soon as you switch off the power the data gets lost. For microcontroller RAMs are typically used to store data in the data memory, but when you talking about program memory the place where you are storing a program, this memory is typically non volatile; which means because it is a ROM, once you store it even if you switch off the power the program will not get destroyed.

In most microcontrollers there is an area of program memory which is implemented using ROM. But nowadays there are microcontrollers which use some alternate technologies like flash memory, which is also non volatile where you can store some program, you could also change the program if you want, but if you switch off the power the program does not get destroyed.

(Refer Slide Time: 07:28)

Classification of CPU Architecture

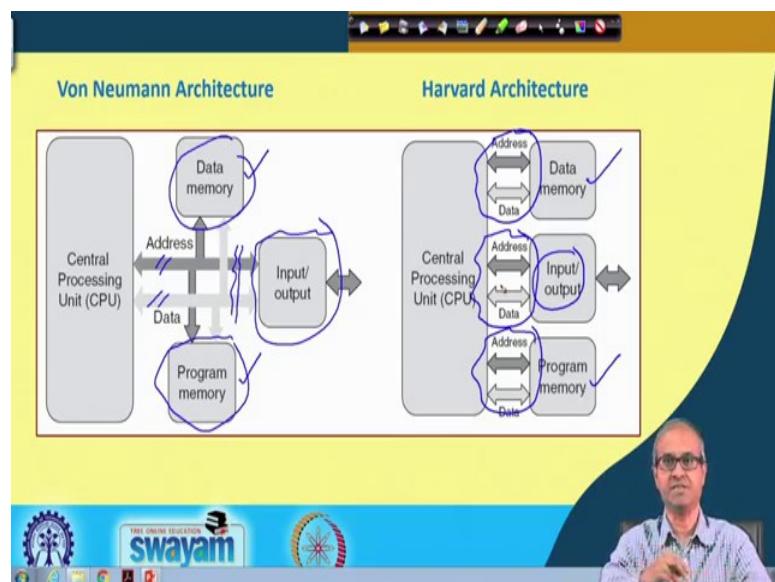
- Broadly two types of architectures:
 - a) Von Neumann Architecture
 - Both instructions and data are stored in the same memory.
 - This model is followed in conventional computing systems.
 - b) Harvard Architecture
 - Instructions and data are stored in separate memories.
 - Typically followed in microcontrollers, used for building embedded systems.
 - Instructions are stored in a ROM (permanent), while temporary data are in RAM.

Broadly with respect to CPU architectures we are so far talking about instruction set architectures, now talking about how the CPU works there is a broad classification here. We talk about Von Neumann and Harvard class of architectures. The basic difference is

that in the Von Neumann Architecture we have a single memory where both instructions and data are stored. Most of the conventional computer systems that you see around us are based on Von Neumann architecture.

But in Harvard architecture conceptually there are two separate memories; in one you store the program, in another you store the data. Most of the microcontrollers follow this principle; they have separate memories. So, instructions are typically stored in a ROM while data are stored in a RAM.

(Refer Slide Time: 08:47)



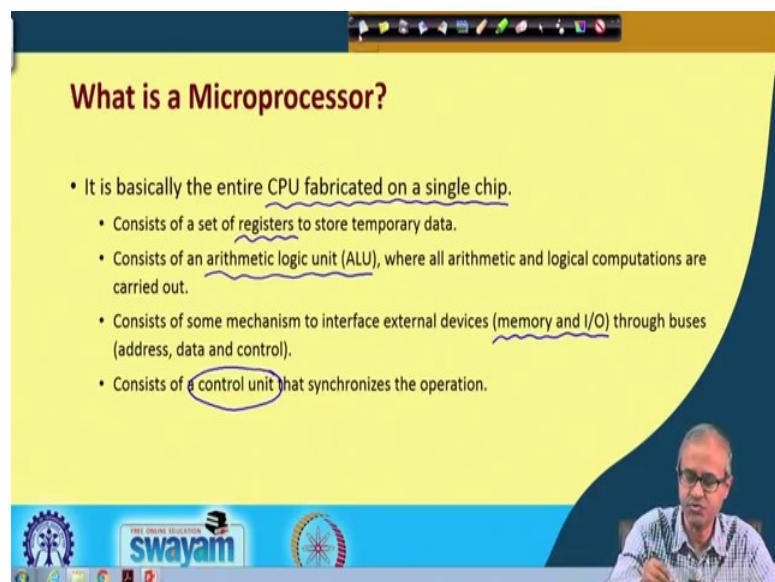
Pictorially Von Neumann and Harvard architectures can be shown like this. In a Von Neumann architecture as you can see here are the data and program memory; although you are showing them as separate, but CPU is accessing them over the same data bus using the same address.

So, with respect to CPU it appears that the memory is the same, but in practice we may store them separately in separate parts as this diagram shows. But with respect to the CPU the way the addresses are generated are unified, same address and data lines are used to transfer program as well as data. And IO devices are also typically connected through the same bus. This is how typical Von Neumann Architectures look like. Of course, there are sophisticated architectures where there can be multiple buses for parallel transfer of data and so on.

In Harvard architecture there are separate data and program memories, and address and data buses are entirely separate. So, while you are fetching a program in parallel you can also fetch or write some data into data memory. And in some architectures the IO devices are also connected via separate address and data buses.

The drawback is that you need so many buses, lot of IO pins to interface the external devices, but the advantage is that you get on the average faster and parallel data transfer features.

(Refer Slide Time: 10:56)



What is a Microprocessor?

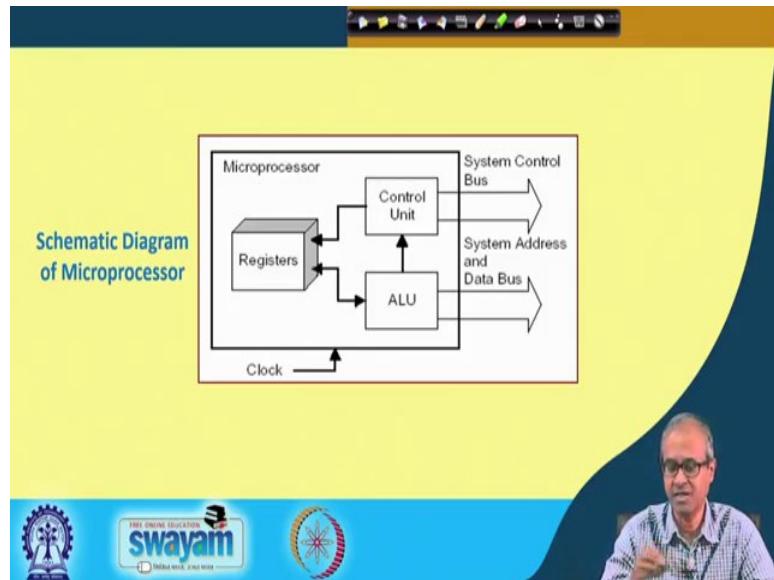
- It is basically the entire CPU fabricated on a single chip.
- Consists of a set of registers to store temporary data.
- Consists of an arithmetic logic unit (ALU), where all arithmetic and logical computations are carried out.
- Consists of some mechanism to interface external devices (memory and I/O) through buses (address, data and control).
- Consists of a control unit that synchronizes the operation.

Let us now come to a microprocessor. What is a microprocessor? Well we have seen what is a CPU. Microprocessor is a general term which means a CPU fabricated within a single integrated circuit or IC chip. With the advancement in semiconductor technology earlier CPU circuits were very large and bulky; they have started to become smaller and smaller, and now they can fit inside a single chip. For example, if we look at the Pentium CPU chip there are close to billions of basic components or transistors inside the chip.

Inside the microprocessor what are the things that are typically there? There are a set of registers, some are general purpose some are special purpose, which are used for temporary storage of data. There is an ALU where all the data processing are carried out, both arithmetic operations and also logical operations, and there is some mechanism for interfacing memorial IO devices. There are some external bus and control mechanism through which you can connect memory or IO devices with the microprocessor. And of

course, there is a control unit which can be considered as the brain of the system, which controls the entire operation of the microprocessor, it synchronizes all internal operations. This is what a microprocessor basically is.

(Refer Slide Time: 12:44)



Schematically you can see these are the main components, you can see the registers here, you can see the ALU, you can see the control unit. And externally there are some buses, address and data bus and there can be control bus, address and data buses will be used to interface with external memory and IO devices. And the control bus will contain other signals like read, write, enable, different kinds of interrupts and other things, some signals for interfacing slow memories. There are many such miscellaneous signals that are used for control purposes.

(Refer Slide Time: 13:30)

What is a Microcomputer?

- It is a computer system built using a microprocessor.
- Since a microprocessor does not contain memory and I/O, we have to interface these to build a microcomputer.
- Too complex and expensive for very small and low-cost embedded systems.

The diagram illustrates the architecture of a microcomputer. At the center is the **Microprocessor**, which is connected to three main components via **Address Bus**, **Data Bus**, and **Control Bus**: **Memory** (represented by a box with a gear icon), **I/O Ports** (represented by a box with a plug icon), and an external **Monitor** (represented by a screen icon). The **Memory** and **I/O Ports** are also connected to each other.

Now, let us see what is a microcomputer. Microcomputer is a computer which is built around a microprocessor. You look at our desktop, you look at our laptop, there is a Pentium we talk about Intel i3, i5, i7 class of processors inside our modern desktops and laptops. They are essentially microprocessors and inside a desktop or a laptop it is not only the microprocessors, there are memories, there are other devices, there is a disk drive there are some other interfaces to connect keyboard and mouse for example, printers that makes our entire computer. This is a microcomputer, it is a computer system built using a microprocessor.

Now, a microprocessor contains basically some registers, ALU and control; it does not contain any memory or any facility for IO. To make a computer system we have to interface all these devices with the microprocessor. But, the trouble is that since you have to interface so many things around a microprocessor, the system becomes complex, bulky and also expensive. In addition it also consumes significantly higher power. It is fine if you use it for a high performance application like a desktop or a laptop, but not for embedded system applications where ultra low power, portability, small size these features are quite essential.

A microcomputer as you can see is built around a microprocessor, you have memory, you have I/O ports and through this I/O ports you can interface with several I/O devices.

(Refer Slide Time: 15:50)

Microcontrollers: The Heart of Embedded Systems

- It is basically a computer on a single chip.
 - Very inexpensive, small, low power.
 - Convenient for use in embedded system design.
- It operates on data that are fed through its serial or parallel input ports, controlled by the software stored in on-chip memory.
 - Often has analog input pins, timers and other utility circuitry built-in.

Microcontroller

```
graph TD; MC[Microcontroller] --- Memory[Memory]; MC --- Registers[Registers]; MC --- ALU[ALU]; MC --- IOPorts[I/O Ports]; MC --- Counters[Counters]; MC --- Timing[Timing & Control]; MC --- Interrupt[Interrupt Circuits]; Memory <--> Registers; Registers <--> ALU; ALU <--> IOPorts; IOPorts <--> Counters; IOPorts <--> Timing; Counters <--> Timing; Timing <--> Interrupt; Interrupt --> ALU
```

The slide also features a video player interface at the bottom with a video thumbnail showing a man speaking, and the Swayam logo.

Now, let us come to microcontrollers that are the heart of embedded systems. What is a microcontroller? A microcomputer we have said, it is a computer system built around a microprocessor. Now, if the whole of the microcomputer we can shrink and put inside a single chip, I get a microcontroller.

Since I am putting everything on a single chip, the total chip area has to be shared by processor, memory, etc. So, a microcontroller is basically a computer on a single chip, because it is on a single chip it is relatively very low cost and inexpensive. It is small, it also consumes very low power.

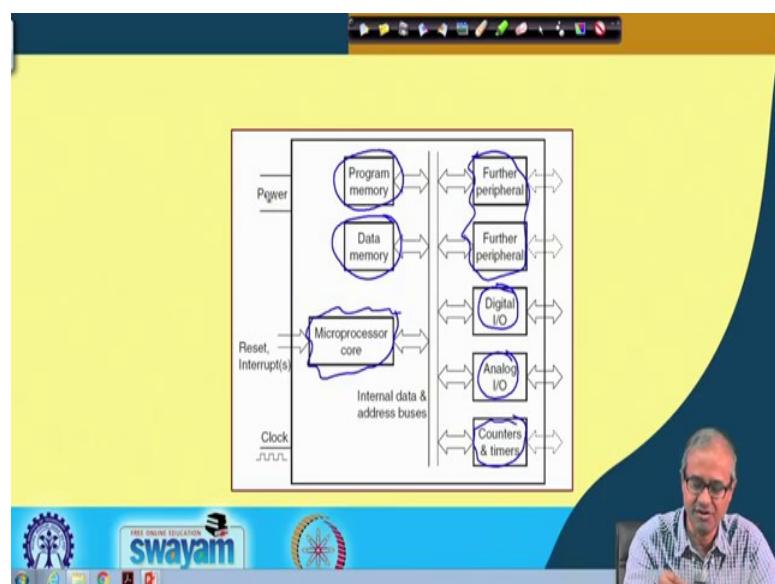
In fact and these are very widely used in embedded system design. If you look at this schematic what are the typical things that are present, you will see that, there is CPU, registers, ALU's and control. There is some memory of course, you cannot have very large memory, some memory is there that is small in size. There can be some facility for connecting IO devices. There are some timers and counters that are required for many applications, there are interrupt circuits.

And there are many other facilities like analog ports. Most of the signals that are coming from the outside world they are continuous or analog in nature, they are not digital. So, if we have some kind of analog to digital converter built into the chip, it becomes very convenient to interface such devices. Typically all those facilities are provided inside the

chip. Microcontrollers typically operate on data that are fed through some input ports; data coming from outside through the ports, and there is a program which is running.

And as I have said they have analog pins, timers, counters and other utility circuitry. For example, you can have a pulse width modulation circuit which is very useful as we shall see later.

(Refer Slide Time: 18:59)



This is a slightly more detailed diagram of a microcontroller. You have the basic microprocessor here, register, ALU and the control, there is some program memory, a memory to store the program, there is a memory to store your data (RAM) and there can be some input output pins. As you can see some of the IO can be digital, some may be analog; there will be counters and timers, and there can be sophisticated kinds of interface also, like serial IO interfaces, pulse width modulated interfaces, interrupt etc. All these facilities are provided in typical microcontrollers today, so that you can use it for almost any kind of embedded system applications. In addition you have power supply, reset, interrupts, clocks etc.

(Refer Slide Time: 20:07)

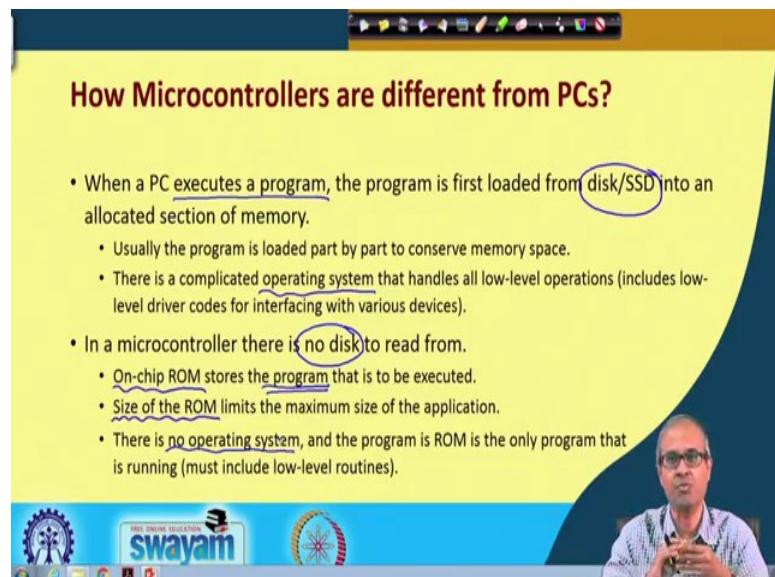


Microcontroller Packaging and Appearance

From left to right:
PIC 12F508, PIC 16F84A, PIC 16C72, Motorola 68HC05B16, PIC 16F877, Motorola 68000

You can see these are some typical microcontrollers that are manufactured by a company called PIC. Now, in comparison you see one of the older microprocessors Motorola 68000. This was a 48 pin chip, so large in size. In comparison the smallest PIC processor is a fraction of the size of a coin as you can see.

(Refer Slide Time: 20:53)



How Microcontrollers are different from PCs?

- When a PC executes a program, the program is first loaded from disk/SSD into an allocated section of memory.
 - Usually the program is loaded part by part to conserve memory space.
 - There is a complicated operating system that handles all low-level operations (includes low-level driver codes for interfacing with various devices).
- In a microcontroller there is no disk to read from.
 - On-chip ROM stores the program that is to be executed.
 - Size of the ROM limits the maximum size of the application.
 - There is no operating system, and the program in ROM is the only program that is running (must include low-level routines).

Now, microcontrollers are computers in their own right. PC's are also computers built around a microprocessor. So, what are the main differences? There are still some differences; when a PC executes the program, from where does the problem come from?

They are initially stored either in the hard disk or in solid state drive based on flash memory technology, from there the program gets loaded into memory and from memory the instructions are executed one by one. This is how a typical computer system works. There is an operating system like Windows or Linux, they are fairly complicated programs. They are responsible for handling all low level operations while the program is getting executed. Whenever there is an input output operation it is the operating system, the drivers therein who will be invoked to take care of the IO operations.

But a microcontroller is a very small system. You cannot afford to have a disk, everything will be inside that microcontroller itself. In a computer system you can load any program you want from the disk and run it, but a microcontroller that is sitting inside an AC machine will only run that program that is meant for controlling the AC machine. So, in a typical microcontroller there is a small ROM inside the chip, this will be storing the fixed program, the program cannot be changed. The point to note is that the maximum size of the ROM that is provided limits the size and complexity of the program that you can run on the microcontroller.

And there is no operating system, whenever you reset or power on your system the program which is stored in the ROM starts running. These are broadly the main differences between a microcontroller and a normal computing system like the PC.

(Refer Slide Time: 23:35)

The slide has a yellow header with the title 'Where are Microcontrollers Used?' in red. The main content is a bulleted list of applications:

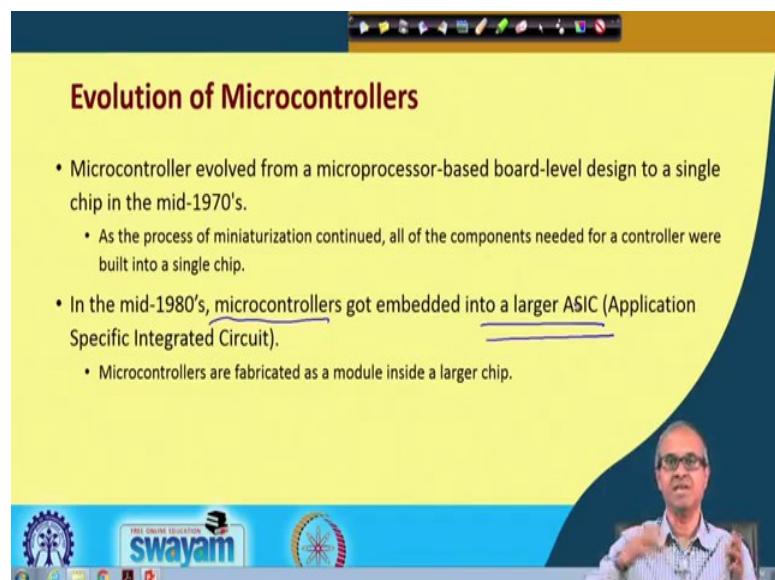
- Typically in applications where processing power is not critical.
 - Modern-day household can have 10 to 50 such devices embedded in various devices and equipments.
- One-third of the applications are in the office automation segment.
- Another one-third are in consumer electronics goods.
- Rest one-third are used in automotive and communication applications.

At the bottom, there is a video player interface showing a man speaking, with the 'SWAYAM' logo and other educational icons.

Microcontrollers are used in applications where processing power is not critical, you do not need very high processing power rather you need low power, small size, low cost, these kinds of attributes. You typically find many such devices in modern day households, whatever equipment you purchase you deploy and install, there will be some embedded microcontroller inside it.

So, there are many applications I already talked about in office automation segment, consumer electronics, automotive, vehicles, airplanes, rockets missiles, communication you will find these devices everywhere.

(Refer Slide Time: 24:26)



Evolution of Microcontrollers

- Microcontroller evolved from a microprocessor-based board-level design to a single chip in the mid-1970's.
 - As the process of miniaturization continued, all of the components needed for a controller were built into a single chip.
- In the mid-1980's, microcontrollers got embedded into a larger ASIC (Application Specific Integrated Circuit).
 - Microcontrollers are fabricated as a module inside a larger chip.

Now, talking about evolution, since the 1970's the 1st generation of microcontroller started to evolve. There was one microcontroller which was developed by Intel, it was 8051, it is still being used, but talking about the power, flexibility and innovativeness, 8051 has an old kind of an architecture, it does not have too much flexibility.

Starting from mid 80's and the process is continuing, what we see now is that microcontrollers are getting embedded inside larger chips. These are called application specific integrated circuit. Like one of the microcontroller that we shall be demonstrating as part of this course, this microcontroller board which is from ST microelectronics, it has something called ARM Cortex M4 chip inside, now ARM Cortex M4 is not only the ARM processor, not only a microcontroller, but lot of other things all integrated in the same chip.

(Refer Slide Time: 26:03)

Advantages of using microcontrollers

- Fast and effective
 - The architecture correlates closely with the problem being solved (control systems).
- Low cost / Low power
 - High level of system integration within one component.
 - Only a handful of components needed to create a working system.
- Compatibility
 - OpCodes and binaries are the SAME for all 80x51 / ARM / PIC variants.

swayam

To summarize the advantages of microcontrollers, they are fast, they are effective from the point of view of solving some particular application at hand. It must be low cost because if it is of a higher cost it also increases the cost of the product, it must consume low power, so you know your electric bill should not take a hit.

And compatibility is very important. Typically you will find that microcontrollers come from certain companies; you can say these are family of microcontrollers. PIC, ARM these are some typical examples. Now, if you look at the microcontrollers across the family they are all instruction set compatible so that once you develop software for one member of the family, and you move to a better processor, the same code can run or execute with very little modification. This is called compatibility and it is very important with respect to maintainability.

With this we come to the end of this lecture. In the next lecture we shall be starting some discussion on how we can make processing faster with particular regard to the ARM family of microcontrollers. We shall specifically see what are the features that are inside the ARM family of microcontrollers and why they are different, , why people are using them so widely.

Thank you.

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

Lecture – 04
Architecture of ARM Microcontroller (Part I)

In this lecture we shall be starting our discussion on the ARM microcontrollers. What are their architecture like, what are their specific features, and how are they different from the earlier generation of microcontrollers. The topic of this lecture is Architecture of ARM Microcontroller, this is the first part of the lecture.

(Refer Slide Time: 00:41)



In this lecture we shall be covering some general ideas about the ARM series of microcontrollers, how they have evolved and some of the important architectural features.

(Refer Slide Time: 00:57)

History of ARM Series of Microcontrollers

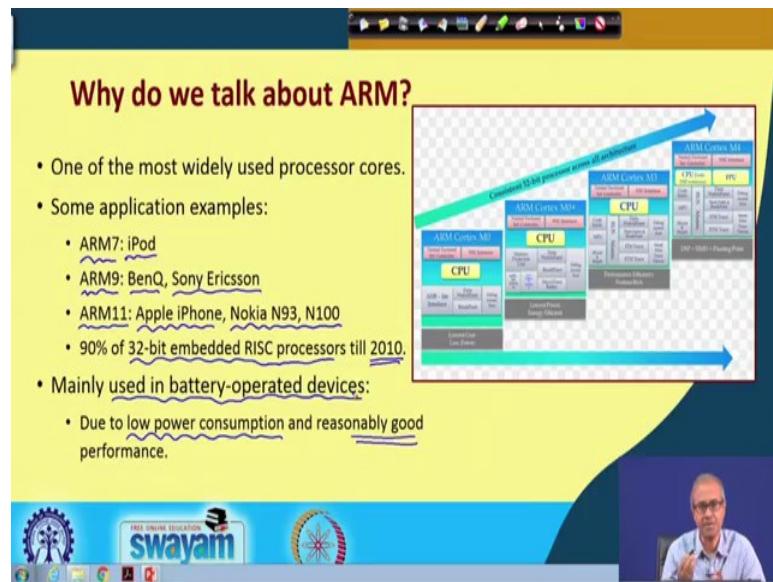
- Architectural ideas developed in 1983 by Acorn Computers.
 - To replace the 8-bit 6502 microprocessor in BBC computers.
 - The first commercial RISC implementation.
- The company founded in 1990
 - Advanced RISC Machine (ARM).
 - Initially owned by Acorn, Apple and VLSI.

Let us look into the history. The architectural ideas that have evolved into this ARM class of microcontrollers were developed long back in 1983. There was a company called Acorn computers that was the first to develop and evolve such ideas. Now these ideas were little different because they started to develop architectural ideas based on the RISC architecture concept. And at that time there was a very popular microprocessor called 6502 from a company called Mostek that was used in one of the very popular microcomputers called BBC micro.

The first attempt of these people was to replace that processor by a more powerful processor, which will make the BBC micro faster and more powerful. This resulted in the first commercial RISC implementation. It was not called ARM during that time, but it evolved into the ARM architecture. There was a company that finally got founded in 1990. The name ARM is the acronym for Advanced RISC Machine. So, you see in the name itself the word RISC is embedded.

ARM architecture essentially borrows concepts from the RISC architectural concept. Initially this company ARM was jointly formed and owned by Acorn which was the initiator, Apple was also there and there was another company called VLSI. These three companies came together and formed this new company called ARM.

(Refer Slide Time: 03:37)



Now what is so interesting about ARM? Why do we have to talk specifically about ARM? You may ask in this course why are we specifically trying to use ARM as the vehicle for teaching embedded systems. The reason is ARM has been increasingly used in many applications, they are the most popular category of microcontrollers that are seriously used in embedded system applications.

Let us take some examples that you all know about. The iPods from Apple through which you can listen to music, there was an ARM processor inside. Benq, Sony Ericsson these are very well known companies who manufacture TV sets and many audio visual equipments, there are ARM processors inside each of these equipments. Typically they started to use ARM 9, but subsequently they upgraded them to the later version of ARM processors. The Apple iPhone all of us are familiar with and some of the very popular Nokia phones all have ARM11 processor inside them.

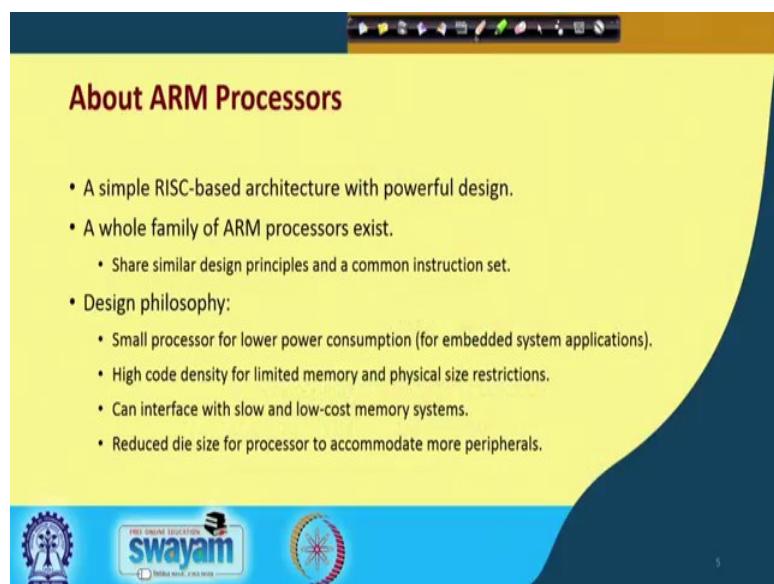
Now, this is a pretty old piece of statistics. Till 2010, 90% of all serious embedded applications had this kind of ARM processors inside them. When you talk about embedded system application, depending on the application you decide how much power you need from the processor. If it is a very simple application you do not need a processor as powerful as ARM. You can use 8-bit PIC microcontrollers.

ARM processors are typically 32 bit and above. So, you use ARM processor when you need reasonably powerful computation capability that will make the heart of your

embedded system. Now another thing is that ARM processors have very low power consumption and of course, reasonably good performance. Because of this low power consumption, they are very widely used in battery operated devices. There are many battery operated devices like the mobile phones.

If you look at this diagram, it shows an evolution of ARM based processors and ASICs over the years.

(Refer Slide Time: 08:09)



First thing is that, ARM is essentially a RISC based architecture. We shall go into some detail what a RISC architecture is. It borrows some advanced architectural ideas in contrast to conventional microcontrollers that had very primitive kind of designs.

I talked about 8051 that was a very popular microcontroller, but architecture wise it was pretty primitive, it did not use any kind of architectural enhancement or advanced features. As I told you this ARM processor is not just one, but a whole family of processors and the most important thing is that in order to maintain backward compatibility, all these share essentially a common instruction set.

Of course in the later generations some additional instructions have been added, but the older instructions are also carried through, such that a program that was developed for a older generation would run pretty well for the next generation also. Now the design

philosophy here was of course we need a small processor so that we can have lower power consumption and can be used for embedded systems application.

So, the size of the processor should be small, it should consume low power, and high code density. You see in microcontrollers I told that program memory and data memory are all inside the chip. There is a scarcity of real estate; you cannot put very large memory inside.

So, there is a maximum limit to the size of the program that you can run. Let us say the program memory size is 100 kilobytes. Whatever I write must fit within this 100 kilobytes. So, if my instruction set supports that in this 100 kilobytes, I can pack my code very nicely, so that I can implement more functionality as compared with some kind of competing architecture where a much more memory would be required to implement the same thing. What I mean to say is that, suppose I have an application X that I want to implement in a conventional architecture maybe it will be requiring 120 kilobytes but in RISC ARM Architecture I can fit it within 100 kilobytes.

This is something called high code density. There are some instruction features that allows us to reduce the number of instructions required. This can take advantage of limited memory and physical size restrictions. And of course, here there is lot of flexibility in the interface. We can interface with a wide variety of memory systems, very slow or also relatively. And of course, reduced die size means when you are actually fabricating the chip the size of that silicon is very small, so that when you develop that ASIC it would occupy a very small portion of it. So, you can use the remaining space to put in much more; with additional functionality to make the ASIC very powerful.

(Refer Slide Time: 12:40)

Popular ARM Architectures

- ARM7
 - 3 pipeline stages (fetch/decode/execute)
 - High code density / low power consumption
 - Most widely used for low-end systems
- ARM9
 - Compatible with ARM7
 - 5 stages (fetch/decode/execute/memory/write)
 - Separate instruction and data cache
- ARM10
 - 6-stages (fetch/issue/decode/execute/memory/write)

FREE ONLINE EDUCATION
swayam
India's first online education platform

Some of the popular ARM architectures are shown here. There are many, I am only showing three, ARM 7, 9, 10 there are 11 and beyond. ARM 7 has 3 pipeline stages, we shall be talking about pipeline later. Now pipeline stage essentially means how the instructions get executed.

There are 3 stages: fetch, decode, execute. It supports high code density and low power consumption. You do not need very high power ARM processors everywhere, whatever you need inside a mobile phone you will not need possibly inside a refrigerator, you need very simple kind of calculations there. Coming to ARM 9 first thing is that these are all backward compatible, but the pipeline stages are increased to 5; fetch, decode, execute, memory, write. And the concept of cache memory came in, and there is a separate instruction cache and separate data cache

In ARM 7 instruction and data were both in the same memory. So, it was like a von Neumann architecture, but from ARM 9 onwards the architecture started a shift towards Harvard architecture. Moving to ARM 10, the main difference was the pipeline was further enhanced by adding another stage called issue.

In this way the basic architecture started evolving making the processor more powerful and faster by adding novel architectural concepts.

(Refer Slide Time: 14:42)

The table compares the following parameters for four ARM family members:

	ARM 7 (1995)	ARM9 (1997)	ARM10 (1999)	ARM11 (2003)
Pipeline depth	3-stage	5-stage	6-stage	8-stage
Typical clock frequency (MHz)	80	150	260	335
Power (mW/MHz)	0.06	0.19	0.50	0.40
Throughput (MIPS/MHz)	0.97	1.1	1.3	1.2
Architecture	Non Neumann	Harvard	Harvard	Harvard
Multiplier	8 x 32	8 x 32	16 x 32	16 x 32

This table gives a quick comparison among 4 ARM family members ARM 7, 9, 10 and 11. You can also see the year when it was first introduced. First thing is pipeline depth; depth means how many stages of the pipeline are there..

So, we are enhancing the number of stages in the pipeline to make the execution faster in some sense. Sometimes the speed of a processor is determined by how fast we can make a clock. In ARM7 it was 80 MHz then 150, 260, 335 and so on. So, the clock frequencies are increasing, the processors are becoming faster. Power consumption is also a measure of the clock frequency, faster is the clock more will be the power consumption.

So, you should estimate the power consumption with respect to the clock frequency, because every microcontroller has a range of permissible clock frequencies. It depends what clock frequency you want, if you can operate with a lower clock frequency and serve your purpose it is fine, you will be consuming lower power. So, power consumption in microcontrollers is typically measured by milliwatt per megahertz.

In microcontrollers you measure throughput typically by million instructions per second per megahertz. ARM7 was based on von Neumann, but subsequently there is a move towards Harvard Architectures and inside the processor there is a built in multiplier.

There was an 8 x 32 multiply in the first two generations, whereas for the next two generation there was a 16 x 32 multiplier because many applications frequently require

multiplication operation. So, if there is a hardware multiplier built in, it speeds up operation quite significantly.

(Refer Slide Time: 18:13)

ARM is based on RISC Architecture

- RISC supports simple but powerful instructions that execute in a single cycle at high clock frequency.
- Major design features:
 - Instructions: reduced set / single cycle / fixed length
 - Pipeline: decode in one stage / no need for microcode
 - Registers: large number of general-purpose registers (GPRs)
 - Load/Store Architecture: data processing instructions work on registers only; load/store instructions to transfer data from/to memory.
- Now-a-days CISC machines also implement RISC concepts.

LOAD
STORE

FREE ONLINE EDUCATION
swayam

The point to note is that ARM is based on the RISC architecture. RISC is based on some architectural features.

These architectural features are like this. With respect to instructions, there is less number of instructions. Instructions are simple so that all instructions can be executed in a single cycle. They are all of fixed length so that decoding of the instruction becomes very easy, and the hardware for the controller becomes very simple. Then with respect to the pipeline here we shall see later that instructions are typically executed in a pipeline in all modern day processors. Now if the instructions are simple they are fixed length, then decoding of the instruction becomes very easy. You can decode in one stage itself, you do not have to again look at the instruction and try to find out what this instruction was.

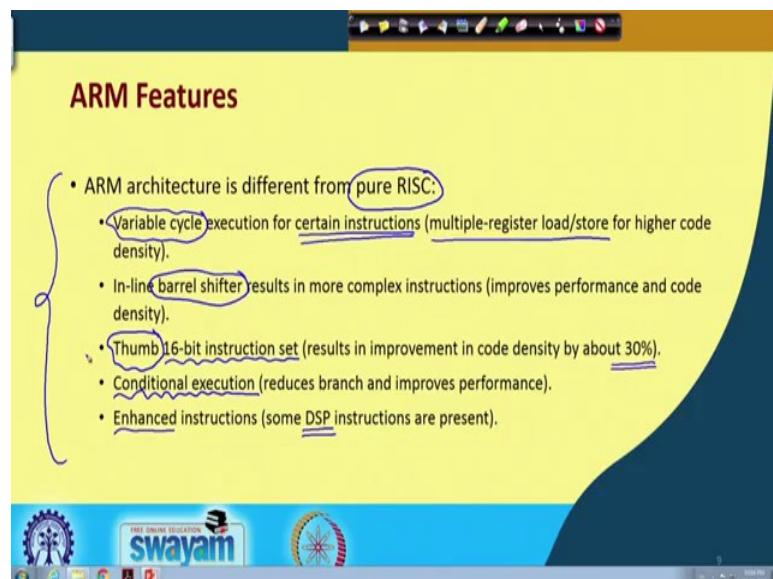
There is no need for microprogramming that is a standard norm for complex instruction set computers. You can directly implement the control unit in hardware, this also becomes much faster and you can run it at a higher clock. One characteristic of RISC architecture is that there is a very large number of general purpose registers, typically 32 or more. There are very few special purpose registers unlike CISC where there are lot of special purpose registers like program counters, stack pointer, base registers, and so on. And another important thing is that RISC is based on a load/store architecture, which

means there are some load and store instructions responsible for transferring data between registers and memory. All other instructions only work on registers, they do not access memory.

This kind of instruction set is sometimes called load store architecture, where only load and store instructions access memory and all other instructions work only on the registers. Now I told earlier that even the CISC machines of today like the Intel class of machines they use micro programming, they translate those complex instructions into some kind of microprograms that look more like the RISC instructions.

So, they also implement RISC concept in some way, they make an initial translation after which they execute using standard RISC techniques. Now talking about ARM, well although the name ARM contained this RISC this middle R is the acronym for RISC,

(Refer Slide Time: 22:13)



strictly speaking ARM is not a pure RISC architecture. There are some features that have been introduced in the architecture because they are very useful in embedded system applications, which are not RISC characteristics. Some of the differences are as follows. Certain instructions require variable number of clock cycles for execution. While talking of RISC, I said all instructions should be executed in a single clock cycle, but in ARM some of the instructions can be more complex, it can require multiple clocks.

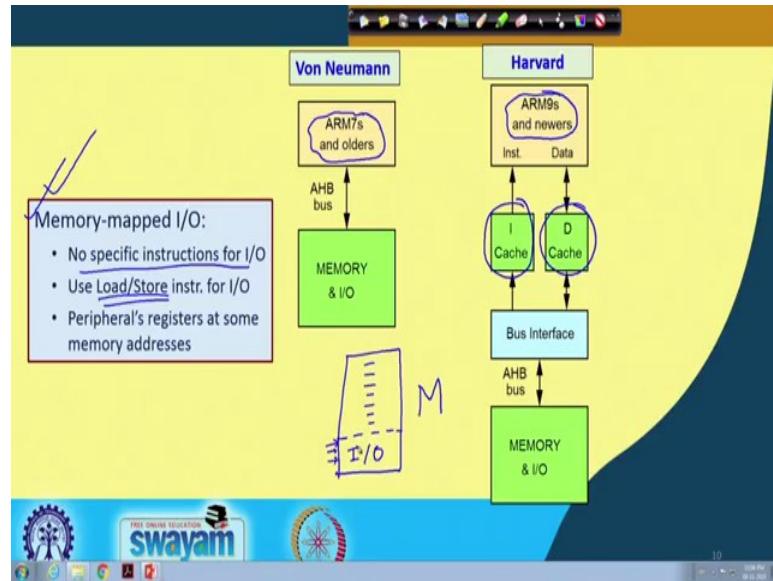
One classical example is multiple register load store. Normally we load a value from memory into a register, but ARM allows you to specify that the value loaded will be loaded into let us say 4 registers. So, to write into those 4 registers you need 4 clock pulses. You cannot do this in one cycle, such multiple data transfer instructions are supported in ARM. And there is something called a barrel shifter that is a very common architectural concept. It is a hardware which allows multiple bit shifting very efficiently in a single cycle. This barrel shifter is part of the ARM architecture and there are many instructions which directly utilize this barrel shifting capability. Let me take an example. Suppose there is an ADD instruction which adds 2 registers, let us say r2 and r3. It adds, but I can also say you add r2 and r3 shifted left by 4 positions. Shift left by 4 positions will be done by the barrel shifter, it will not take any additional time, in that single clock cycle everything can be done.

Because of the presence of the barrel shifter this kind of shift and operate kind of instructions are possible. And another feature is that you can configure ARM in the thumb mode. Thumb is a subset of the ARM instructions, which works in 16-bit mode. Normally ARM processors are 32-bit processors, but there may be many application where you do not need that power, you need much simpler power. You can have the thumb instruction set which is essentially a 16-bit instruction set.

If we use instructions that are smaller, this can further lead to a shortening of the total code size. Your code density can further improve. And there is another very interesting feature we shall be discussing this in detail, called conditional execution. You can say you add these 2 numbers provided the 0 flag is set. In conventional processors if the 0 flag is set you can have a JZ or JNZ kind of instruction, you do a jump then check if it is not 0, and so on; that means, you need so many jump instructions.

But if you have a conditional instruction, like you say add if 0 flag is set, then you are avoiding the jump instruction altogether. The number of instructions also get reduced. And of course, there are some enhancement like multiply and add, this kind of instructions have been added to the instruction set. Because of these ARM has deviated slightly from the pure RISC, but still it is a fairly powerful processor mostly based on RISC, only for a few cases it deviates because of very good reasons of course.

(Refer Slide Time: 27:14)

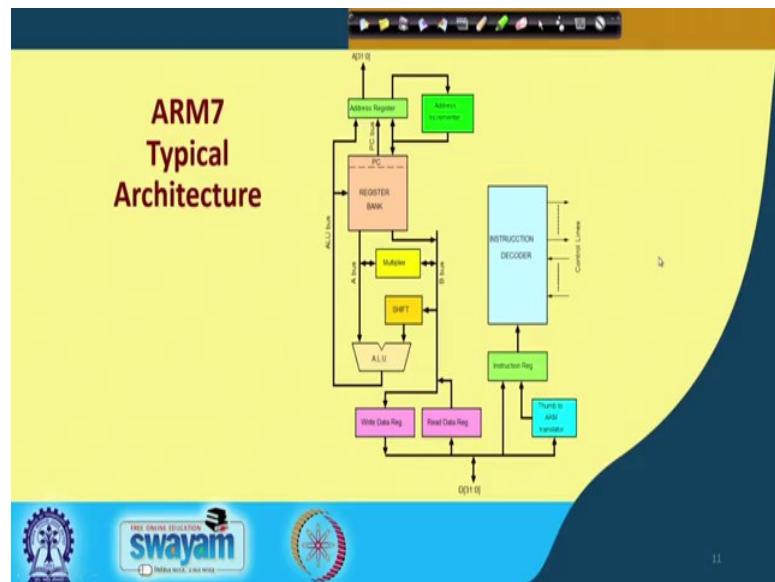


Now talking about this von Neumann and Harvard architecture you already talked about earlier, this ARM7 and the even older processors were based on von Neumann, there was a single memory. The later processors have 2 separate memory, instruction memory and data memory, and inside also there is an instruction cache and a data cache. Now another feature is that ARM processors does not have separate instructions for input/output, they use something called memory mapped IO.

Like let us say this is your total memory area, this is your memory. There is one part of memory that you reserve for the IO devices. Normally when you access memory you store the data here, but when you are trying to access some address in this region the decoding circuitry will automatically be accessing the IO ports instead of the memory.

But there will be the same decoder for memory and IO operation. Say load store instructions are typically used to transfer data between memory and register, the same load store instructions will be used for reading from IO port or writing into IO ports. There are no separate instructions for input and output. The address to be used will be the address corresponding to the IO devices.

(Refer Slide Time: 29:08)



This is the typical ARM architecture, I just wanted to show you a snapshot of it. You see you have the register bank, here we have the arithmetic logic unit. One of the data is coming directly from the register bank here, and for the other one you see there is a barrel shifter sitting here.

So, the other data can be shifted and then applied to ALU or it can even come without that, with no shifting. The multiplier is sitting here; whenever you need this multiplier hardware you can multiply and bring it to the a-bus. There are some other instruction features. There is an address register, address incrementer, so these are the address bus and here is the data bus for interfacing with memory.

But inside this is interesting, there is a multiplier, there is a barrel shifter that are sitting before the ALU. This makes the implementation of some of the instructions very efficient.

So, with this we come to the end of this lecture. We shall be continuing this discussion over the next couple of lectures where we shall be looking at some of the more additional features that are there in the ARM instruction set and also the ARM architectures.

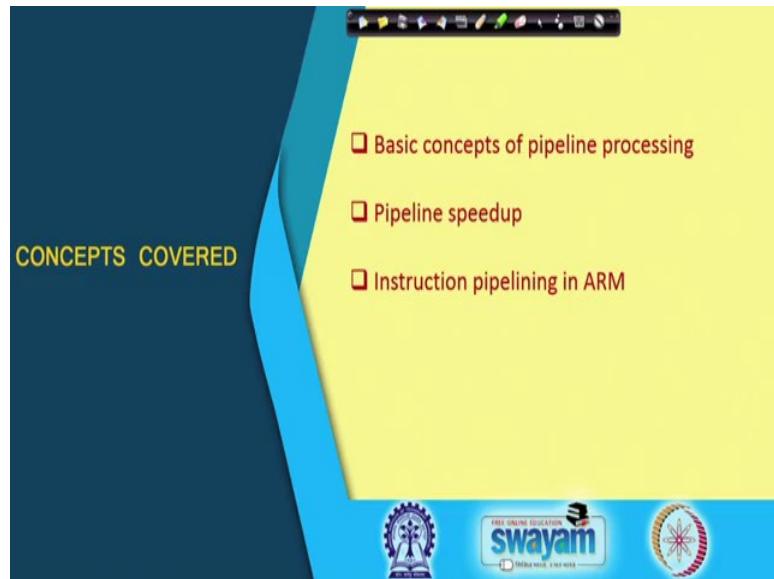
Thank you.

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

Lecture – 05
Architecture of ARM Microcontroller (Part II)

We continue the discussion on ARM microcontrollers.

(Refer Slide Time: 00:32)



In this lecture, we shall be mainly talking about the pipeline features that are present in the ARM processor. Because the concept of pipelining may be new to some of you, I shall be devoting some time in explaining the basic concept of pipeline. Then I shall very briefly tell how the pipeline in the ARM processor looks like, and what is expected to be gained out of it, why do use pipeline, what are the advantages that you have out of it.

(Refer Slide Time: 01:16)

What is Pipelining?

- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of k sub-computations (or stages).
 - Very nominal increase in the cost of implementation.
 - Very significant speedup (ideally, k).
- Where are pipelining used in a computer system?
 - ✓ **Instruction execution:** Several instructions executed in some sequence.
 - **Arithmetic computation:** Same operation carried out on several data sets.
 - **Memory access:** Several memory accesses to consecutive locations are made.

swayam

We start with the basic question what is pipelining? Basically pipelining is a mechanism for overlapped execution. When you say we are carrying out certain tasks, normally we do a task, complete it and then start with the next task. In pipelining the concept is that even before you finish the first task, we start with the second task, even before we finish the second task we start the third task. So, different parts of the tasks can be carried out in parallel in an overlapped fashion.

We partition a computation that is being carried out into k number of sub-computations or stages. Then it is possible to have very significant speed up, we shall see maximum up to k . But the very interesting thing is that we are not increasing the cost by a factor of k . Well, we can have k number of processors naturally we will be getting k times speed up. We are not doing that. By very nominal increase in cost we are achieving close to k speed up.

In the present context we are talking about instruction execution, but pipelining can be used very effectively in other domains of processing also, during arithmetic operations, during memory access of a vector of data, etc. But in the present context, since instruction execution is the only thing we are concerned about, we shall not be going to too much detail about the other ones.

(Refer Slide Time: 04:18)

A Real-life Example

- Suppose you have built a machine M that can wash (W), dry (D), and iron (R) clothes, one cloth at a time.
 - Total time required is T .
- As an alternative, we split the machine into three smaller machines M_W , M_D and M_R , which can perform the specific task only.
 - Time required by each of the smaller machines is $T/3$ (say).

For N clothes, time $T_1 = N \cdot T$

For N clothes, time $T_2 = (2 + N) \cdot T/3$

$$\frac{(2+N)T}{3} \approx \frac{NT}{3}$$

SWAYAM

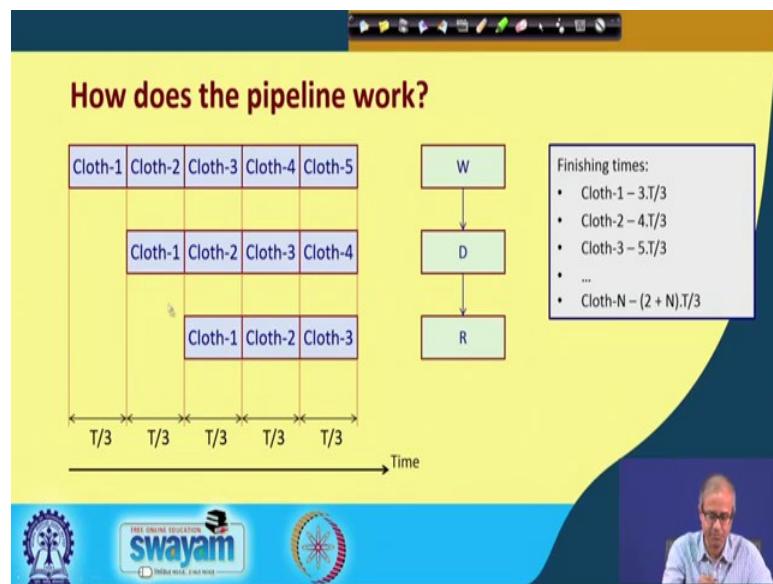
Let us understand first the concept. We take a real life example, which has nothing to do with computers. Suppose we want to wash some cloths. Suppose I have built a machine M that can do three things one by one, wash, dry and iron. I give the machine a cloth, it will wash it, dry it, iron it, and output that cloth in the ironed form. Let me assume that the total time required for the whole thing is T .

Pictorially I show it like this. I have my machine that does washing, drying and ironing, the total time taken is T . So, if I have a N number of clothes, then the total time required will be $N \times T$.

Now let us assume that instead of a single very complex machine that can do everything, let me divide this machine into three smaller pieces. Three simple machines one which can only wash, one can only dry, and one can only iron. Naturally this will not be costing three times as compared to the original machine, this will be cheaper. Let us make another assumption; the total time earlier was T , let us say for each of these time is $T/3$, $T/3$ and $T/3$, so that the total time still remains T .

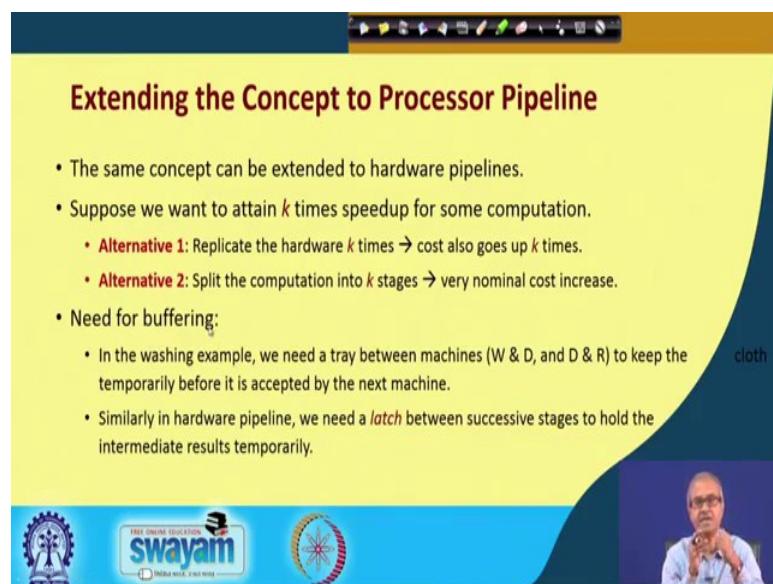
We shall be seeing in the next slide that for processing N number of clothes the total time required will be only $(2 + N) T / 3$. If N is very large, then this 2 can be neglected, and you can approximate it to $NT/3$. So, I have got a 3 time speed up, but I have not paid 3 times more, I am using simpler machines. This is the concept of pipeline.

(Refer Slide Time: 07:11)



Let us look at this diagram in an animated form. The first cloth comes for washing, this will be taking $T/3$ time. After it is finished with washing, this cloth can go for drying. And when cloth-1 has come for drying, machine W is free again. W can start with cloth-2. Next step, cloth-1 is coming for ironing, cloth-2 is coming for drying, cloth-3 for washing and so on. So, you see after all the machines are all filled up, every $T/3$ time one cloth will be coming out. Earlier one cloth was coming out every time T . So, I am getting a 3 times speed up effectively. This is the essential idea behind pipelining.

(Refer Slide Time: 08:44)

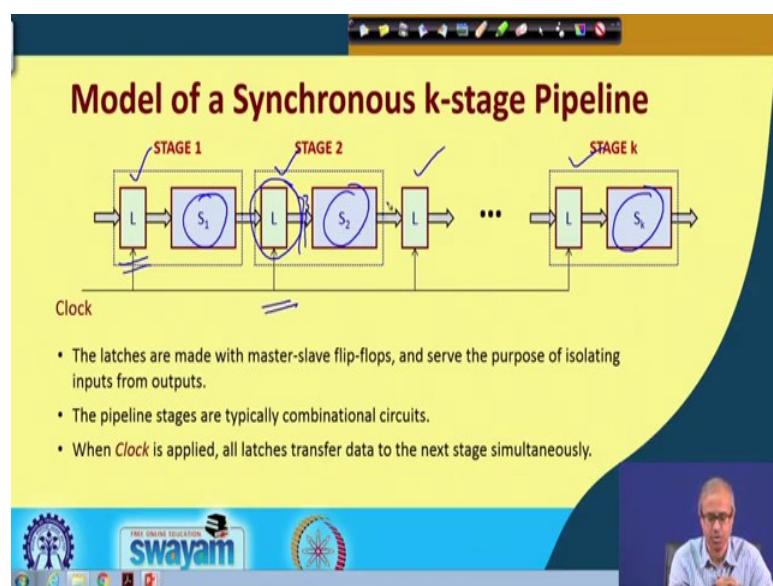


Extending the concept to processor pipeline, we want to increase the speed of a processor now. Suppose, I have a CPU and I want to make it k times faster. I have two alternatives. I can buy k copies of this CPU, and run k instructions in parallel. Alternatively, I can divide the execution into k -stages of pipeline. The instructions will be executing in a pipeline fashion just like that washing, drying, ironing I mentioned. There also you can expect a k times speedup.

In alternative 1 the cost will also go up k time, because you have to buy k copies of this CPU. Alternative 2 is to split the computation into k -stages; here you are not multiplying the hardware by k , rather the you are splitting it into k pieces resulting in very nominal increase in cost.

But in order do this, you need some buffering. Take the earlier example of clothes. When the first cloth washing is finished, you want to give it for drying. You must keep it somewhere in between, so that you can accept the next cloth for washing. There must be a buffer or a tray between the machines, these are something called buffering requirements. For a instruction pipeline also we need some registers or latches in between the stages, which will be temporary storing the result of the previous stage, which will be used by the next stage for processing. If you do not use this registers, then the previous stage can go and modify this value, so the next stage might carry out some wrong computation because of that.

(Refer Slide Time: 11:05)



A k-stage pipeline in general looks like this. There are k numbers of stages S₁, S₂ up to S_k with a latch or register in between every pair of stages. Whenever a stage completes its calculation, it will store the data into this latch, and it will take the data from the next computation into its input latch. If this latch was not there, then while the next calculation is going on this input will go on changing. So, this calculation can become wrong. This is how it works.

(Refer Slide Time: 11:51)

Speedup and Efficiency

Some notations:

- τ : clock period of the pipeline
- t_i : time delay of the circuitry in stage S_i
- d_L : delay of a latch

Maximum stage delay $\tau_m = \max \{t_i\}$

Thus, $\tau = \tau_m + d_L$

Pipeline frequency $f = 1/\tau$

- If one result is expected to come out of the pipeline every clock cycle, f will represent the maximum throughput of the pipeline.

FREE ONLINE EDUCATION SWAYAM

Let us carry out a quick calculation of the speedup and efficiency of a pipeline in the general sense. Earlier we showed the calculation for the washing example. Let us say τ is the clock period of the pipeline, which means, every τ time data moves from one stage to the other. And t_i is the time delay for the circuit that is there in stage S_i . And the latches delay will be d_L .

What will be the maximum stage delay, see this is t_1 , this is t_2 , this is t_3 . So, the slowest stage in the pipeline will determine what is the maximum speed with which we can shift the data, because this slowest stage will be become the bottleneck. So, $\max\{t_i\}$, let us call it τ_{max} , is the maximum delay. And to it we have to also add the latch delay d_L . So, $\tau_{\text{max}} + d_L$ will be your clock period τ .

Now, the pipeline frequency will be $1 / \tau$. f will also be the maximum throughput of the pipeline if you are expecting one result to come out every clock. With this assumption let us try to make a quick calculation.

(Refer Slide Time: 14:02)

• The total time to process N data sets is given by

$$T_k = \underbrace{[(k-1) + N] \cdot \tau}_{=} \quad \begin{array}{l} (k-1) \tau \text{ time required to fill the pipeline} \\ 1 \text{ result every } \tau \text{ time after that} \rightarrow \text{total } N \cdot \tau \end{array}$$

• For an equivalent non-pipelined processor (i.e. one stage), the total time is

$$T_1 = \underbrace{N \cdot k \cdot \tau}_{=} \quad \text{(ignoring the latch overheads)}$$

• Speedup of the k -stage pipeline over equivalent non-pipelined processor:

$$S_k = \frac{T_1}{T_k} = \frac{N \cdot k \cdot \tau}{k \cdot \tau + (N-1) \cdot \tau} = \frac{N \cdot k}{k + (N-1)}$$

As $N \rightarrow \infty$, $S_k \rightarrow k$

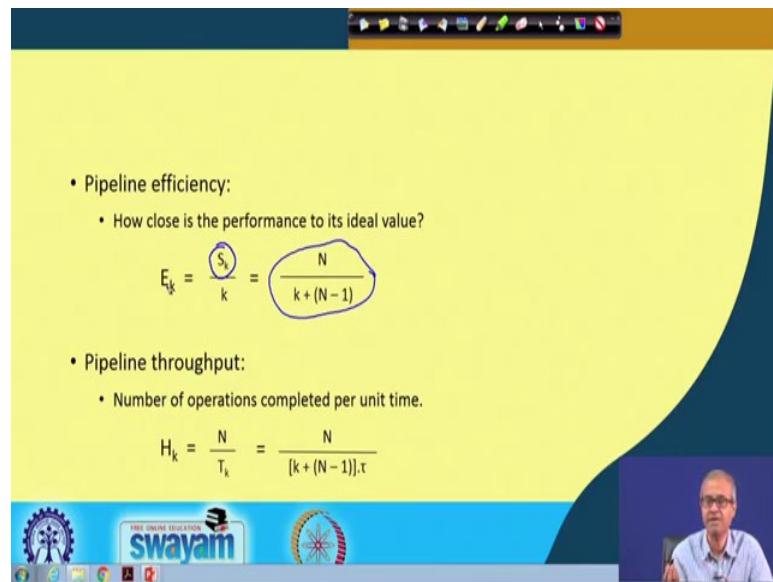
swayam



We calculate the total time to process N sets of data. τ is my clock period. Now, $k - 1$ clocks are required to fill up the pipeline. There are k -stages, so I need $k - 1$ clocks to reach a stage where all the k -stages are working on something. After that every τ time there will be one new result being generated. So, $(k - 1) \times \tau$ will be the initial time for the pipe to fill up, and then this $N \times \tau$ for the outputs to be generated. This will be the total time to process N data sets.

Now, if we have an equivalent non-pipelined processor, if you ignore the latch delays for the time being, then the total time can be estimated as $N \times k \times \tau$. So, in a pipeline the speedup S_k we are getting is this. As N becomes very large this S_k approaches k . So, for a large number of data that you are processing the pipeline, your speedup will be close to number of stages k . This is an important result.

(Refer Slide Time: 16:21)



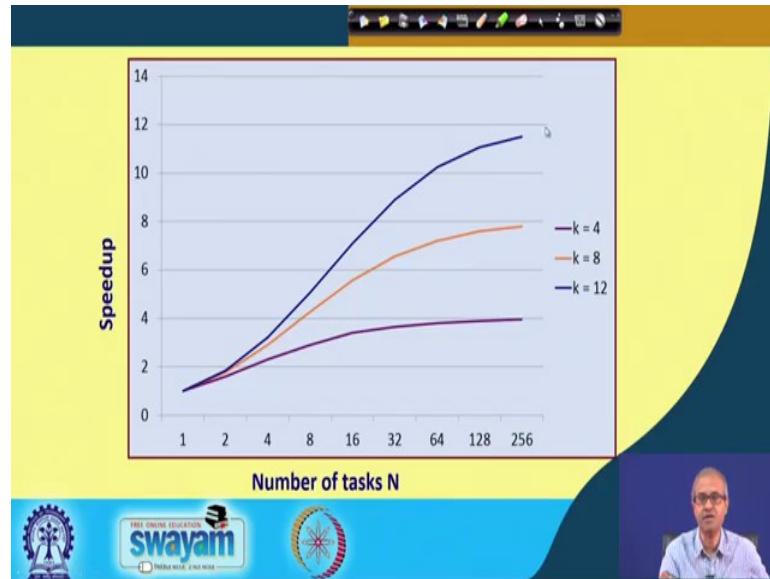
- Pipeline efficiency:
 - How close is the performance to its ideal value?

$$E_k = \frac{S_k}{k} = \frac{N}{[k + (N - 1)]}$$

Now, there is another term we define called pipeline efficiency; how much is the performance close to the ideal value. Well, this S_k we just calculated will tend to k , so that is when the pipeline is operated at maximum efficiency. If I divide by that, k , k cancels out, so I get a factor. This I can define as the actual pipeline efficiency. So, it will never be 100%, maybe it is working in 90% efficiency.

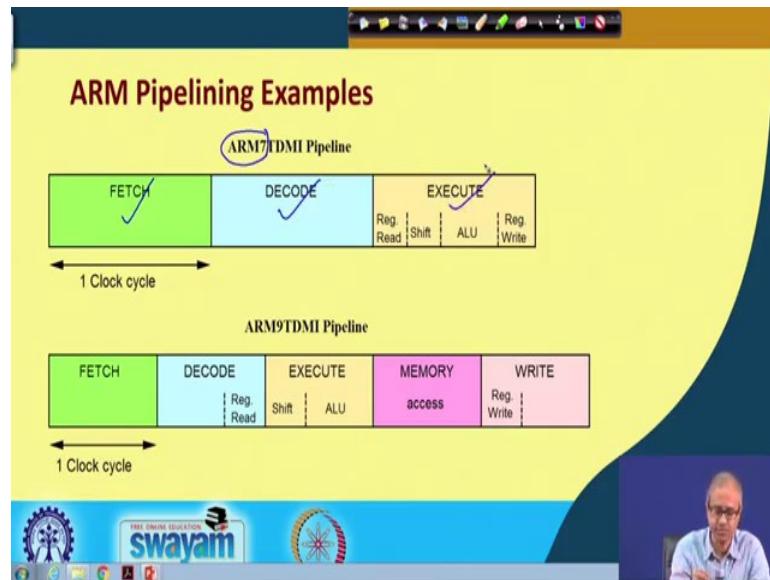
And another term which is of course is not that important in the present context is called pipeline throughput; the number of operations completed per unit time. Total number of operation is N . And the time taken is T_k . If you divide it, you get an expression, this is pipeline throughput.

(Refer Slide Time: 17:48)



This is a very typical plot I am showing, number of task N versus speedup for various values of k. Let us say k = 4; you see as the number of tasks increases, the speedup increases increase and levels to very close to 4. For k = 8, it levels to very close to 8; and so on. Here I have shown up to 256. You get some idea what is actually happening.

(Refer Slide Time: 18:35)

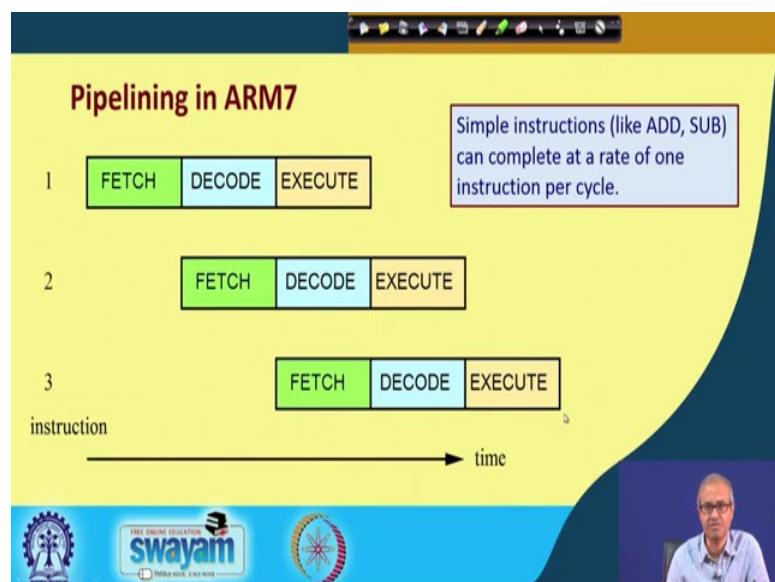


Now, coming to ARM I am not going into the details because this is not a course where I am teaching computer architecture rather I am trying to tell you that ARM uses instruction pipelining. If I have a k-stage pipeline, I can expect to have k times speedup.

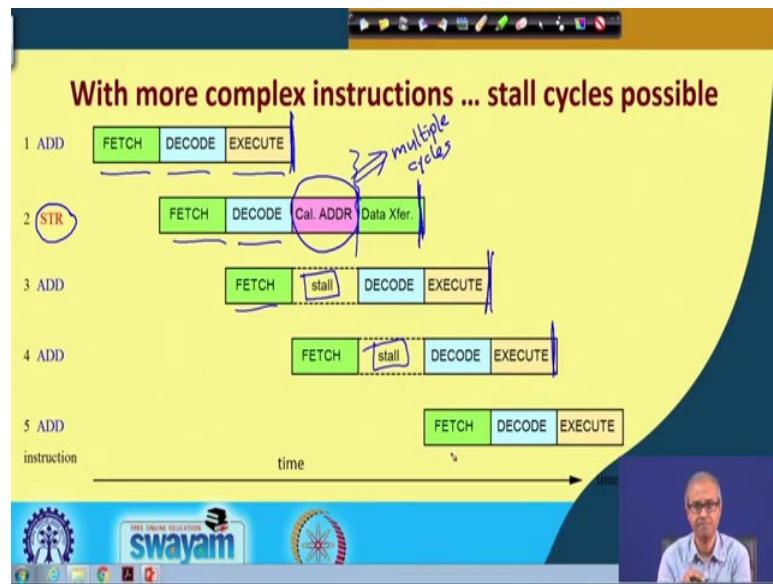
In ARM7 architecture this is one particular processor TDMI. There are three stages, fetch, decode and execute. If everything else is fine, we are expected to get about 3 times speedup in terms of instruction execution.

Similarly ARM9 has a 5 stage pipeline, fetch, decode, execute, memory, write. We can see some smaller things. Within the decode, the register values are also read whatever registers are required. During execute the barrel shifter is also working, ALU operations also carried out, memory access are carried out here; during write, the results written back into the register bank, so it is done here. And for ARM7 all of these were done during execute. The register read, shift, ALU, register write everything was done in execute. But in ARM 9, you are making the pipeline more elaborate and more flexible. So, here the speedup can be maximum 5.

(Refer Slide Time: 20:33)



(Refer Slide Time: 21:15)



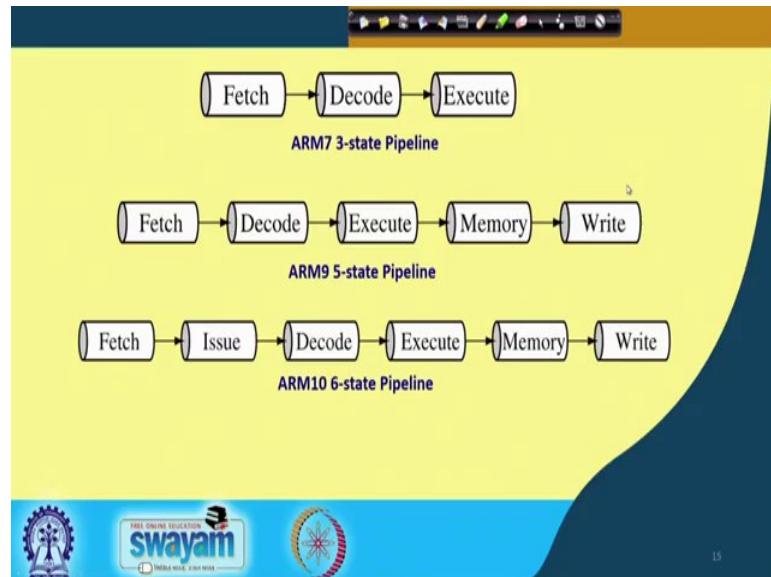
Just one thing I want to mention here is that this speedup of k that I am talking about is an ideal speed up. It is the speedup you can get when the pipeline is operating in its full speed, but sometimes due to some reason, you cannot operate the pipeline at full speed. In those cases, the speedup will become less than k . I am giving one example. Suppose, there are some instructions that are executed and let us say these are all ADD instructions, and there is one complex multi register store instruction.. So, it will need multiple clock cycles.

The idea is that normally everything finishes in one clock. But for STR instruction what might happen that this pink colored box can actually require multiple cycles. What does that mean? Multiple cycle means here you cannot decode this next instruction, unless this execute is over you cannot decode it. So, there will be some delay here because it is requiring multiple cycles. Such delays are referred to as stalls; we call them stall cycles.

Stall cycle means some cycles are wasted. You see here first instruction is finished; second instruction was finished here, third instruction here, fourth instruction here. But because of this delay this instruction was supposed to finish here, but it got delayed. Not only this, all subsequent instructions got delayed and there can be many such instructions like this in between. So, for every such instruction there will be some stall cycle inserted. And once a stall is there this stall will be carried by all subsequent instructions until that

instruction exits the pipe. Such cases can slow down the maximum operational speed of a pipeline.

(Refer Slide Time: 24:07)

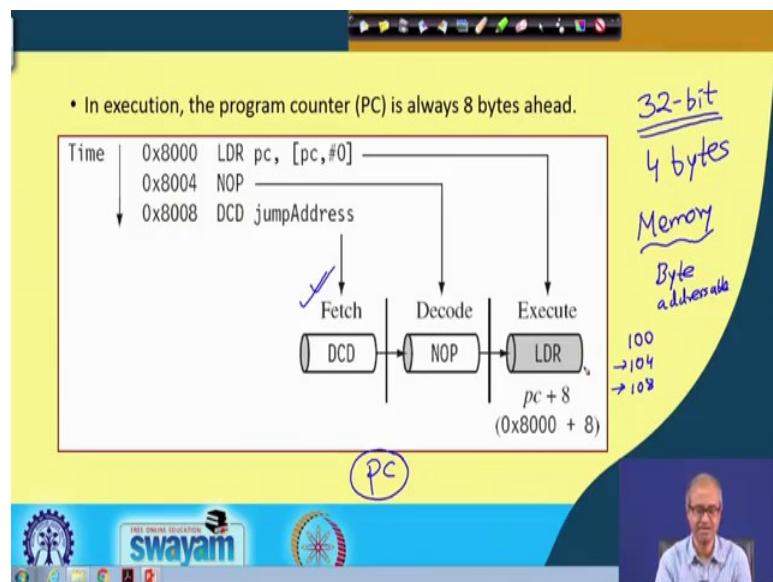


We give a bird's eye side view of the ARM7, ARM9 and ARM10 pipelines.

There are something called pipeline hazards. There can be data dependency whether you can feed the next instruction or not. There are a lot of architectural issues that can prevent the pipeline from operating at its maximum speed.

Stall instructions can occur due to this, one example I gave because of a complex instructions, but there can be other reasons. There are situations called data hazards, there can be structural hazards, there can be control hazards. Because of various sequence of operations that are being carried out, and some instructions like jumps and branches you may have to insert stall cycles. All of these prevent the pipeline from operating at the maximum clock frequency.

(Refer Slide Time: 25:39)



Just another thing let me tell you, in this ARM7 kind of architecture a 3-stage pipeline is there. Let us say when an instruction reaches the execute phase, one thing you remember I told you all instructions are 32-bit instructions. And your memory is byte addressable. So, if the first instruction is stored in memory location 100 the next instruction will be stored in memory location 104, next location will be stored in location 108, because each instruction will be requiring 4 bytes.

The point is that when some instruction is executed the PC will always be 8 bytes ahead. You add 4 to it because you will be fetching this. Each instruction will be adding 4 to the memory address. And PC is a special register which always stores the address of the next instruction to be fetched. When you are fetching this instruction, this will be the PC of the current instruction plus 8, because current instruction is here. If we add 4 to it, we will be getting the next instruction; if we add 8 to it, we will be the next to next instruction. Here you are always fetching the next to next instruction because there are three stages that is why the PC is always 8 bytes ahead.

Because when you are executing here already incremented PC two times it is always 8 bytes ahead, so that when you are fetching you should accordingly adjust and fetch accordingly. With this we come to the end of this lecture. In the next lecture, we shall be looking at some of the unique features of ARM with respect to register organization, the various execution modes and so on. This we shall be discussing in the next lecture.

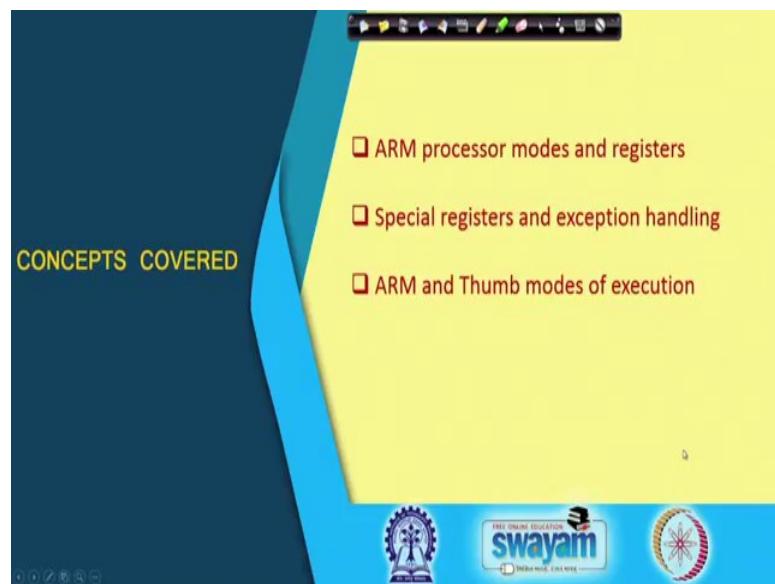
Thank you.

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

Lecture – 06
Architecture of ARM Microcontroller (Part III)

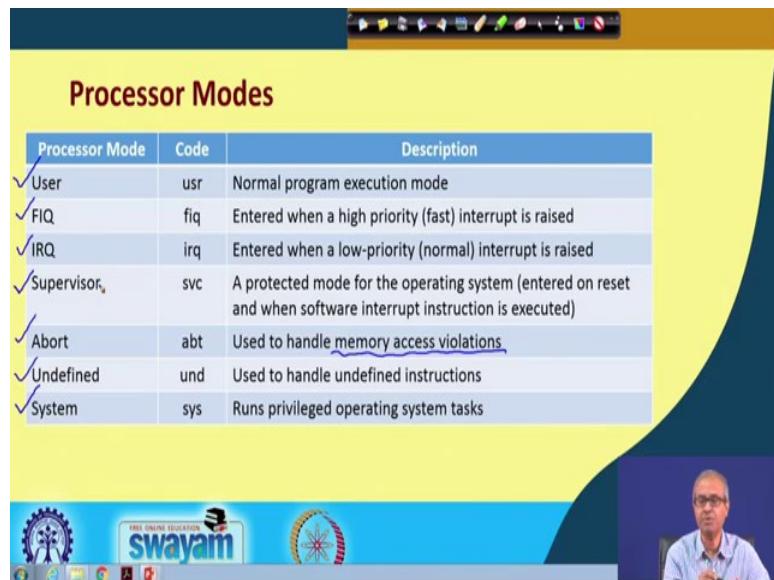
We continue with our discussion. In this lecture we shall be talking about the Architecture of ARM Microcontroller, the third part of it.

(Refer Slide Time: 00:30)



In this lecture we shall be mainly talking about the processor modes and registers in the ARM architecture. There are some special registers, we shall be trying to tell you the necessity and roles of these special register; something about exception handling and there is something called thumb mode of execution also very briefly about that. Again we shall not be going into very much detail of this because this course is primarily meant for a hands on demonstration for designing various systems, but learning some basic concepts on the architectural issues will always help you in becoming a better designer. This is the main purpose of trying to give you with some of the initial backgrounds of embedded system design in the architectural concepts.

(Refer Slide Time: 01:33)



Processor Mode	Code	Description
✓User	usr	Normal program execution mode
✓FIQ	fiq	Entered when a high priority (fast) interrupt is raised
✓IRQ	irq	Entered when a low-priority (normal) interrupt is raised
✓Supervisor	svc	A protected mode for the operating system (entered on reset and when software interrupt instruction is executed)
✓Abort	abt	Used to handle <u>memory access violations</u>
✓Undefined	und	Used to handle undefined instructions
✓System	sys	Runs privileged operating system tasks

Let us start with the processor modes. The ARM processor during execution at a given time, can be in one of 7 modes. This table summarizes these 7 processor modes.

The user mode is the normal mode during execution of a program. When a program is executed normally, we say that the system is in user mode. This user mode acronym is usr. Now you know in processors, we can have interrupts coming from external devices. If it comes, the program that is executing will be suspended, we will have to go to some interrupt handling routine, run that routine and then again come back and resume the interrupted program. In ARM two different levels of interrupt processing are permissible or allowed, one is called high priority or other is called normal priority.

This FIQ is the high priority mode. The FIQ mode is entered when a high priority interrupt is activated and is acknowledged. The point to note is that when a high priority interrupt is being processed, if some lower priority interrupt comes in the meantime; they will not be acknowledged, they will be ignored. So, higher priority interrupt will be having real higher priority over the lower priority ones.

And IRQ is the low priority or the normal priority interrupts.

There is a supervisory mode which most of the modern processors have, there are some instructions like supervisory call or trap or SWI software interrupt. There are various names, but the purpose of this instructions is that, these instructions allow to transfer

control to the operating system. Well in a computer where there is an operating system, these instructions allow the processor mode to be changed from user mode to supervisor mode and then just like a subroutine call control will jump to the supervisor or the operating system.

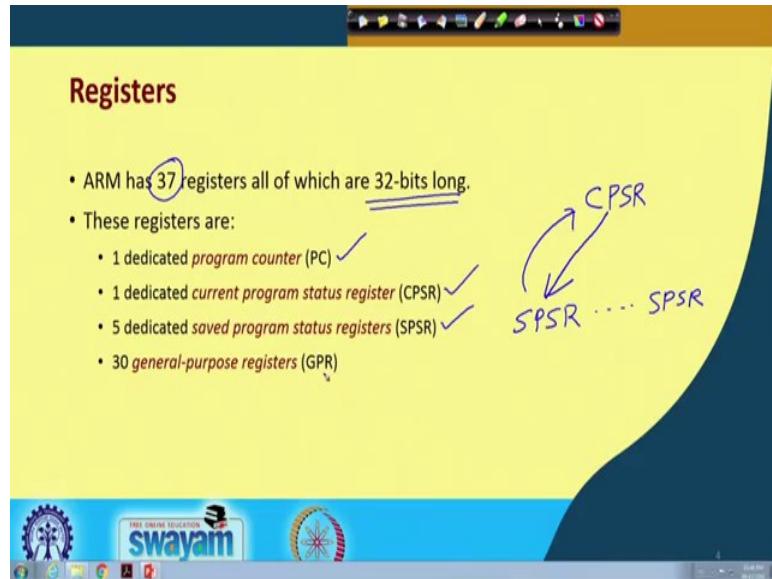
Now, the supervisory mode is svc. This is a protected mode and this mode is required only when there is an operating system in your implementation. In all embedded system application you will not require this, but in system where there is really an operating system and you need some protection you need to have this mode of execution.

And abort is an optional mode for cases where you want to have memory protection. You want to see that when a program is executing, you are supposed to access only this region of memory. If accidentally your program tries to access any memory location beyond the permissible limits, this abort interrupt will be generated and the corresponding processor mode is called the abort mode; for handling memory access violations the processor goes to the abort mode.

And undefined is actually something which is kept for future expansion. Well some instruction opcodes have not been defined, so they are undefined. So, if by mistake you are trying to execute an instruction whose opcode is undefined, you get an undefined interrupt and the processor mode goes to undefined state und.

And there are instances where you want to run some privileged operating system tasks, and for that you typically go to this system mode. The system mode and supervisory mode are very much related. I am not going into detail of this because for an embedded system design, these are not really required. ARM supports all these modes. ARM is quite flexible in terms of its architectural features. It has a large set of general purpose registers as I mentioned earlier.

(Refer Slide Time: 07:25)



In total there are 37 registers. All of these are 32 bit long. Talking about dedicated registers, there are 7 registers which have specific purpose; one is the PC. Then there is CPSR. For those of you who are familiar with the some microprocessors like 8085, you will know that there are something called condition flags. Whenever some instructions are executed the condition flags are set or reset; there is zero flag, carry flag, overflow flag, and so on. This is like a flag register. It consists of several flags that are set depending on the mode of the processor or the instruction execution.

And there are 5 saved program status register. Now this concept is interesting. There is one CPSR and there are several SPSR. You see in a conventional processor whenever an interrupt comes, what do we do normally? We save all the registers in the stack that can include also the status registers and the flags, go to the interrupt handler, finish everything, come back, restore all registers and status register and resume execution, but in ARM there is no support for stack, there is no instruction to push or pop instructions in stack or from stack.

Suppose a program is running current status is stored in CPSR, there is an interrupt that has come. The content of the CPSR will get copied into an SPSR, then interrupt handler will run, before coming back the content of the SPSR will be restored back into CPSR. Since there is no stack it is the users' responsibility to restore the value from the CPSR

and then again move it back and forth. And of course, there are 30 general purpose registers; these are named typically r0 to r29.

(Refer Slide Time: 10:39)

Registers (contd.)

- The current processor mode governs which of several register sets is accessible.
- Only 16 registers are visible to a specific mode of operation. Each mode can access:
 - A particular set of registers (r0-r12)
 - r13 (SP, stack pointer)
 - r14 (LR, link register)
 - r15 (PC, program counter)
 - Current program status register (CPSR)
- Privileged modes (except System) can also access a particular SPSR.

PC → LR

FREE ONLINE EDUCATION **swayam**

Now, depending on the processor mode, it depends how many of these registers you can actually access. 16 registers are typically visible in a specific mode of operation; these 16 registers are as follows.

There are 13 general purpose registers r 0 to r12. There is no stack, but r13 is earmarked as the stack pointer. If you want to implement a stack yourself by software, you can use r13 for the purpose. And r14 is a link register. For subroutine call instructions for a conventional processor, the value of PC is pushed in the stack, but here there is no stack. So, there is a special register called the link register, the value of the PC gets copied into the link register. Whenever there is a subroutine call and when you are returning back from the subroutine, whatever is then the link register is copied back into PC.

PC is accessed as r15. And then we CPSR. I shall show a picture later with the different registers, how they exist in different modes.

(Refer Slide Time: 12:29)

The slide is titled "General-purpose Registers" in a dark red font. Below the title is a bulleted list of facts about ARM registers:

- 6 data types are supported (signed/unsigned)
 - 8-bit byte, 16-bit half-word, 32-bit word
- All ARM operations are 32-bit.
 - Shorter data types are only supported by data transfer operations.

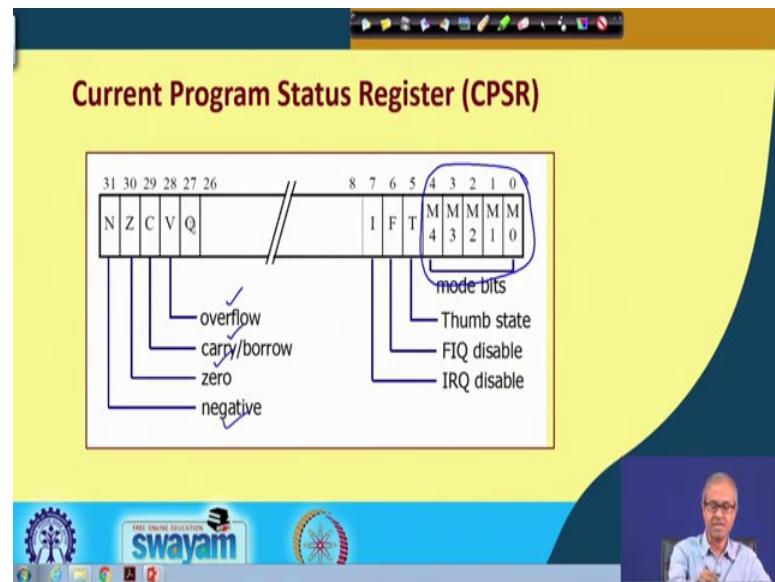
Below the list is a diagram of a 32-bit register. The register is divided into four fields: 31-24, 23-16, 15-8, and 7-0. The 7-0 field is labeled "8-bit Byte", the 15-8 field is labeled "16-bit Half word", and the entire 32-bit width is labeled "32-bit word".

The Swayam logo is visible at the bottom left, and a small video window shows a speaker at the bottom right.

In terms of the general purpose registers where you are expected to store some data, I told you the registers are all 32 bits in size. But in terms of the operations which are supported, you can have 8-bit operations, 16-bit or half word operations, or full 32-bit word operations.

When you are using half word operations, the lower 16 bits is used, and for byte operations the last 8 bits are used. And these are not used for arithmetic operations, only for data transfer operations because all arithmetic operations are only 32 bits in size. But when you are transferring data from one place to another, you can transfer 8 bits or 16 bits or 32 bits. From memory you can transfer 8 bits of data into a register, it will go into the last 8 bits, etc.

(Refer Slide Time: 13:58)



What does CPSR contain? It contains various flags. There is an overflow flag denoted by V, there is a carry flag C for subtract operation; you call it the borrow, there is a zero flag Z, it checks whether the result is zero or nonzero, and the sign flag N. In addition the processor can be in many modes.

I said there are 7 modes, but to keep with future expansion ARM keeps 5 bits to store mode. In 5 bits you can support up to 32 modes. Bit 5 is used to denote thumb state. Then bits 6 and 7 are for enabling and disabling the high priority interrupt and the normal prior to interrupt. If these bits are set to 1, these are disabled; if there is 0, they are enabled. The other bits in between are unused.

(Refer Slide Time: 15:26)

The slide has a yellow header and a blue footer. The header contains the title 'Special Registers'. The main content is a bulleted list of special registers:

- **PC (r15)**: Any instruction with PC as its destination register is a program branch.
- **LR (r14)**: Saves a copy of PC when executing the BL instruction (subroutine call) or when jumping to an exception or an interrupt handler.
 - It is copied back to PC on return from those routines.
- **SP (r13)**: There is no stack in the ARM architecture.
 - R13 is reserved as a pointer for the software-managed stack.
- **CPSR**: Holds the visible status register.
- **SPSR**: Holds a copy of the previous status register while executing exception or interrupt handler routines.
 - Copied back to CPSR on return from exception or interrupt.

The footer features the 'swayam' logo and a small video window of a speaker.

Talking about the special register, just a relook again. This is r15 or the program counter. As you know PC always holds the address of the next instruction in memory to be executed right. So, if the destination register is PC it means you are jumping there. Link register is r14 used for subroutine call instruction.

In ARM the subroutine call instruction is called BL, branch and link. It specifies the address where to branch. Before branching, the current value of the PC will be saved into LR (r14) and then the destination which is specified will be loaded into PC. When you are coming back, the value of LR will be copied back into PC. So you resume execution from where you left.

r13 can be used as stack pointer. If you want to implement stack in software, you can do it.

And CPSR holds the current status register with respect to the program that is being executed, and whenever some exception or interrupt comes, the CPSR gets copied into one of the special or saved program status registers SPSRs. They will hold a copy of the CPSR. When you come back from the exception routine the SPSR has to be copied back into CPSR before you resume execution.

(Refer Slide Time: 17:45)

Program Counter

- When the processor is executing in **ARM mode**:
 - All instructions are 32-bits wide, and must be word aligned.
 - The last two bits of PC are zero (i.e. not used).
 - Due to pipelining, PC points 8 bytes ahead of the current instruction, or 12 bytes ahead if the current instruction includes a register-specified shift.
- When the processor is executing in **Thumb mode**:
 - All instructions are 16-bits wide, and are half-word aligned.
 - The last bit of the PC is zero (i.e. not used).

Handwritten notes on the slide:
All instructions must start from an address that is a multiple of 4.
00
multiple of 2
0

Talking about PC, there are two modes of execution; one is the ARM mode, one is the Thumb mode. Let us make the distinction clear here. We first talk about the ARM mode of execution which is the default mode. ARM mode says that all instructions are 32 bit wide and their word aligned. Word aligned means all instructions must start from a memory address that is some multiple of 4.

The need for doing this I am not going into detail. When you interface memory with the microcontroller, there is a concept called memory interleaving. If you have 4-way interleaving, then 4 consecutive bytes starting from an address that is a multiple of 4 can be transferred in a single clock cycle. But if it is not so, you will be requiring 2 clock cycles. So, your memory transfer will become slower. To have faster memory access, ARM insists on word alignment and multiple of 4 means the last two bits are 00. Any address with the last two bits 0 means it is a multiple of 4. Therefore, when it is in ARM mode, the PC is expected to hold the address of our instruction, always the last 2 bits will be 0.

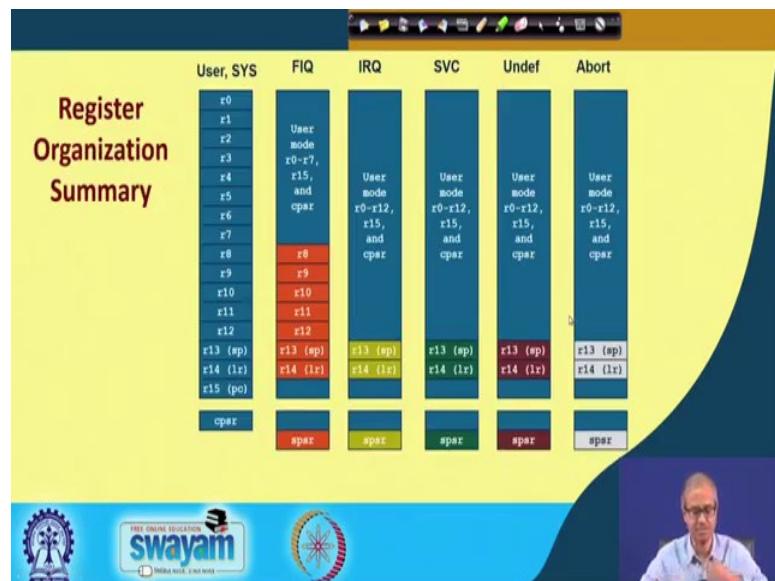
So, the last two bits of PC are actually not used in the ARM mode, as they will always be 0. Also, PC can point to 8 bytes ahead or 12 depending on the number of pipeline stages.

But there is another mode of execution when you do not need the full power of 32 bit processing. Maybe we are using the ARM processor in a very simple application, where smaller 16 bit instructions are there, some instruction subset. Here in the Thumb mode,

the instructions are 16-bit wide; that means, 2 bytes and they are half word aligned means the instruction addresses are multiple of 2. So, earlier it was multiple of 4, here it is multiple of 2. Multiple of 2 means the last bit of the address will be 0.

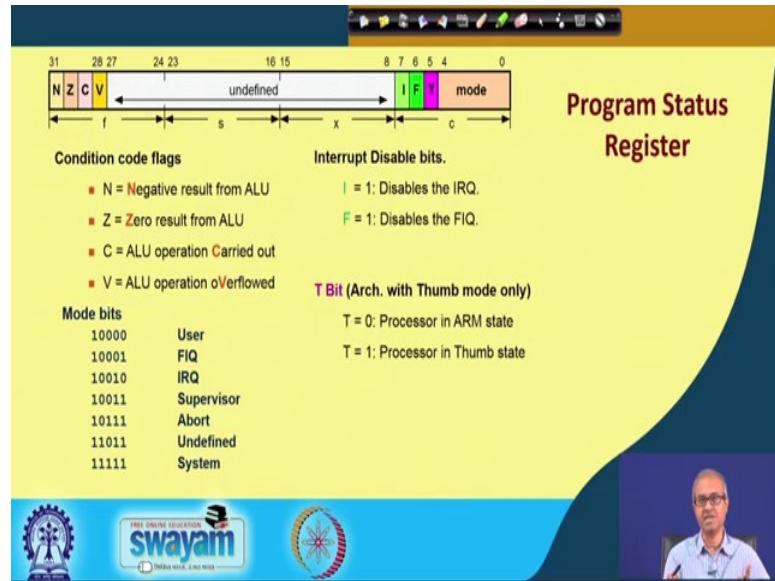
So, in the PC, the last bit is 0 which is not used. This is the main difference between the ARM mode and the Thumb mode. ARM mode is a superset where all instruction are encoding in 32 bits, Thumb mode is a subset of that where you make all the instructions shorter in 16 bits because of which code density will be higher. For small applications, you can have much cheaper mode of implementation. So, Thumb mode is used for such cases where you need small systems.

(Refer Slide Time: 21:50)



This diagram gives you an overview about the registers in the different modes. In the in the user or the system mode, you have access to the 13 registers r0 to r12, then r13, r14, r5, spv, link, PC, CPSR. The 4 SPSRs which are there, they correspond to the other 5 modes. If interrupt FIQ comes then CPSR gets copied into this SPSR and this r0 to r7 will be there in FIQ, but the other registers will be specific to a FIQ. If you come back, they will again change and in IRQ r0 to r12 the whole thing is there, only r13 r14 are specific to FIQ. Similar for SVC and abort.

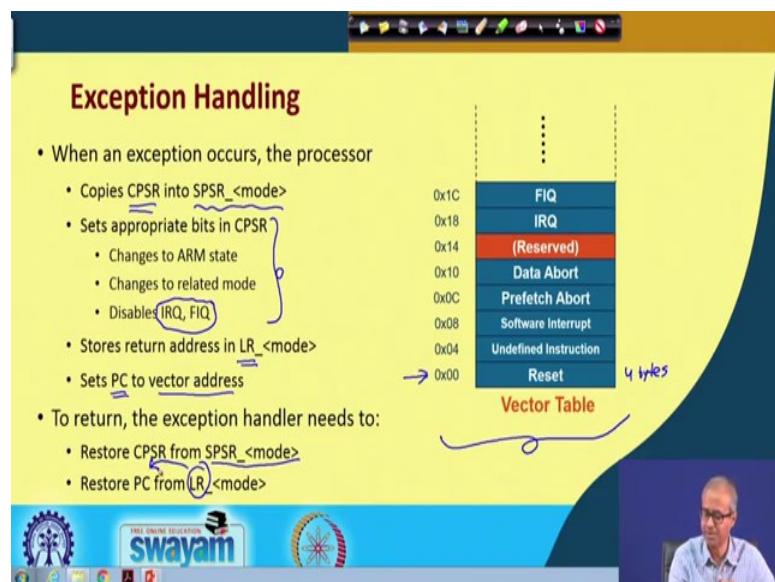
(Refer Slide Time: 23:36)



Now, talking about the CPSR once more this is again the picture of the CPSR I told earlier. The last 5 bits, now the specific bit combinations are shown here. For these 7 modes, the mode bit combinations are defined like this 10000 the means user mode, 10001 is FIQ mode, and so on.

The other flag bits are also shown.

(Refer Slide Time: 24:38)



Talking about exception handling, a process like this is followed; there is an interrupt vector kind of a table called vector table. This vector table is present from address 0,

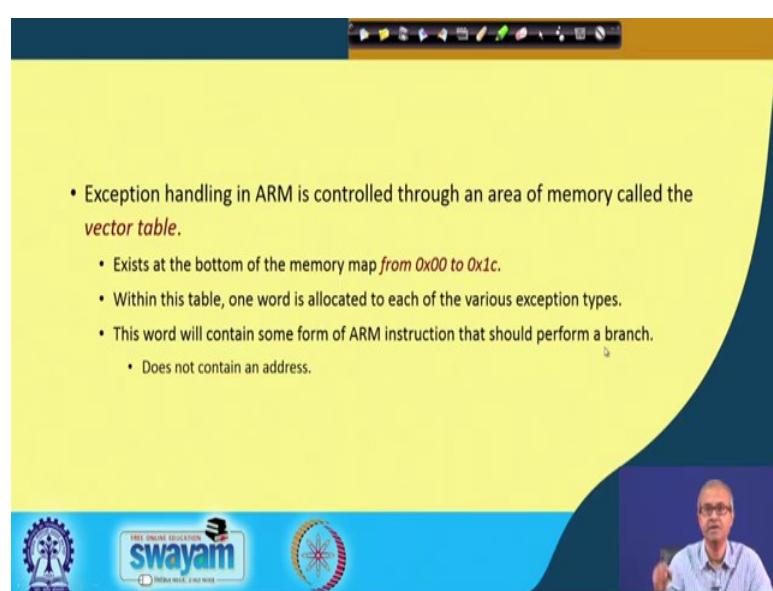
0x00 means hexadecimal 00; these are hexadecimal numbers 004, 008, 00C. Each of these entries is 4 bytes.

Whenever any exception occurs, the CPSR will be copied into the corresponding SPSR. You know there are 5 SPSRs depending on the type of exception CPSR will be copied to the corresponding SPSR.

And the corresponding bits of the CPSR will be set depending on which mode you are in. And during interrupt processing, the interrupt bits will be disabled because if they are enabled then another interrupt may come in between also. In order to avoid nested interrupt from coming while you are doing the interrupt handling, interrupt can be disabled. Then before coming back the return address will have to store in LR register corresponding to that mode, there are separate LR registers for the different modes and PC is loaded with the vector address. If you jump to this, for return the exception handler will do the reverse thing. It will be loading back that SPSR into CPSR, LR will be transferred to PC.

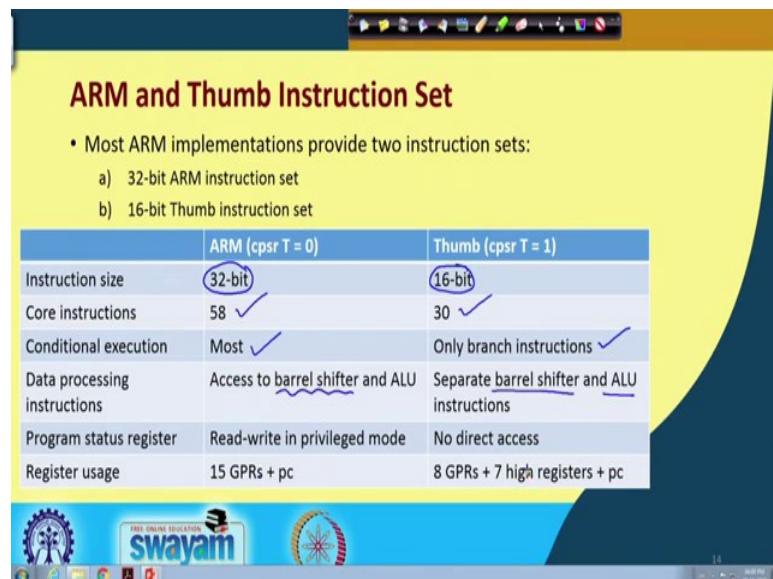
The point to notice is that entries out here, these are not addresses of interrupt handler rather these are instructions. Typically these are jump or call kind of instructions. So, when you come here, there will be a jump from here to the routine which is handling the interrupt. So, these are all one word instructions are stored in the vector table.

(Refer Slide Time: 28:58)



This is what I have already mentioned. The vector table is stored from address 0x00 to 0x1c. One word is allocated for every exception type and as I have said, this word will contain some instruction; some jump, some branch instruction. It will branch to the corresponding interrupt handler. It does not contain any address.

(Refer Slide Time: 28:27)



ARM and Thumb Instruction Set

- Most ARM implementations provide two instruction sets:
 - 32-bit ARM instruction set
 - 16-bit Thumb instruction set

	ARM (cpsr T = 0)	Thumb (cpsr T = 1)
Instruction size	32-bit	16-bit
Core instructions	58 ✓	30 ✓
Conditional execution	Most ✓	Only branch instructions ✓
Data processing instructions	Access to barrel shifter and ALU	Separate barrel shifter and ALU instructions
Program status register	Read-write in privileged mode	No direct access
Register usage	15 GPRs + pc	8 GPRs + 7 high registers + pc

FREE ONLINE EDUCATION **swayam**

I am talking about the ARM and Thumb instruction set, a quick comparison. In ARM mode the T bit in the CPSR is set to 0 and for Thumb mode set to 1. The differences are as follows. In ARM mode, there are 32 bit instructions; in Thumb mode there are 16 bit instructions. The number of main instructions are 58 and 30. Conditional execution is a feature where some instruction will be executed depending on whether some condition is true or false.

In ARM almost all the instructions support condition execution, but in Thumb only for branch instruction you can check for conditions, but other instruction do not support conditional execution. Data processing instructions in ARM mode supports barrel shifting. But in Thumb such instructions are not there, here there are separate shift instructions, and there are separate ALU instructions.

Program status register in ARM mode, you can do read write in privileged mode, but in Thumb mode you cannot access program status register. And for registers in ARM mode, you can use the 15 GPR r0 to r14 and the PC, but in Thumb you can use the 8 GPRs, 7

high registers and PC. High register means the high order 16 bits of the registers. There are some specific registers which can be accessed.

(Refer Slide Time: 31:01)

The slide has a yellow header with the title 'What is Conditional Execution?' in red. The main content is a bulleted list:

- It controls whether or not the CPU will execute an instruction.
- Most instructions have a condition attribute that determines whether it will be executed based on the status of the condition flags.
 - Prior to execution, the processor compares the condition attribute with the condition flags in CPSR.
 - If they do not match, the instruction is not executed.
- Example:
 $MOVEQ \text{ r1},\#0$ (if zero flag is set, then $r1 = 0$)
- The condition attribute (here *EQ*) is suffixed to the instruction mnemonic.

At the bottom, there is a logo for 'swayam' and a small video window showing a person speaking.

Conditional execution, let me take an example here. Conditional execution controls whether or not CPU will execute the current instruction. Now most instruction in ARM allows a condition attribute to be specified. Let me take an example instruction like this. Normally if you do not use this EQ, just imagine this EQ is not there, then $MOV r1,\#0$ means, $r1$ is initialized to 0. Now if I give this $MOVEQ$, this means I am checking this zero flag in the CPSR. If the zero flag is set, then only you do the move, otherwise you do not do the move.

So, this instruction will be executed depending on certain condition. This is how the conditional executions execute. We shall see later that using this kind of condition execution allows us to prevent branch instructions in many cases. It makes the code density higher, with less number of instructions.

With all this, we stop our brief discussion on the ARM architectures. From the next lecture onwards, we shall be moving on to some aspects about the ARM instruction set and other features of ARM that will be helpful in the actual experimentation that we shall be showing. All the demonstrations that we shall be showing would be based on some ARM board, and also a few experiments we shall be showing you on Arduino boards.

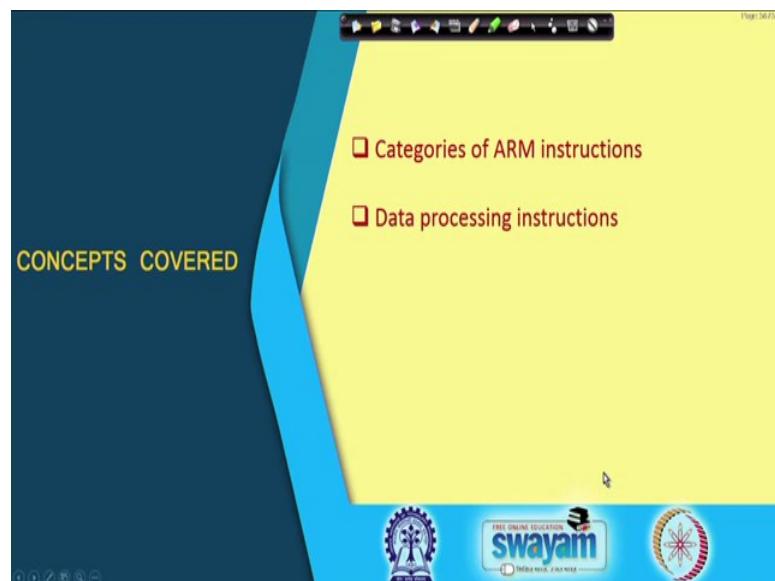
Thank you.

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

Lecture – 07
ARM Instruction Set (Part I)

We shall now start a very short discussion on the ARM instruction set. In the later part of this course, we shall be showing you lot of demonstration on ARM based and Arduino based boards. There we shall be writing our programs in a high level language, but here in this and a couple of other lectures we shall be discussing about the low level features of the ARM processor, and for that we need to have a basic idea about the assembly language features that ARM architecture provides.

(Refer Slide Time: 01:07)



Here we shall be broadly talking about the various categories of ARM instructions and in particular the data processing instructions that are provided by the ARM instruction set architecture.

(Refer Slide Time: 01:27)

The ARM Instruction Set

- ARM instruction can be categorized into three groups:
 - a) Data processing instructions
 - Operate on values in registers
 - b) Data transfer instructions
 - Move values between registers and memory
 - c) Control flow instructions
 - Change the value of the program counter (PC)

Registers

PC

Program Memory

Data Memory

Talking about the ARM instruction set, you see as a developer you need to develop or design an embedded system, you will be having a hardware platform, a microcontroller based system and you have to write software for it.

Today most of us develop the software in some high level language, either in C or in Python or in Java, but it is always good to know the lower level features of the processor in order to have an informed decision that which processor may be better for a particular application. With this motivation we are giving you a brief introduction to the assembly language features of the ARM processor that is a reflection of the hardware features that are provided by the ARM platform.

Broadly speaking any instruction set can be categorized into various groups or broad categories. In ARM let us define these categories as data processing, data transfer and control flow.

Most of the modern microcontrollers are based on the Harvard architecture, where you will be having a separate program memory and a separate data memory. The instructions that constitute the program will be stored in the program memory, while all temporary data that will be stored in the data memory. Among the registers this PC is one register which will be pointing to the address of the next instruction in the program memory. Whenever a new instruction is brought or fetched from memory it will be fetched from the address which is stored in the PC.

When I am talking about the data processing instructions we are talking about carrying out various arithmetic and logic operations on data. Now, ARM is based on the RISC philosophy of architectures, most of the ARM features are RISC based. Here all data processing instructions operate only on registers; that means, when I say add we will be adding the contents of two registers and storing the result back into another register. So, memory is not coming into the picture here at all.

Now, when we talk about data transfer there are options. We can transfer from one register to another or we can transfer from register to data memory or data memory to register. These options are all there, and these will constitute data transfer instructions. And, control flow instructions are those that will alter the sequence of execution of a program. Normally as I said PC will be pointing to the next instruction to be executed, as the instructions are executing the value of the PC gets incremented. But, whenever you have some instructions like jump or a subroutine call, suddenly that sequence will be disturbed, and you will be going to some other address and from there you will be starting to fetch your instructions.

This is what is meant by change of control flow, and such instructions are termed as control flow instructions. Essentially control flow instructions are those that will be altering the value of PC in some way so that the normal sequence of instruction execution will be altered.

(Refer Slide Time: 06:17)

The slide has a yellow header bar with the text '(a) Data Processing Instructions'. The main content area is yellow and contains a bulleted list of instructions. A blue footer bar at the bottom features the 'SWAYAM' logo and other navigation icons. A video feed of a man is visible in the bottom right corner of the slide area.

- All operands are 32-bits in size:
 - Either registers
 - Or literals (immediate values) specified as part of the instruction
- The result, if any, is also 32-bit in size and goes into a specified register.
 - One exception: long multiply, that generates 64-bit results.
- All operand and result registers are independently specified as part of the instruction.

We first talked about the data processing instructions that are present in the ARM instruction set. Now, in the ARM instruction set the first thing to notice that the registers are all 32-bit in size, the ALU inside the processor also has the capability of operating on 32-bit numbers. So, all operands that are being operated on are 32-bits in size. You may carry out these operations on registers, or some of the operands may be constants -- these are called literals or immediate values. I may say add the value 10 to the content of a register. That 10 is like a constant that is called a literal or an immediate value, that value 10 is specified as part of the instruction and is called immediate operand.

The result after the calculation is also a 32-bit result and this will be stored in some particular register. There is of course one exception; there is a special multiply instruction where the result is stored as a 64-bit number. Now, you know whenever we multiply two n-bit numbers the result can be $2n$ bits. So, when you multiply two 32-bit numbers the result can be maximum 64-bit in size. There is a version of multiply instruction where the result can be stored in two registers or 64-bits.

In these lectures we shall not be discussing about the instruction encoding; that means, exactly how the bits of the instruction word specify the registers; we shall not be going into that detail of the architecture, but essentially what kind of instructions are supported those we shall be discussing.

(Refer Slide Time: 08:55)

Page 63/70

- Arithmetic instructions:

ADD r0,r1,r2 ; r0 = r1 + r2	ADC r0,r1,r2 ; r0 = r1 + r2 + C (C is carry bit)
SUB r0,r1,r2 ; r0 = r1 - r2	SBC r0,r1,r2 ; r0 = r1 - r2 + C - 1
RSB r0,r1,r2 ; r0 = <u>r2 - r1</u>	<u>SUB r0,r2,r1</u>
RSC r0,r1,r2 ; r0 = <u>r2 - r1 + C - 1</u>	

- All operations can be viewed as either unsigned or 2's complement signed.
 - Means the same thing.

First let us talk about the arithmetic instructions. The most basic instructions are addition and subtraction. ARM provides with various addition and subtraction instruction alternatives. Let us see what kind of instructions are there. First is a simple add instruction, well here as an example I have shown three registers r0, r1, r2, but you can have any registers here not necessarily only r0, r1, r2. The first one represents the destination, the second and third indicates the two source operands.

So, when I write ADD r0, r1, r2 the values of r1 and r2 will be added and the result will be stored in r0. There is a version add with carry that is normally used to handle multi precision arithmetic. When you are adding two 64-bit numbers, you add first two 32-bit numbers, if there is a carry the next 32-bit numbers you would be adding with that carry. So, you should have an ADC version of instruction, this is add with carry.

Then there is a normal SUB instruction this is $r1 - r2$, result is going into r0. There is a similar borrow concept in subtraction for multi-precision arithmetic again. There is an BBC version where the carry flag is used. The actual calculation is done like this: $r1 - r2 + C - 1$. This takes care of the borrow during subtraction operation. Now, there are two variation of the subtract instructions, where the role of the operands are reversed. Like in SUB instruction we are saying $r1 - r2$. Now, if I say RSB this stands for reverse subtract this means $r2 - r1$. Similarly, there is a version with borrow, reverse subtract with carry; it is $r2 - r1 + C - 1$.

You may ask why you need two kinds of subtract instruction. I can always write this instruction as SUB r0, r2, r1. This ARM instruction allows the second operand to be specified in more flexible ways. If you have a reverse version of subtract then that flexible operand can be subtracted from or the normal data. We shall be seeing what kind of flexibility the second operand provides you, but this reverse subtract allows you to add a normal operand from a flexible operand, or from a flexible operand you can subtract a normal operand, both options are provided.

And, in these subtract instructions you are viewing these 32-bit numbers as either unsigned or as 2's complement signed numbers. Now, with respect to addition and subtraction in binary these two make no difference. The way the arithmetic is carried out is exactly identical.

(Refer Slide Time: 13:21)

• Bit-wise logical instructions:

- ✓ **AND** $r0, r1, r2$; $r0 = r1 \text{ and } r2$
- ✓ **ORR** $r0, r1, r2$; $r0 = r1 \text{ or } r2$
- ✓ **EOR** $r0, r1, r2$; $r0 = r1 \text{ xor } r2$
- ✓ **BIC** $r0, r1, r2$; $r0 = \underline{\underline{r1}} \text{ and } \underline{\underline{\text{not } r2}}$

• BIC is the acronym for "bit clear"

- Each 1-bit in r2 clears the corresponding bit in r1.

Diagram illustrating the BIC operation:

- Register $r1$ (4 bits): $\boxed{\quad \quad \quad \quad}$
- Register $r2$ (4 bits): $\boxed{\quad \quad \quad \quad}$
- Result Register $r0$ (4 bits): $\boxed{\quad \quad \quad \quad}$ (with some bits marked with = and not)

swayam

There are some bitwise logical instructions. There are instructions like AND, ORR and EOR that performs bitwise logical operation on the specified operands.

There is another instruction also that is supported at the bitwise level this is called bit clear instruction, in short BIC. The actual meaning is the register r1 is ANDed with the complement of r2; you take a NOT operation and then do an AND. So, that in r2 whichever bit is 1, if you do NOT those bits will become 0; if you do AND with r1 those corresponding bits of r1 will become 0.

So, wherever in r2 you have 1 those corresponding bits in r1 will be made 0; that means those will be cleared. So, this is also called a bit clear instruction.

(Refer Slide Time: 15:33)

• Register-register move instructions:

- ✓ **MOV** r0, r2 ; r0 = r2
- ✓ **MVN** r0, r2 ; r0 = not r2

• MVN is the acronym for "move negated"

- Each 1-bit in r2 clears the corresponding bit in r0.

• In the instruction encoding, the first operand r1 is not specified, as these are unary operations.

Then you have some register to register move instructions. This also we are defining under the data manipulation category. Here there are two kind of register to register move that are supported, one is a simple register to register move. You see when I am saying move register to register, I need to specify only two operands. When I say MOV r0,r2 it means the value of r2 is copied into r0. There is another version move negated; that means, if I say MVN r0,r2 here this 2 will be complemented and will be moved into r0.

There are many applications where negative value of our 32-bit number is required. There you can use this MVN instruction. Now, here in the encoding I said you do not need three registers. This r1 which we are mentioning earlier, the middle operand that is not required here, only two operands are required.

(Refer Slide Time: 17:13)

The slide is titled 'Comparison instructions' and contains the following text and handwritten notes:

- Comparison instructions:
 - ✓ **CMP** r1,r2 ; set cc on (r1 - r2)
 - ✓ **CMN** r1,r2 ; set cc on (r1 + r2)
 - ✓ **TST** r1,r2 ; set cc on (r1 and r2)
 - ✓ **TEQ** r1,r2 ; set cc on (r1 xor r2)
- All these instructions affect the condition codes (N, Z, C, V) in the current program status register (CPSR).
- These instructions do not produce result in any register (r0).

Handwritten notes on the right side of the slide:

- Z
- S
- V
- $0 \oplus 0 = 0 \}$
- $1 \oplus 1 = 0 \}$
- $0 \oplus 1 = 1 \}$
- $1 \oplus 0 = 1 \}$

The slide also features a logo for 'swayam' and a photo of a man in the bottom right corner.

There are a variety of compare instructions. You recall in the CPSR that we discussed in our previous lectures there are some condition flags. You recall there were Z flag, there were a N flag, there was a V flag and so on. These flags actually keep track of the results of some arithmetic or logic operation. When you add two numbers these flags will keep track of the fact whether the result was 0, whether the result was positive or negative, whether there was an overflow that took place because of that operation, etc.

Sometimes you just need to compare two numbers and take a decision. You are actually not doing addition or subtraction and storing the results somewhere, just you need to compare two values. There are a host of compare instructions available. Like you can say **CMP r1,r2**; that means, internally you are doing $r1 - r2$ and you are checking whether result is 0, negative or positive, accordingly the flags will be set.

There is another version compare negative; actually it means you do $r1 + r2$ and then set the flags. Here these instructions do not produce any result in a register; this is what you should remember. It only sets the condition flag so that you can use that result later depending on the condition flag.

Similarly, there is test kind of a comparison. This comparison means you are doing logical and bitwise AND of $r1$ and $r2$; then you are checking the result is 0 or nonzero and then accordingly set the flags. Similarly, you test for equality, **TEQ**. Equality means

exclusive or; if you take the exclusive OR of 0 and 0 the result is 0, if you take exclusive OR of 1 and 1, that is also 0, but if you take exclusive OR of 0 and 1 or 1 and 0, this is 1.

So, the result of an exclusive OR will tell you whether the bits are equal or not equal. So, if you take XOR of entire 32-bit numbers and see the result is 0, this means that the two numbers are equal, all the bits are giving XOR value of 0, that is why the result is 0.

(Refer Slide Time: 20:53)

The slide is titled 'Specifying immediate operands' and contains the following content:

- Notations:**
 - # indicates immediate value
 - & indicates hexadecimal notation
- Allowed immediate values:**
 - 0 to 255 (8 bits), rotated by any number of bit positions that is multiple of 2.

Handwritten notes on the slide include:

- ADD r1, r2 (#2) ; r1 = r2 + 2
- SUB r3, r3 (#1) ; r3 = r3 - 1
- AND r6, r4, #60f ; r6 = r4[3:0]

Handwritten binary values are shown: 000F and 1111.

The Swayam logo is visible at the bottom of the slide.

There are ways of specifying immediate operands. Like as I said you can specify some immediate data, I can write like this with this hash symbol. I can use the second operand as an immediate operand. The ampersand indicates that the number I am specifying is in hexadecimal.

With respect to the immediate values there is some restriction on the maximum value of this number you can represent. The range is typically 0 to 255, you can also regard it as a 2's complement number you can give a negative value also. And there is another facility, there are additional 4 bits in the instruction where you can specify that these operand value will be rotated and means within that 32-bit range these 8 bits can be positioned either here or here or here or here. You can make your number as per your requirement and then you can do ADD, SUB, etc.

(Refer Slide Time: 23:33)

The slide is titled 'Shifted register operands' and contains the following content:

- The second source operand may be shifted either by a constant number of bit positions, or by a register-specified number of bit positions.

```
ADD r1, r2, r3, LSL #3 ; r1 = r2 + (r3 << 3)  
ADD r1, r2, r3, LSL r5 ; r1 = r2 + (r3 << r5)
```

- Various shift and rotate options:
 - LSL: logical shift left
 - LSR: logical shift right
 - ROR: rotate right
 - RRX: rotate right extended by 1 bit

Diagram illustrating shift and rotate options:

- LSL: arithmetic shift left (represented by a circular arrow pointing left)
- ASR: arithmetic shift right (represented by a circular arrow pointing right)
- RRX: rotate right extended by 1 bit (represented by a circular arrow pointing right with a small 'C' indicating carry)

The slide is part of a presentation on the Swayam platform, featuring a video of a speaker on the right.

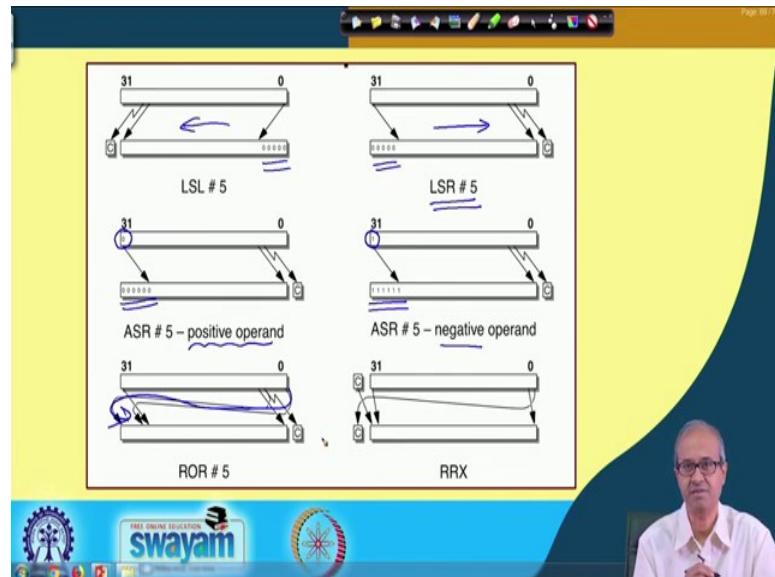
I talked about some flexibility in the second operand, you see here this shifted register operand comes in. You recall in the architecture I told in ARM there is a barrel shifter. The second operand optionally goes through the barrel shifter, you can shift it and then you can do some arithmetic or logic operations. In the assembly language level you can specify like this: ADD r1, r2, r3, LSL #3. In the fourth parameter you say logical shift left by 3; that means the second operand is shifted left by three positions and then added to r2. You can specify this as a constant or you can also specify some register, logical shift left by r5; whatever is the value in r5 that many bits will be shifted.

You see here we have a facility of specifying the second operand in a flexible way. Either in the normal form or in some shifted form that is why this reverse subtract facility is also there.

And, not only logical shift left, there are various shift and rotate instructions or options available; logical shift left, logical shift right, rotate right. Rotate right means when you shift right the last bit coming out will again get inside the register. Rotate right extended means the register you are rotating right, then there will be your carry flag. This bit will go into this carry flag and carry flag will get rotated in it, that is why this is something like a 33-bit rotation, this is called extended by 1 bit. And arithmetic shift left is same as logical shift left, no difference. And, arithmetic shift right means if it is a negative

number when you shift right, 1 will be added to the most significant bit side if it is positive number 0 will be added, these are the various options.

(Refer Slide Time: 26:17)



In this diagram some of these are shown. You see here if you say logical shift left 5, the original register will be shifted left by 5 positions and 0's will be added on the right. If you say logical shift right by 5 positions similarly you shift right and 0's get added.

If I say arithmetic shift right, but the number is positive which means the MSB was 0 then 0's will be added, but if the number was negative which means most significant bit was 1 then 1's will be added on the left side. And, rotate as I said whatever goes out will be getting inside here. These are the various options.

(Refer Slide Time: 27:23)

The slide is titled 'Multiplication instruction' and 'Multiply-accumulate instruction'. It contains the following text and code snippets:

- Multiplication instruction**
`MUL r1, r2, r3 ; r1 = (r2 x r3) [31:0]`
 - Only the least significant 32-bits are returned.
 - Immediate operands are not supported.
- Multiply-accumulate instruction:**
`MLA r1, r2, r3, r4 ; r1 = (r2 x r3 + r4) [31:0]`
 - Required in digital signal processing (DSP) applications.
 - Multiplication with 64-bit results is also supported.

The slide is part of a presentation on the Swayam platform, featuring a logo for 'FREE ONLINE EDUCATION swayam' and a photo of a speaker in the bottom right corner.

There are some multiplication instructions also. `MUL r1, r2, r3`. You multiply the two numbers `r2, r3` and you take only 32-bits of the result because multiplication result can be 64-bit, but you ignore the high order bits you assume that the result will fit within 32-bits and you take only the last 32-bits. And in multiplication immediate operations cannot be used.

There is another instruction that is very much useful for digital signal processing applications, this is called multiply and accumulate. There you need to continuously multiply a number with some other number and add to another value continuously.

With this we come to the end of this lecture where we have talked about some of the arithmetic and logical instructions that are available and supported by the ARM instruction set architecture. In the next lecture, we shall be talking about the data transfer instructions; what are the various kinds of data transfer instructions that can be used to transfer data between registers and memory.

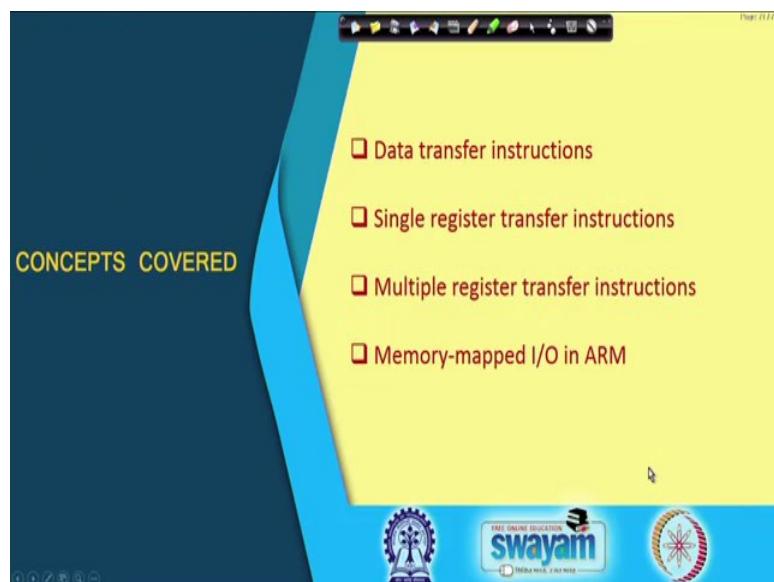
Thank you.

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

Lecture – 8
ARM Instruction Set (Part II)

In the last lecture we were discussing about the ARM instructions with which you can do some arithmetic and logic operations. Now in this lecture we shall be further looking into instructions that allow you to transfer data between the registers and memory.

(Refer Slide Time: 00:44)



In this lecture we shall be primarily talking about data transfer instructions, mainly between registers and memory, and you will see that there are instructions which can transfer a single register value and also multiple register values. And we shall at the end talk about input/output operations in ARM that supports something called memory mapped IO.

(Refer Slide Time: 01:18)

(b) Data Transfer Instructions

- ARM instruction set supports three types of data transfers:
 - Single register loads and stores
 - Flexible, supports byte, half-word and word transfers
 - Multiple register loads and stores
 - Less flexible, multiple words, higher transfer rate
 - Single register-memory swap
 - Mainly for system use (for implementing locks)

semaphore

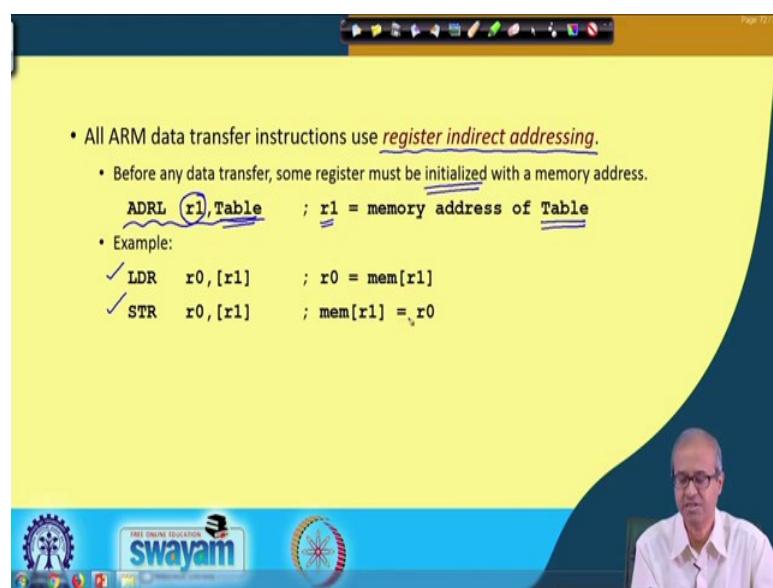
Let us look at the data transfer instructions that are supported by the ARM architecture. Broadly there are three types of data transfer that are allowed. Here we are mainly talking about registers and memory. You can have single register load and store it from memory, you can load the value into a single register, there is a load instruction; store means from a register you write that result into some memory location.

There is a single register variant where one value is loaded or stored; or you can have an option for multiple registers load and store. Like you can say that not one, but four registers will be loaded, or let us say four registers will be written into memory one after the other. You can specify that in the same instruction. Also there is a very special kind of instruction where a register-memory swap operation is allowed; swap means exchanging the values of a register and a memory location in a single instruction.

This kind of instruction is very useful not in the normal scenario primarily by the operating system, it is used by the operating system for implementing several different kind of locks. There is a very commonly used lock which is called semaphores and these kind of register-memory swap instructions are very much useful to implement these semaphores.

But for normal application development, we do not require to use this kind of instruction.

(Refer Slide Time: 03:32)



The first thing to note is that in ARM the data transfer instructions use register indirect addressing. That is, you must use some register to point to memory location and via that register you are doing load and store. Before you can access memory using load and store, you will have to initialize some register with the address of a memory location. There is an instruction ADRL. Here you specify some register, say r1, and you specify some memory location, Table. In assembly language, you do not specify the absolute address; you specify some label with respect to the data.

So, what will happen? The memory address corresponding to this data item Table will be loaded in r1. Once you have done this, you can use load register and store register instructions in LDR, STR. LDR r0,[r1] means memory location whose addresses in r1 will be transferred to r0. In STR, just the reverse; memory location whose address is in r1 will get the value from r0. These kinds of load and store operations are supported.

(Refer Slide Time: 05:24)

The slide is titled 'Single register loads and stores' in red. It contains the following text and code:

- The simplest form uses register indirect without any offset:
`LDR r0,[r1] ; r0 = mem[r1]`
`STR r0,[r1] ; mem[r1] = r0`
- An alternate form uses register indirect with offset (limited to 4 Kbytes):
`LDR r0,[r1,#4] ; r0 = mem[r1+4]`
`STR r0,[r1,#12] ; mem[r1+12] = r0`
- We can also use auto-indexing in addition:
`LDR r0,[r1,#4]! ; r0 = mem[r1+4], r1 = r1 + 4`
`STR r0,[r1,#12]! ; mem[r1+12] = r0, r1 = r1 + 4`

The slide is part of a presentation on the Swayam platform, as indicated by the logo at the bottom.

This is what we call single register load and store, where we are loading and storing one register at a time. Now as I said ARM supports various other options using multiple registers, which we shall discuss a little later. But before that even in single register load and store, there are various options we can have.

First option is simple register indirect. We can have register indirect with offset, like we can specify an offset value with respect to the current instruction. This offset is limited to 4 kilobytes, because in the instruction the space that is provided is 12 bits, and in 12 bits we can specify $2^{14} = 4$ kilobytes. Earlier within the square bracket you are specifying only the register, here you are specifying also some immediate data like this; `[r1,#4]`. What does this mean? The value of this register is added to this offset and then memory location corresponding to that address is loaded into the register. Similarly for store.

So, not only register indirect, I can also specify an offset that will be added to the register to generate the final memory address from we are loading and storing. There is a third optional also available. You can have auto indexing in addition to using register indirect with offset. Auto indexing means, there are many application where you want to load or store data one by one consecutively from a block of memory locations. The register which was pointing to some memory location automatically gets incremented after one transfer is done. And how do I specify this? I specify this by this ! symbol.

(Refer Slide Time: 09:23)

Page 76/77

- We can use post indexing:
`LDR r0,[r1],#4 ; r0 = mem[r1], r1 = r1 + 4
STR r0,[r1],#12 ; mem[r1] = r0, r1 = r1 + 12`
- We can specify a byte or half-word to be transferred:
`LDRB r0,[r1] ; r0 = mem8[r1]
STRB r0,[r1] ; mem8[r1] = r0
LDRSH r0,[r1] ; r0 = mem16[r1]
STRSH r0,[r1] ; mem16[r1] = r0`

B
SH

FREE ONLINE EDUCATION
swayam

A man in a white shirt is visible in the bottom right corner of the slide.

There is another variation called post indexing. Here meaning is slightly different, this means you first load the value as usual and then increase r1 not by 4 always, but whatever offset you have given. If you give #4, it will be incremented by 4. If you give #12, it will be incremented by 12.

Sometimes you may need to increase the register by some other value than 12, you can use this variation of the instruction in those cases. And the other thing is that whatever we have discussed so far we are assuming that the entire 32-bit word is transferred. But there are many applications where you do not need entire 32-bit of data; 8 bits or 16 bits of data may be sufficient. These are called byte or half word. There are some categories of load and store that work with byte and half word. Byte is indicated by B and half word is indicated by SH.

(Refer Slide Time: 11:45)

• Multiple register loads and stores

- ARM supports instructions that transfer between several registers and memory.
- Example: $r2-r5$

LDMIA $r1, \{r3, r5, r6\}$; $r3 = \text{mem}[r1]$ A: After
; $r5 = \text{mem}[r1+4]$ B: Before
; $r6 = \text{mem}[r1+8]$

- For LDMIA, the addresses will be $r1+4$, $r1+8$, and $r1+12$.
- The list of destination registers may contain any or all of $r0$ to $r15$.

- Block copy addressing

- Supported with addresses that can increment (I) or decrement (D), before (B) or after (A) each transfer.

Now, let us come to multiple register load store variety. Here we are saying that in ARM instruction set, there are facilities to transfer data between several registers and memory. You recall we mentioned earlier that there are some facilities where ARM is deviating from RISC architecture.

One philosophy of RISC architecture is that all instructions should be of equal size. They should be very simple, they should be able to execute in a single cycle, but when you are transferring multiple words naturally you cannot finish execution in one cycle; multiple cycles will be required because you are loading or storing multiple memory words. So, this is where we are slightly deviating from the pure RISC concept, but this is a very powerful instruction. In many applications you may want to use it. You see one variation of this instruction is LDMIA, there is another version LDMIB. The letters A and B actually stands for After and Before respectively.

Let us see how these instructions are specified. Normally in load register, there are two operands. But here the first one is r1. It is assumed that the first operand is containing the memory address. Here we are not giving that square bracket that is implied, and in the second operand within curly bracket we are specifying a set of registers. Now there are many ways you can specify. If the registers are consecutive in number, we can write like this r2-r5; this means r2, r3, r4, r5. Otherwise you can separate by commas as r3,r5 r6. This A means you first load and then after that increase the address.

There is another variation LDMIB. B or before means you first increase it by 4 and then load. In the list of destination registers, we can use any of the 16 registers. In addition to this B and A, we can also use I and D options. I stands for increment, and D stands for decrement. All these options are available.

(Refer Slide Time: 16:02)

Page 78/79

- Examples of addressing modes in multiple-register transfer:

The diagram illustrates four memory states corresponding to different ARM instructions:

- STMIA r9!, {r0,r1,r5}**: Shows a stack-like structure with r9 at the top. The stack grows downwards. The memory state is: 1018_{16} (top), $r5$, $r1$, $r0$, 1000_{16} , $100C_{16}$, 1000_{16} , 1018_{16} (bottom).
- STMIB r9!, {r0,r1,r5}**: Shows a stack-like structure with r9 at the bottom. The stack grows upwards. The memory state is: 1018_{16} (top), $r5$, $r1$, $r0$, 1000_{16} , $100C_{16}$, 1000_{16} , 1018_{16} (bottom).
- STMDA r9!, {r0,r1,r5}**: Shows a stack-like structure with r9 at the top. The stack grows downwards. The memory state is: 1018_{16} (top), $r5$, $r1$, $r0$, 1000_{16} , $100C_{16}$, 1000_{16} , 1018_{16} (bottom).
- STMDB r9!, {r0,r1,r5}**: Shows a stack-like structure with r9 at the bottom. The stack grows upwards. The memory state is: 1018_{16} (top), $r5$, $r1$, $r0$, 1000_{16} , $100C_{16}$, 1000_{16} , 1018_{16} (bottom).

Legend:

- LDMIA, STMIA
 - Increment after ✓
- LDMIB and STMIB
 - Increment before ✓
- LDMDA, STMDA
 - Decrement after ✓
- LDMDB, STMDB
 - Decrement before ✓

FREE ONLINE EDUCATION **swayam** INDIA WIDE, 24x7x365

Let us see some examples here. The first example I am showing here is store memory location increment after. So, here this r9 here I am giving this ! sign means this r9 will get incremented; auto indexing auto increment. Similarly, the other examples.

(Refer Slide Time: 18:58)

Page 79/79

- Point to note:

- ARM does not support any hardware stack.
- Software stack can be implemented using the **LDM** and **STM** family of instructions.

FREE ONLINE EDUCATION **swayam** INDIA WIDE, 24x7x365

One interesting thing is that there is no stack in ARM architecture, which is so much popular in other contemporary architectures. Of course, a programmer can maintain a software stack, but there are no explicit push or pop instructions that are normally a part of the instruction set of conventional computer architectures. LDM and STM instructions can be used to implement stack; we shall see some examples later.

(Refer Slide Time: 19:38)

An Example

- Copy a block of memory (128 bytes aligned).
- r9: address of the source
- r10: address of the destination
- r11: end address of the source

16 Loop: LDMIA r9!, {r0-r7} 64 instr.
 STMIA r10!, {r0-r7}
 CMP r9, r11
 BNE Loop

Page 79/100

Let us take one example of copying a block of 128 memory locations. Suppose I have an application where in memory I have a block of consecutive 128 bytes.

I want to transfer this entire block into some other memory location. Let us say I want to transfer it here. Now using single register moves, what I can do? I can write a loop. I initialize some register with the starting address, and I do it 128 times -- load one number from the initial block, store it here; load the next number, store it in the next location; load the third number, store it in the third location, and I check whether all 128 number of data have been transferred.

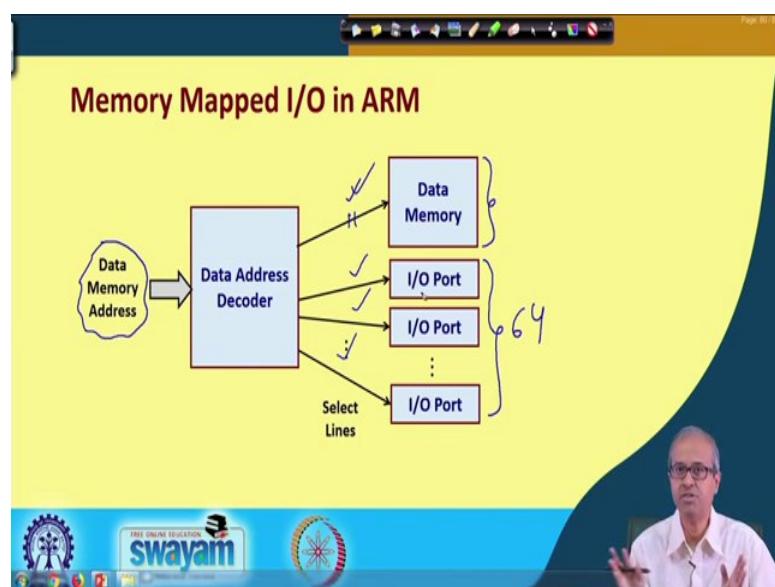
Now here I will have to loop for 128 times. But when I use this kind of multiple register transfer instruction, say, I have 8 registers at my disposal. If I load 8 registers together and store 8 registers at one time; that means I am transferring 8 bytes in one go. In terms of the loop how many times I have to loop in that case? Only 16 times, as $16 \times 8 = 128$. So, the number of instructions being executed will be drastically less. This will also

reduce the overall execution time because every instruction execution has its own overhead.

So, if you transfer multiple memory location as part of one single instruction, the total number of clock cycles will reduce. The program is shown.

This will be much faster because we are looping 16 times, with 4 instructions per loop. In the earlier case you were requiring $128 \times 4 = 512$ instructions, but in comparison you are here only requiring $16 \times 4 = 64$ instructions to be executed.

(Refer Slide Time: 24:25)



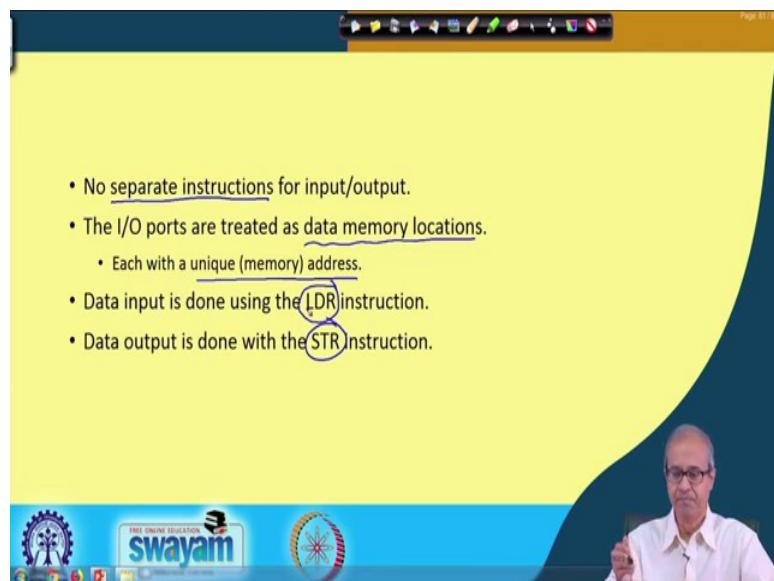
Lastly one thing we want to discuss, there is a concept called memory mapped IO that is prevalent in many computer systems.

The basic idea is like this. Whenever some data are accessed, data memory address is generated by the processor. There is a decoder which decodes this address and it selects the corresponding memory location. Normally the data memory is accessed, but in memory mapped IO what we are saying in addition is that the IO ports are also regarded as memory locations.

Let us take an example, suppose there are 6 IO ports. We are assuming that there are 64 memory locations also. The same decoder depending on the address will be selecting either one of the data memory addresses or one of these ports. So, the IO ports or IO

devices are treated as memory locations. This is the basic concept behind memory mapped IO. There is no separate instruction for doing input and output.

(Refer Slide Time: 26:12)



There are no separate instructions for input and output, which is there for conventional processors like 8051 microcontroller or the 8085 microprocessor where you will find IN and OUT instructions that are meant primarily for input and output.

The IO ports are treated as data memory location, where each of them will be having a unique address. When you want to read some data from an input device, you will be using load instruction; when you want to write some data into an output device you will be using store instruction. That is how ARM handles it. This provides some added flexibility, because there are so many flexibilities the instruction set offers when you are accessing memory locations. Now if you treat your I O devices also as memory locations then even when accessing the IO devices the same kind of flexibility you can utilize.

With this we come to the end of this lecture. In the next lecture we shall be talking about the control flow instructions that are available in the ARM instruction set.

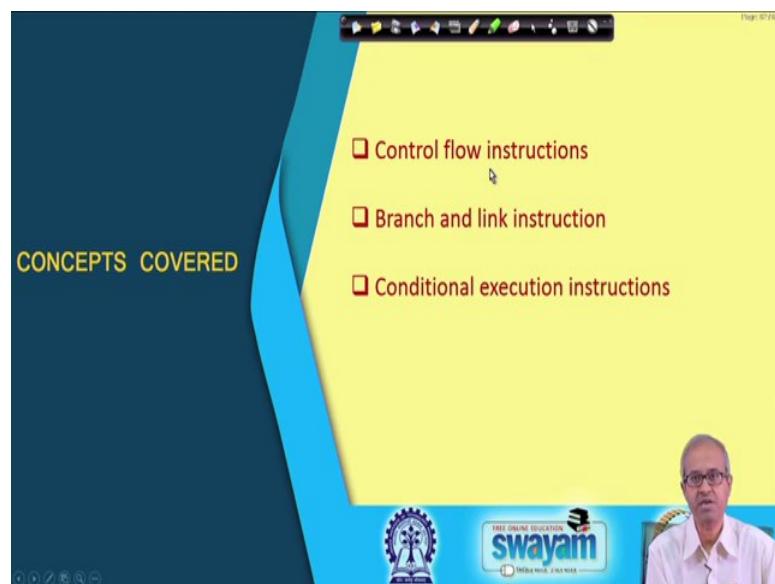
Thank you.

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

Lecture – 09
ARM Instruction Set (Part III)

We now discuss the control flow instructions that are available in ARM.

(Refer Slide Time: 00:31)



This is the third part of our lecture series on ARM instruction set. Here we shall be talking about the control flow instructions. In particular we shall be looking at the branch and link instruction that are primarily used for handling subroutine calls, and we shall talk about the conditional execution instruction that you can also regard to be belonging to this category control flow.

(Refer Slide Time: 00:58)

(c) Control Flow Instructions

- These instructions change the order of instruction execution.
 - Normal flow is sequential execution, where PC is incremented by 4 after executing every instruction.
- Types of conditional flow instructions:
 - ✓ Unconditional branch
 - ✓ Conditional branch
 - ✓ Branch and Link
 - ✓ Conditional execution

$PC = PC + 4$

Let us look at the control flow instructions. As I had said earlier, control flow instructions are those that change the sequence of instruction execution. In the normal flow of execution; the instructions are normally stored one after the other in memory, PC points to the next instruction to be executed. Whenever one instruction finishes execution, you increment PC by 4, so that PC will point to the next instruction. Each instruction is of size 32 bits, so you will have to increase it by 4 bytes.

Now in control flow, this PC is not being incremented by 4, but rather you are updating PC with some other value. So, the next instruction will be from somewhere else. In terms of the types of control flow instructions, there can be unconditional branch, conditional branch, branch and link, and conditional execution instructions. There can be broadly these 4 categories of instructions.

(Refer Slide Time: 02:35)

- Unconditional branch instruction:

Diagram: B (Target) ... Target

Handwritten notes: $PC = PC + 4$
 $PC = Target$

- Conditional branch instruction:

```
LOOP    MOV    r2, #0
        ...
        ADD    r2, r2, #1
        CMP    r0, #20
        BNE    LOOP
        ...
```

Let us look at these instructions one by one. The first one is the unconditional branch instruction. In unconditional branch, we have an instruction called branch whose mnemonic is just B. In assembly language we just say B Target.

So it means after this instruction we shall not go to the next instruction, but rather next instruction will be here at Target. The way PC will be incremented is that instead of doing $PC = PC + 4$, what we are saying is that we will be doing $PC = Target$, so that the next instruction that will be fetched will be this instruction which is stored here. This is called unconditional branch instruction meaning that whenever you have this B instruction, there will always be a control transfer to this new address.

Now you can also have this transfer of control depending on some condition. These are called conditional branch instruction. Here I have shown one conditional branch instruction called Branch if Not Equal (BNE). Let us look at this simple code segment; you will understand what this is doing. Here we are initializing some register r2 to 0, then some instructions here some calculations are going on. Then we are adding r2 to 1 and the result is stored back into r2.

Then we are comparing whether r2 is becoming 20 or not, if it is not 20 then we go back to loop. We repeat this and once it becomes 20, it comes out and continues with the next instruction. This means this is a simple loop we have implemented that will be repeating

a set of instructions 20 times. You can use this conditional branch instruction to implement this kind of conditional loop.

(Refer Slide Time: 05:42)

The slide contains a list of branch conditions supported by the processor:

- Branch conditions that are supported:
 - B, BAL Unconditional branch
 - BEQ, BNE Equal or not equal to zero
 - BPL, PMI Result positive or negative
 - BCC, BCS Carry set or clear
 - BVC, BVS Overflow set or clear
 - BGT, BGE Greater than, greater or equal
 - BLT, BLE Less than, less or equal

Below the list is a video player showing a man in a white shirt speaking. The video player has a yellow overlay on the left side. The bottom of the slide features the Swayam logo and other navigation icons.

Not only BNE, a variety of other conditions are possible as this list shows.

You can make some calculations or comparisons, and based on that you can take your decision whether to jump somewhere else or not.

(Refer Slide Time: 06:54)

The slide is titled "Branch and link instruction" and contains the following points:

- Used for calling subroutines in ARM.
- The return address is saved in register r14 (called link register). *Current value of PC*
- To return from the subroutine, we have to jump back to the address stored in r14.

Below the text is a code example:

```
BL MYSUB ; Branch to subroutine
            ; Return here
r14 = X
PC = X
...
MYSUB    ... ; Subroutine starts here
...
MOV pc,r14 ; Return
```

A handwritten note "Nested subroutine calls cannot be used in this way." is written below the code. The video player shows a man in a white shirt speaking. The bottom of the slide features the Swayam logo and other navigation icons.

Talking about subroutine calls, in many instruction set architectures, there is some instruction like CALL. There is a CALL instruction that you use to call a subroutine. When you call a subroutine normally what happens? The PC is pushed in the stack, you jump to the subroutine, subroutine gets executed, the last instruction of the subroutine will be a return instruction. What the return will do? It will pop the last element from the stack, it will load it into PC, which means you return back to the program from where the call instruction was executed. This means the next instruction, because PC always points to the next instruction.

That is how the things are handled in a conventional processor, but in ARM there is no CALL instruction rather there is an instruction called branch and link, BL in short. Here there is no stack as I told you. There is a special register r14 which is called the link register. Return address means the current value of PC. Whenever an instruction is executed PC is immediately incremented by 4. So, it is pointing to the next instruction, that is the return address.

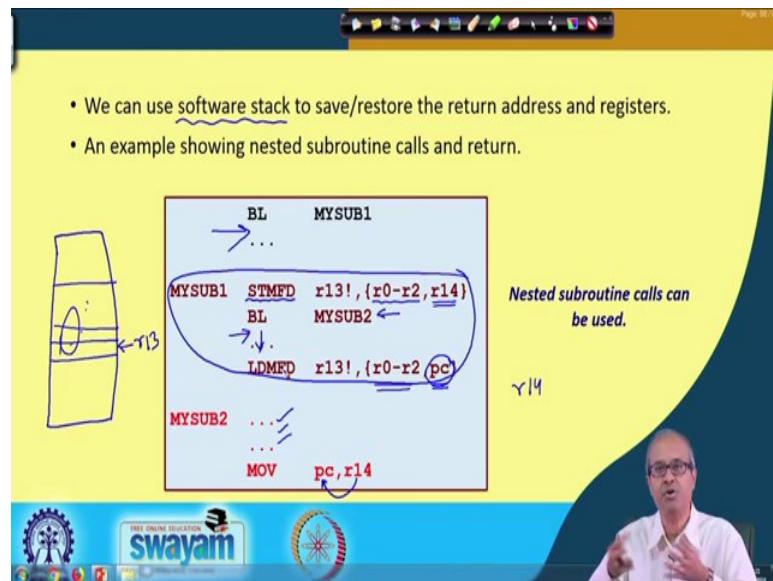
The return address will be saved into the link register, and then you jump to the subroutine. And when you are trying to return back, you will have to jump back to the address that is stored in r14. This is how in ARM subroutine call/return can be handled. Let us take a simple example here; a very small code segment is shown. Suppose in a program somewhere you have given a subroutine call instruction BL and this is the subroutine you are calling.

Let us say MYSUB is the label of the subroutine you are calling and this red box indicates the body of the subroutine. Inside this subroutine there will be some instructions, but before jump into the subroutine what will happen? In the link register r14, this next instruction address let us say this is X, this X will get stored. The current PC if it is X, X will get stored in r14 and then it will be branching to MYSUB. In MYSUB, there will be several instructions, it will execute. At the end when it wants to return back, it executes some MOV instruction. What kind of MOV? MOV r14 to PC.

Essentially you are branching back or jumping back to this location and you are resuming execution from here ok. This is how you are implementing subroutine call and return in ARM. But it is clearly mentioned here that in this way you cannot implement

nested subroutine call or recursion, which is so common; you see here you have only one register to store the return address r14.

(Refer Slide Time: 12:05)



Here I am showing a simple example again where we are illustrating the use of a software stack. I said ARM does not support a stack, but we can implement a stack in software using available instructions. Let us assume that we have a memory location we want to implement or stack here. I said r13 is a register that you typically use as the stack pointer. Let us say we are using r13, these are some memory locations and r13 is pointing here. In this program what I am illustrating from my main program, I am making a call BL MYSUB1.

This is the body of MYSUB1 and from MYSUB1, I am again calling another subroutine MYSUB2. This is a nested call, two level nesting I am demonstrating. In the first level of nesting, just to ensure that my r14 value does not get lost, we are using a multiple register store instruction STMFD. So, what it does? It essentially saves r14 and some other registers which I may be needing. These three and also this r14 link register are stored in r13 stack here. Then you branch to MYSUB2 branch and link.

Now your r14 gets overwritten, it will contain the return address of this instruction. MYSUB2 gets executed and once it is done, you return back to the address stored in r14. So, you return back here and resume execution from here. Now at the end you have a matching load multiple register load instruction, what you do? You load again from r13;

this r13 is pointing there. Here you are loading r0 to r2, and the last one will be loaded to PC.

So, whatever this r14 value was saved in the stack that will now get loaded in PC, which means it will automatically return back here. This kind of a multiple register load and store instruction you can use to implement nested subroutine call or even recursion.

(Refer Slide Time: 15:41)

- Conditional execution
 - A unique feature of the ARM instruction set.
 - All instructions can be made conditional, i.e. will get executed only when a specified condition is true.
 - Helps in removing many short branch instructions (improves performance and code density)
 - An example: `if (r2 != 10) r5 = r5 +10 - r3`

`CMP r2,#10
BEQ SKIP
ADD r5,r5,r2
SUB r5,r5,r3`

`CMP r2,#10
ADDNE r5,r5,r2
SUBNE r5,r5,r3`

Z = 1

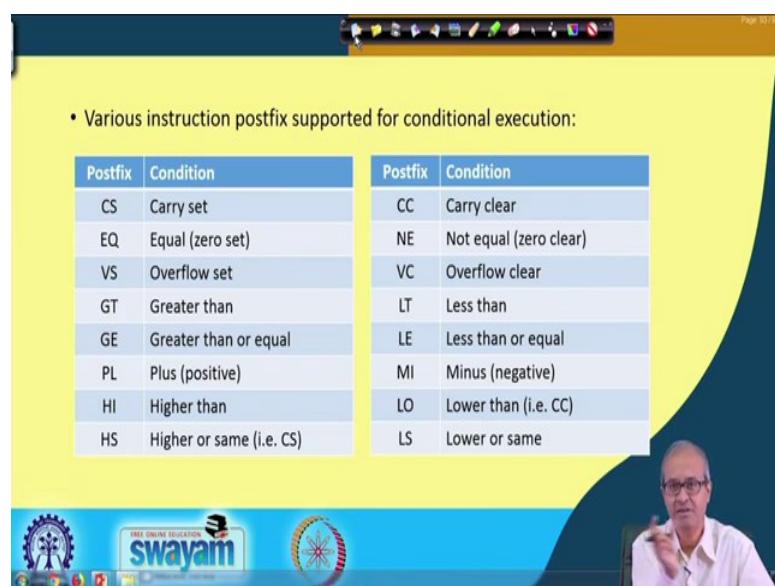
A very important feature of ARM instruction set is something called conditional execution. This is a unique feature that we normally do not see in other instruction sets. The designers for ARM have very carefully thought out these set of instructions. This says that the instructions can be made conditional; conditional means they may either execute or they may not execute. I can specify some condition along with every instruction, there will be a field where some condition is specified; if this condition holds, then it is executed; otherwise it is not executed.

Let us take an example. Suppose in high level language, I want to implement something like this. Here I have a solution. Suppose I implement like this I have to check r2 with 10, whether it is equal to 0 or not equal to 0. My requirement says if it is not equal to 0, then do this calculation. So, what I am doing? I am comparing r2 with 10. If it is equal then I am not supposed to do this; if it is equal I am branching to this SKIP. I am skipping these two instructions, but if it is not equal then I am going through this ADD and SUB.

The only thing that I want to say is that, here I am requiring one branch instruction, but if I use the conditional variety of implementation like this, you see what we have done here? We are again comparing r2 with 10. In the next ADD and SUB instructions we have added the conditional postfix not equal to; that means, if not equal to then you do these.

So, when I say ADDNE, the computer is actually testing the 0 flag; if it is 0 then execute this; similarly SUBNE; if the 0 flag is 0, then execute this. This conditional variety of instructions helps in avoiding one branch instruction. This helps in removing many short branches. In a general code you will find many such instances. These conditional instructions can help in eliminating many such branch instructions make your code shorter and more efficient. This is the main advantage.

(Refer Slide Time: 20:18)



Postfix	Condition
CS	Carry set
EQ	Equal (zero set)
VS	Overflow set
GT	Greater than
GE	Greater than or equal
PL	Plus (positive)
HI	Higher than
HS	Higher or same (i.e. CS)
Postfix	Condition
CC	Carry clear
NE	Not equal (zero clear)
VC	Overflow clear
LT	Less than
LE	Less than or equal
MI	Minus (negative)
LO	Lower than (i.e. CC)
LS	Lower or same

Now the various instruction postfix that are available are shown here.

As part of an instruction there are 4 bits reserved that will be specifying the condition.

(Refer Slide Time: 21:13)

Another example:

```
if ((r1 == r3) && (r5 == r6)) r7 = r7 + 10
```

CMP r1,r3	BNE SKIP
CMP r5,r6	BNE SKIP
ADD r7,r7,#10	

→ SKIP ...

→

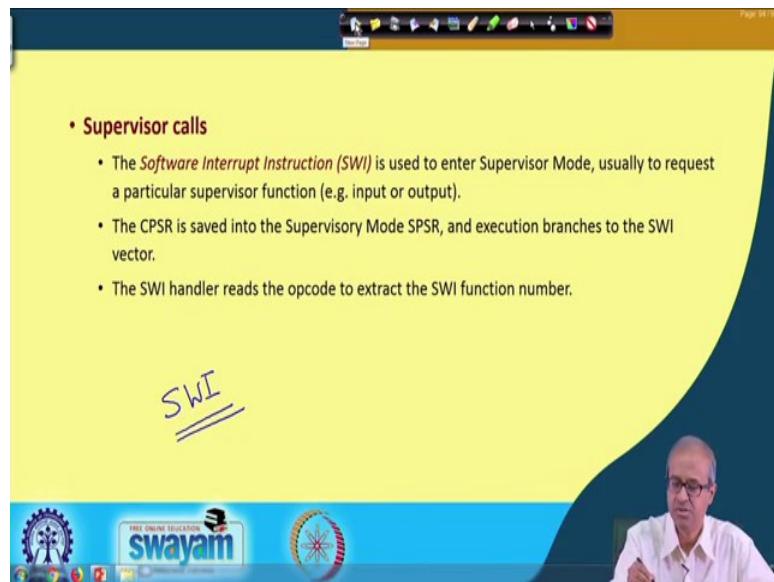
CMP r1,r3	CMPEQ r5,r6
	ADDEQ r7,r7,#10

z

Let us take another example. Here we are checking for two conditions together. This is the conventional approach, you first compare r1 and r3; if they are not equal you can straight away skip; that means, you have you do not have to do it. If it is not equal to straight away come to SKIP, then you check the next condition r5 and r6. If they are not equal then you go to SKIP otherwise you add.

But in the conditional variety, the second compare instruction also you can make conditional. So, you see here you are eliminating two branch instructions. Your code is becoming so much more efficient. The addition of conditional execution instructions, this has added a new dimension to the way people can write codes. The code can be very compact and efficient, and in terms of execution time will also take less time because there are fewer number of instructions.

(Refer Slide Time: 23:14)



- **Supervisor calls**

- The *Software Interrupt Instruction (SWI)* is used to enter Supervisor Mode, usually to request a particular supervisor function (e.g. input or output).
- The CPSR is saved into the Supervisory Mode SPSR, and execution branches to the SWI vector.
- The SWI handler reads the opcode to extract the SWI function number.

SWI

There is another kind of an instruction that also changes the value of PC. Of course, it is not so much important from the point of view of embedded system design. These are called supervisory calls. There is an instruction called software interrupt or SWI, this is like a subroutine call. This will also jump to a subroutine by saving the return address in some register r14, but the main difference is that while doing this jump it will also change the processor mode to supervisor mode. These instructions are typically used in environments where there is an operating system and you are trying to request some service from the operating system. You give SWI call, it jumps to the OS and also the mode changes to supervisory mode. Normally when the OS executes, the processor mode is supervisory, that automatically happens. We are not going into too much detail of this.

We have seen in a nutshell, the various kinds of instructions that are there in ARM and the unique features. In particular I have talked about the conditional execution in this lecture. Earlier you have seen in terms of the arithmetic and logic instructions when you are executing, the second operand can be specified in so many flexible ways. There are many ways in which you can make your code shorter by taking advantage of these flexible methods, then the multiple register transfer instructions, and so on.

As I said that in this course we shall be looking at a lot of experimental demonstrations based on some microcontroller boards. In the next lecture we shall be introducing you to

one of the boards based on which many of the experiments shall be demonstrated. We shall be discussing the basic features of the boards, the different kind of connectors and how to use them, and subsequently we shall be seeing actually how they are put to use.

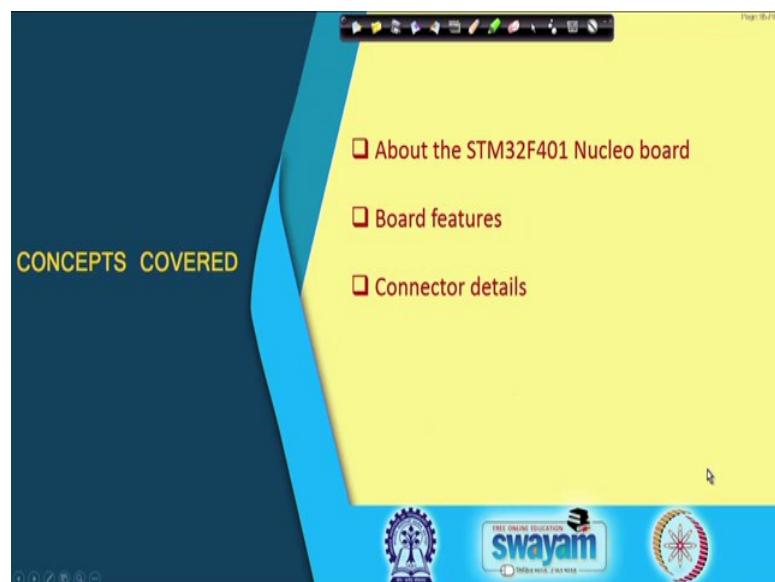
Thank you.

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

Lecture – 10
About the STM32F401 Nucleo Board

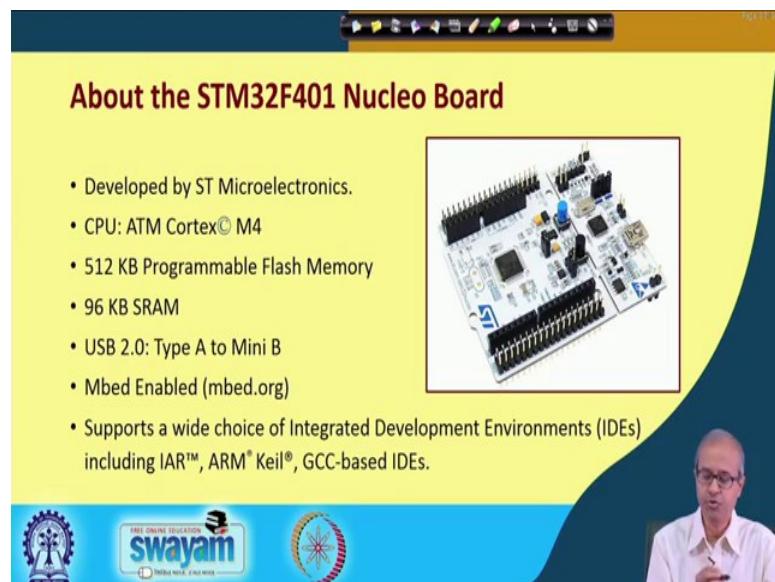
In this lecture we shall be introducing you to one very popular microcontroller development board that is used in embedded system design. The name of the board is STM32F401.

(Refer Slide Time: 00:42)



In this lecture we shall be telling you some notable features about the board.

(Refer Slide Time: 01:10)



About the STM32F401 Nucleo Board

- Developed by ST Microelectronics.
- CPU: ATM Cortex® M4
- 512 KB Programmable Flash Memory
- 96 KB SRAM
- USB 2.0: Type A to Mini B
- Mbed Enabled (mbed.org)
- Supports a wide choice of Integrated Development Environments (IDEs) including IAR™, ARM® Keil®, GCC-based IDEs.

So, this STM32F401 is a board you can see the picture here. This is manufactured by a company called ST microelectronics. This board contains a lot of small components. Here you can see a larger IC chip here, this is the actual microcontroller sitting inside and there are many other accessories.

You see the main features that are mentioned here: this board is manufactured by a company ST microelectronics. They do not manufacture the chip, they manufacture the board, and the CPU that is use this middle chip that I am showing the central one, this is ARM Cortex M4. This is the ARM processor that is used and the name of the processor is ARM Cortex M4.

The feature of this ARM Cortex M4 board is, inside there is 512 kilobytes of program memory. They have used flash memory as the program memory, they have not used a ROM because programming a ROM is quite troublesome. Flash memory can be programmed on the fly, you can download it, you can store in flash. And it is a non-volatile memory, even you switch off the power the program will not disappear. And there is also some RAM – random access memory, which you can use to store data. There is 96 kilobytes of that.

There is also an USB interface out here through which you can connect this board to other devices, like your PC or desktop or laptop, and this board is mbed enabled. Well,

mbed is like a platform; if a board is mbed enabled then all the tools that are available under mbed can be used along with this board.

Well, mbed is a set of tools that are primarily meant for the developers of embedded system boards; they have developed a lot of utilities that make the process of development very easy. And, there are some features that supports a wide choice of integrated development environments.

There are several software based interfaces through which you can access these boards, you can program these boards, and so on. This we shall see later because in our experiments we shall be using tools available on mbed.org. There we shall show how to interface, how to use, and how to program this board.

(Refer Slide Time: 04:42)

Some Specific Details

- Contains STM32F401RET6 microcontroller with ARM Cortex-M4 CPU with FPU at 84 MHz clock, 512 KB of flash and 96 KB of SRAM with a variety of peripherals.
- Two types of extension connectors are available:
 - Arduino Uno Revision 3 connectivity.
 - STM Morpho extension pins for full access to all STM32F401RET6 I/Os.
- On-board ST-LINK/V2.1 interfacing with the PC, which provides:
 - Debug and programming port, to use the board with standard programmers.
 - Virtual Com port to send back traces to the PC.
 - Mass storage (USB Disk Drive) for drag-n-drop programming.

Now, some specific details are mentioned here. In this board as I mentioned we have STM32F401 microcontroller, there is a version RET6. This contains ARM Cortex-M4 CPU with lot of other things. There is a FPU which is running at 84 MHz clock, functional processing unit, 512 kilobytes of flash, 96 kilobytes of SRAM, and a lot of IO ports. And other peripherals are provided that makes embedded development very easy.

And, there are two types of extension connectors that are available as part of the board. One is Arduino UNO Revision 3 compatible. You see Arduino is a very popular platform for developing small embedded system design today. Arduino is computationally very

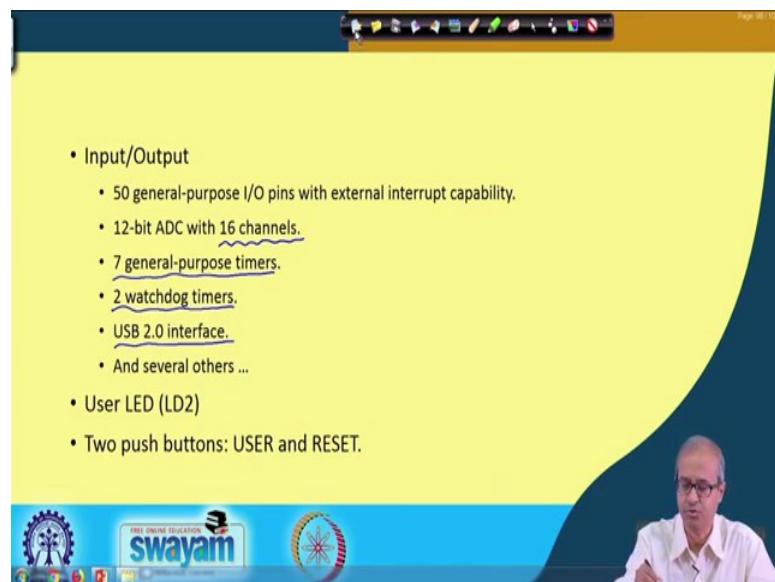
simple, not as powerful as ARM. But, because Arduino is so popular when you are developing something around an ARM processor, it may so happen there may be some other subsystems with which you have to make connections that are based on Arduino. In order to enable that this board provides an Arduino kind of connector.

In addition to that, this STM Morpho extension pins provide you full access to all the input output pins of the STM microcontroller that is sitting inside the board. There are some inbuilt interfaces provided that allows you to interface with several standard integrated development environments using which you can run some debuggers from outside, if your board is not working properly you can test or diagnose the board using those debuggers.

You can use a virtual serial port or virtual com port through which some of the data you can send back to your PC and display on your PC screen. And, just like whenever you mount something on a PC or laptop you see that device mounted as a drive, this nucleo board also when you connect it to a PC through USB connector which also gets mounted as a USB disk drive. When you program this board you simply drag and drop that program into the disk drive icon, that program will automatically get downloaded.

This driver is automatically installed as part of the environment. So, it becomes very easy for the developer. All these things we shall see later when we show you the demonstration.

(Refer Slide Time: 08:25)



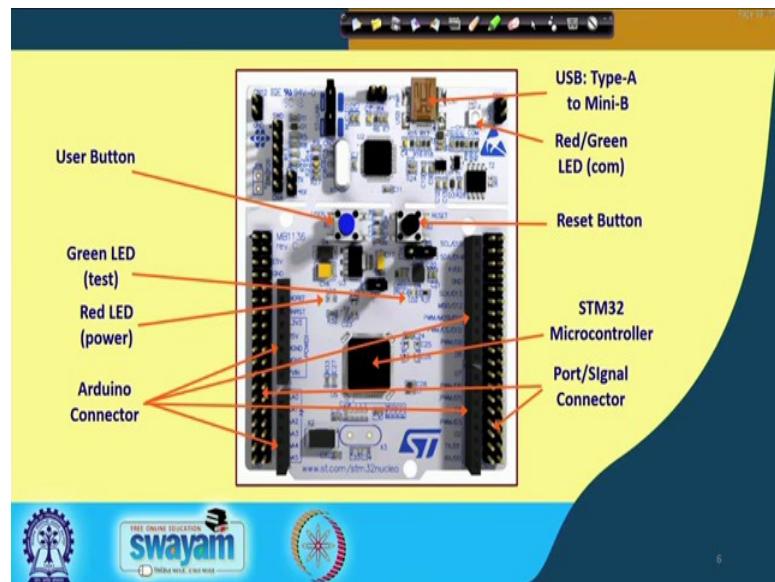
Regarding the external interfaces a very quick summary is shown here. There are fairly large number of IO ports that are provided, 50 general-purpose IO pins are provided. Now, one good thing is that all these IO pins can also be used to connect with external interrupt sources. There are some constraints we shall be discussing later, but the thing is that if we want to implement an application where some external devices are sending interrupt, you can connect it to any of these 50 pins.

This is unlike some older processors; let us say 8085 which many of you may be knowing, which had some dedicated interrupt pins. There were some pins called TRAP, RST 7.5, 6.5, 5.5 and INTR; there are five interrupt lines which are dedicated. When you wanted to connect an interrupting device you had to connect to one of these five lines. But, here you can connect anywhere, some pin you can use as a general purpose IO you can also use as interrupt.

And, another thing is that there is a built in analog to digital converter. A/D converters are very important for embedded system design because many of the external parameters that you want to sense are analog in nature. They are continuously varying parameters that can be translated to voltages. In order to interface them if we have an A/D converter on board it becomes very convenient.

Inside the board there is a 12-bit A/D converter and you can connect up to 16 input devices to it. There are several timers provided on board. There are 7 of them, two of them are watchdog timers, and I mentioned there is a USB 2.0 interface. This is primarily used to connect with a host system like a PC or a laptop, there are some LEDs, some push button switches.

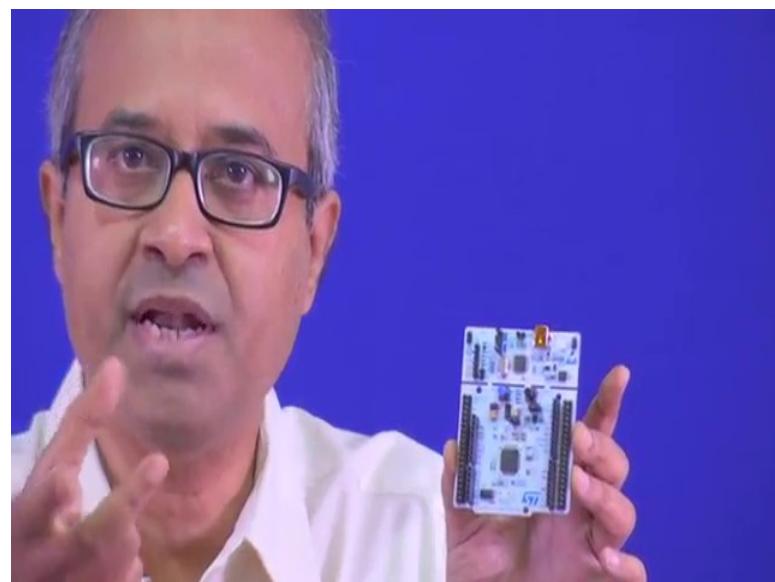
(Refer Slide Time: 11:13)



Here you have a picture of the board. Now, let me show you this board.

(Refer Time: 11:25).

(Refer Slide Time: 11:55)



This is the Nucleo board which we are talking about. If you can see the two sides of it, there are so many connectors you can see. There are some black smaller sized connectors and at the internal part you can see, these are the Arduino compatible connectors. If you look at an Arduino board they will have a connector with an exact shape like this, you can connect a cable from there directly to this. But, if you see the other pins available on

the two sides, the pins provide you with the signals that the internal microcontroller on board is generating. All these signals can be accessed either from these signal pins, or a small subset of that is available over these Arduino interface pins, you can also avail it from there.

You can see there is a USB interface out here, from here you can connect a USB cable. And you can see the microcontroller chip sitting here, and there are other accessories. Now, let us see what these different subsystems contain.

If you see here this is the same board that I had shown. In this board these black connectors are the Arduino connectors that we were talking about. Now, in this board you will see that some labels are given A5, A4, A3, A2, A1, A0 these are the analog input ports; the A/D converter I told you you can connect an analog input directly to these ports. There are some voltage sources available 5 volts, 3.3 volts, ground these are available, and different input output port pins are available here. Some of them are digital data pins, some of them are pulse width modulation pins; these we shall see later.

Over here, STM32 microcontroller pins are available that come in directly from the STM32 microcontroller. All the port pins are available here. There are three ports A, B, C; they are 16-bit ports and there are many other signal lines.

In addition you can see on top we have the USB interface, from here you can connect the USB cable you can connect it to the PC, there is a reset button sitting here you can reset the machine. And for indication there are some small LEDs connected, one LED is here, one LED is here, one LEDs here, and there are some also push button switches.

Other than the reset button there is another push switch here which you can use in a program. We can write a small program and take input from there. Now the thing is that whenever you are using this board to develop an embedded application you will have to make some connections with this board. You must understand what signal lines are available over these connectors, because you will have to make all your connections through these connectors. Having them connected to their Arduino pins will be easier because, most of the time you will be needing only a few connections, but beyond that you may have to have access to the detailed connector pins which are available here.

Let us see those pins quickly into have some idea that that what kind of pins are there.

(Refer Slide Time: 16:47)

The slide is titled 'Some Color Conventions Followed' and contains two main sections: 'Labels usable in code' and 'Labels not usable in code (for information only)'.
Labels usable in code:

- PX_Y** MCU pin without conflict (blue box)
- PX_Y** MCU pin connected to other components (gray box). A note below says 'See PeripheralPins.c (link below) for more information'.
- XXX** Arduino connector names (A0, D1, ...)
- XXX** LEDs and Buttons (LED_1, USER_BUTTON, ...)

Labels not usable in code (for information only):

- XXX** Serial pins (USART/UART) (yellow box)
- XXX** SPI pins (green box)
- XXX** I2C pins (orange box)
- XXX** PWMOut pins (TIMER n/c[N]) (purple box). A note below says 'n = Timer number c = Channel N = Inverted channel'.
- XXX** (AnalogIn)ADC and AnalogOut pins (DAC) (orange box)
- XXX** CAN pins (purple box)
- XXX** Power and control pins (3V3, GND, RESET, ...)

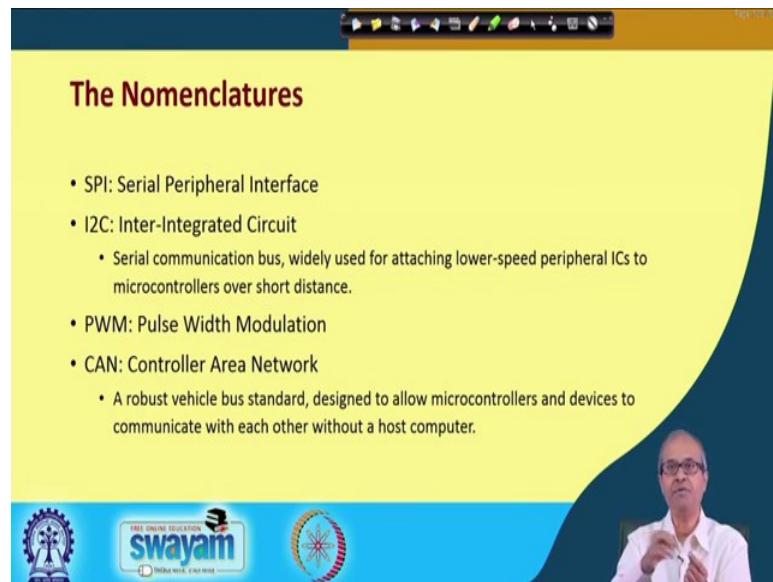
At the bottom of the slide, there is a logo for 'FREE ONLINE EDUCATION swayam'.

Before that there are some color conventions that are followed in those pin descriptions. See the various colors you can see, they indicate different categories of pins. Like this blue color indicates the microcontroller pins without conflict; without conflict means they are available to the user, they are not connected anywhere else. But, these gray colors ones are some pins connected to other components. Maybe you have seen in the board there are some LEDs, some switches, so many things are there maybe some of these pins are internally connected to those devices. So, when you use them from outside you should be careful.

In addition these yellow ones refer to some serial pins that are connected to USART – universal serial asynchronous receiver transmitter for serial communication. Then there are SPI pins, I2C pins, PWM pins, and this Arduino connector pins I have already mentioned. Of course in this board there are no analog output pins, only analog input pins are there and there is something called CAN pins.

We shall be seeing what these acronyms mean SPI, I2C, CAN and of course, there are some power pins 3.3 volt, 5 volts, ground.

(Refer Slide Time: 18:50)



The Nomenclatures

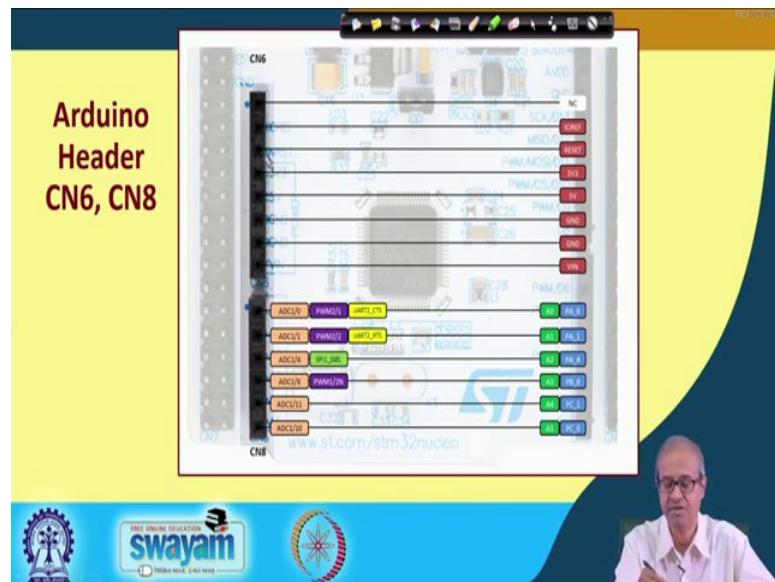
- SPI: Serial Peripheral Interface
- I2C: Inter-Integrated Circuit
 - Serial communication bus, widely used for attaching lower-speed peripheral ICs to microcontrollers over short distance.
- PWM: Pulse Width Modulation
- CAN: Controller Area Network
 - A robust vehicle bus standard, designed to allow microcontrollers and devices to communicate with each other without a host computer.

But, first let us look at the nomenclature. SPI stands for Serial Peripheral Interface. You can use these pins to develop some serial communication application whenever you want.

I2C is a standard bus, this means Inter-Integrated Circuit. This is a serial communication bus that is very frequently used to connect some low speed devices to a microcontroller over very short distances. For embedded system application this I2C interface can be very useful, if we have a device that is I2C compatible then you have to have this I2C connection.

PWM stands for Pulse Width Modulation, this we shall be demonstrating. Pulse width modulation is a very convenient way of controlling external devices. And CAN is the short form for Controller Area Network.

(Refer Slide Time: 20:40)

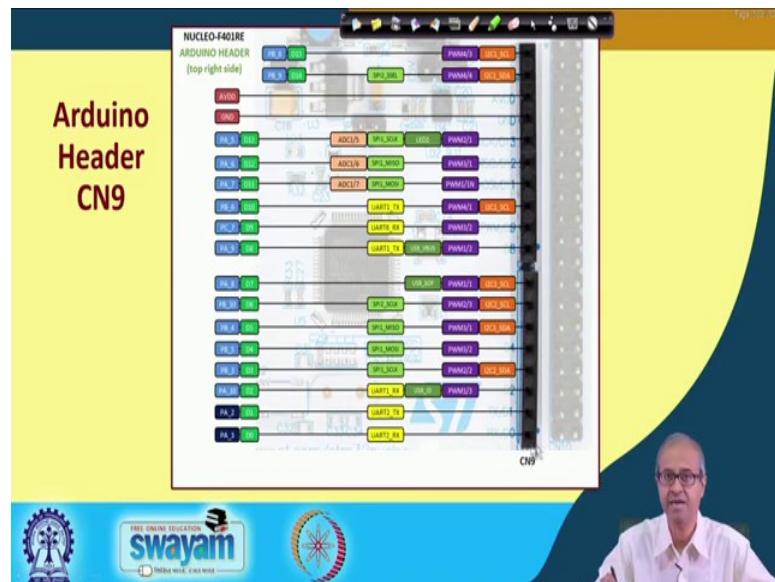


Let us have a very quick look at the different connectors that we have already seen in the board. Many of them are very small you may not be able to read very clearly, but I am telling you. On the left side of the board these are the Arduino connectors the lower one is called CN8, the upper one is called CN6 connector.

In CN8 you will see there are the analog input ports, six of them are provided some of the pins are multifunctional. They can either be used as an analog pin or they can be used as a PWM pin or they can be used as a UART pin. As I told you in ARM the philosophy is that they do not use dedicated pins for different functions, same pin you can use for different applications. And these are actually connected to these ports A0, A1, A4, B0, B1, C0.

Similarly, the upper lines are primarily power supply lines, 5 volt, ground, 3.3 volt, reset and so on. These are the Arduino connector first set.

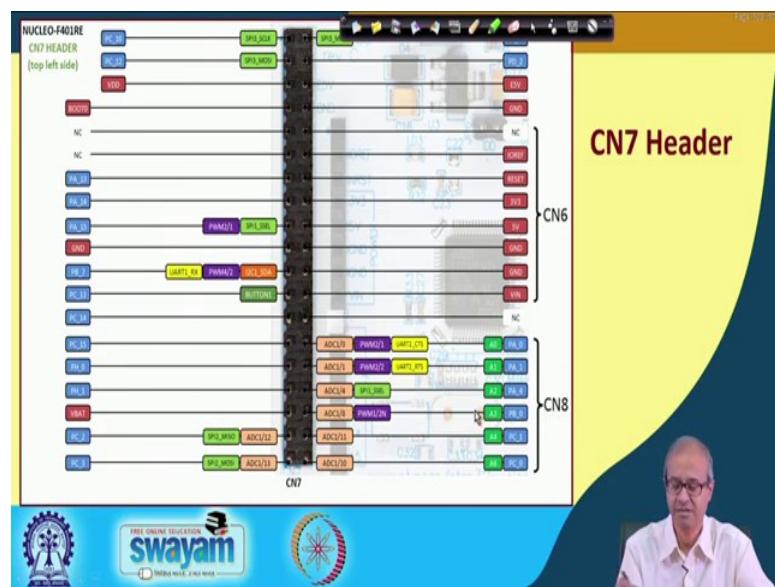
(Refer Slide Time: 22:25)



And, the second set of Arduino connectors that appears on the right side is called CN9 connector, this is a larger sized connector. This contains many signals as you can see, like PWM ports, I2C ports, these are USB, UART pins, these are some additional analog input pins, and these are some port pins.

So, these are the pins that you will be using when you are developing an application.

(Refer Slide Time: 23:39)

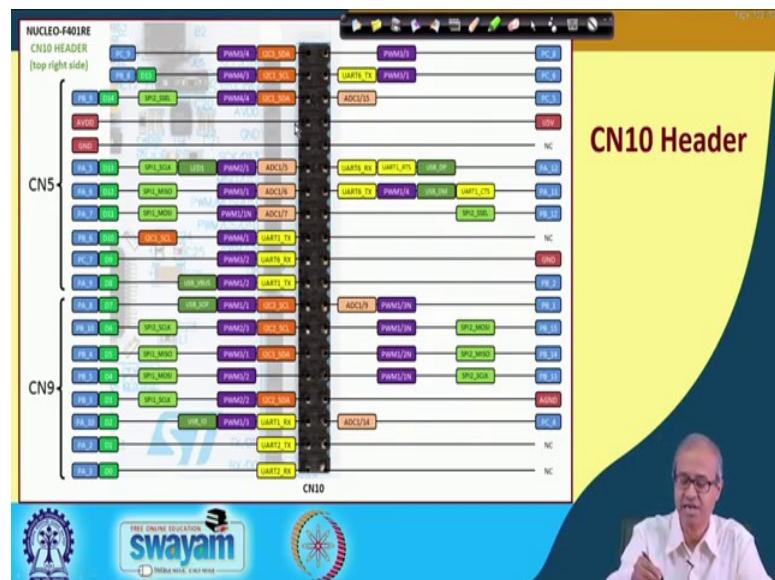


These are the detailed pins that are coming from the microcontroller sitting inside the board. On the left side this is called CN7, this is the big connector, those pins you have

seen, and here the Port A, Port B, Port C -- there are three 16-bit ports which are available in the microcontroller. All those 48 pins are available over these connectors.

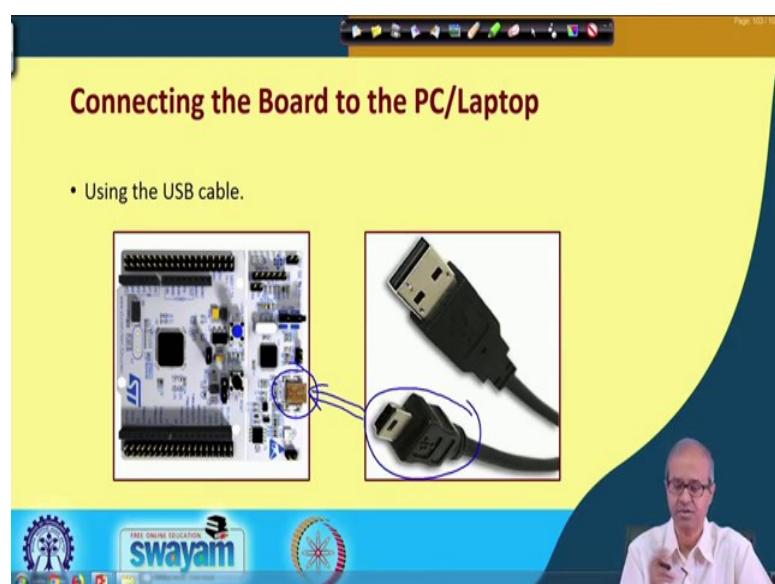
In addition there are so many other things like power supply, etc.

(Refer Slide Time: 24:36)



Similarly, on the right side there is another port this is CN10. Here also you have Port B, Port A, PWM ports, ADC ports, SPI, I2C and so many ports are there. Whatever you need to use and connect you can use from this connectors.

(Refer Slide Time: 25:13)



You need a USB cable like this, the shorter side will be connected here and the other side will be a standard type C connector, this will go into your PC or laptop. This is how we shall be doing the experiment. In the PC, we shall be developing the application, we shall be writing the program, we shall be compiling the code, and with this board we shall be interfacing with all the devices, sensors, actuators everything. And, after writing the code we shall be downloading the code on the board and, whatever you have downloaded, it will start running.

With this we come to the end of this very short introduction of Nucleo board. As part of this course, we shall be showing you some demonstrations. But, if you are really interested to learn embedded system design in a hands-on way we strongly recommend that you should procure some of these boards, and some of the experiments that we shall be showing or demonstrating you should actually do them yourself.

We shall be continuing with our discussion. In our next lecture, we shall be talking about some of the IO port details of this board that we have just now seen, specifically we shall be talking about the pulse width modulation ports, the interrupt lines and so on. And subsequent to that we shall be looking at the analog inputs, outputs and some other applications and demonstrations.

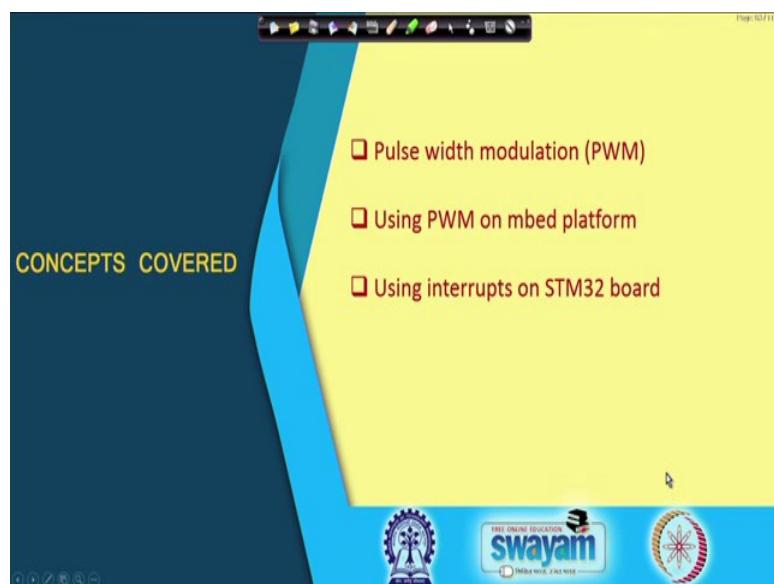
Thank you.

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

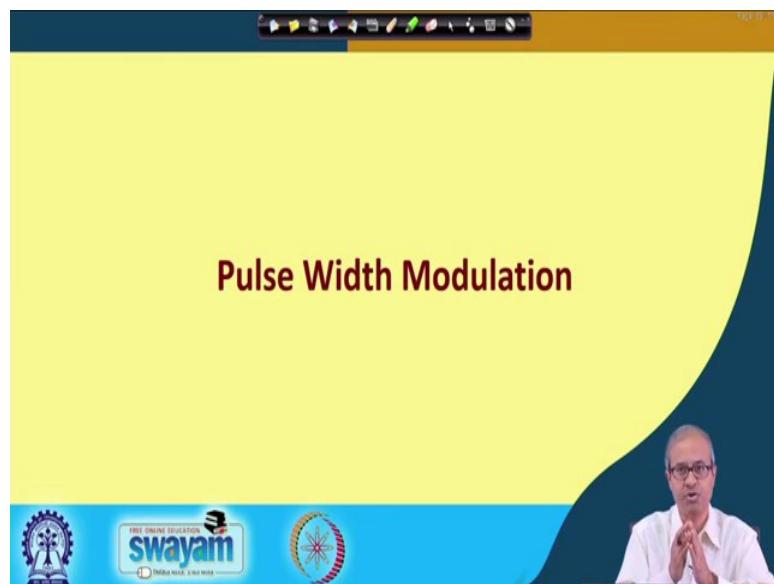
In this lecture, we shall be talking about some of the unique input output features that are available in the STM32 board which you saw in our last lecture.

(Refer Slide Time: 00:43)



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 board, and lastly how to use interrupts on this board.

(Refer Slide Time: 01:08)

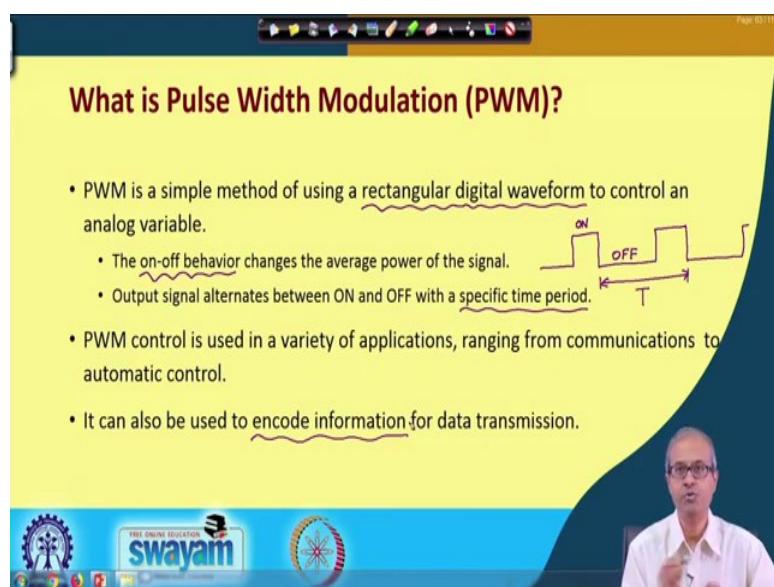


Pulse Width Modulation

The video slide features a man in a white shirt speaking. The background is yellow and blue. The Swayam logo is visible at the bottom.

First let us talk about pulse width modulation. It is a technique where you can encode the value of some kind of digital value into an equivalent analog value, but not directly; there is an indirect way of encoding it. We shall understand it as this lecture proceeds.

(Refer Slide Time: 01:40)



What is Pulse Width Modulation (PWM)?

- PWM is a simple method of using a rectangular digital waveform to control an analog variable.
 - The on-off behavior changes the average power of the signal.
 - Output signal alternates between ON and OFF with a specific time period.
- PWM control is used in a variety of applications, ranging from communications to automatic control.
- It can also be used to encode information for data transmission.

The video slide features a man in a white shirt speaking. The slide contains a list of bullet points and a diagram of a rectangular waveform with 'ON' and 'OFF' states and a period T.

First let us see exactly what pulse width modulation is. The first thing is that using PWM we can generate some kind of rectangular digital waveform, and there will be a particular time period T.

PWM helps us in bringing some kind of feature that 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. The matter of interest is that for how long our waveform is ON and for how long it is OFF. The 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.

PWM can be used in a variety of applications. We can use for 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 the scope of discussion here we do not need to know that.

(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.

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

Diagram illustrating a rectangular waveform:

- "on" time, t_{on}
- Average value
- Period, T

The slide also features a logo for 'swayam' and a photo of a speaker in the bottom right corner.

Now, let us see how exactly PWM works. In this waveform which is being generated by PWM the time period is kept constant. Once we specify this time period T , this will be kept constant. Now, depending on the parameter we want to encode there is something we are trying to vary. This is the pulse width or the ON time let us call it t_{on} ; this t_{on} is something we are varying in proportion to the parameter we want to encode.

In this respect we define something called the duty cycle of this rectangular waveform. It is the proportion of 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 100. In this diagram the pulse on time is t_{on} divided by capital T multiplied by 100 that will give you the duty cycle.

(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.

Diagram of a square wave waveform:

- 'on' time, t_{on}
- Average value
- Period, T
- 5V
- 0V

• Average value of the signal = $\frac{1}{T} \int_0^T f(t) dt = t_{on}V_H + (1 - t_{on})V_L$

• In general, V_L is taken as 0V for ease of calculation.

- Average value becomes $t_{on}V_H$

FREE ONLINE EDUCATION **swayam**

Page 131 | 100

Another thing to note with respect to PWM is that whatever duty cycle you have, 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.

Let us assume this voltage is varying from 0 volt up to some maximum let us say 5 volts. The average value is shown by this dotted line in between. This average value will very much depend on how much time the pulse is on. More the on period this dotted line will be moving up, but if the pulses are very narrow then this dotted line will move down.

Essentially by adjusting the duty cycle we are adjusting the average value of the waveform. 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 $f(t) dt$. If you 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. It is high for t_{on} period of time, it will be low for $t_{off} = 1 - t_{on}$ period of time, because we are dividing by T.

For most cases the low level of voltage is taken to be 0 volt. So, if $V_L = 0$, the second term will disappear. So, your average value will be only $t_{on} \cdot V_H$. So, by adjusting t_{on}

the average value will be becoming proportional to the on period or the duty cycle indirectly. 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)

The slide is titled 'How to Extract the Average Value?' in a maroon font. Below the title is a bulleted list:

- 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.

Below the list is a circuit diagram showing a resistor (R) in series with a capacitor (C) connected in parallel to the output V_{out} . The input is a square-wave PWM signal labeled V_{in} . To the right of the circuit, a text box states: 'In practice, the filter is not always required. Many physical systems have response characteristics that act like low-pass filters.' At the bottom of the slide, there is a logo for 'swayam' and other educational icons.

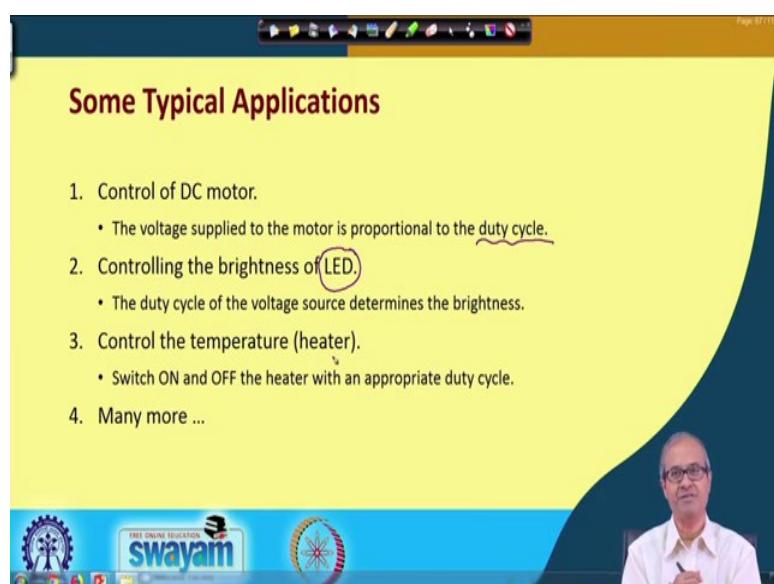
Now, from this PWM waveform, how can we extract the average value? The simplest way is you can use a low pass filter. The simplest kind of low pass filter is an RC circuit, a resistance in series and a capacitor in parallel. If you apply a pulse waveform, you will be getting an output voltage which will be the DC value; provided the RC time constant is chosen in a suitable way such 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.

The interesting point to notice that this kind of a circuit with PWM you can use in place of a digital to analog converter. A digital to analog converter that 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 the 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 we are adjusting the pulse width of the PWM waveform.

The point to note is 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.

(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 ...

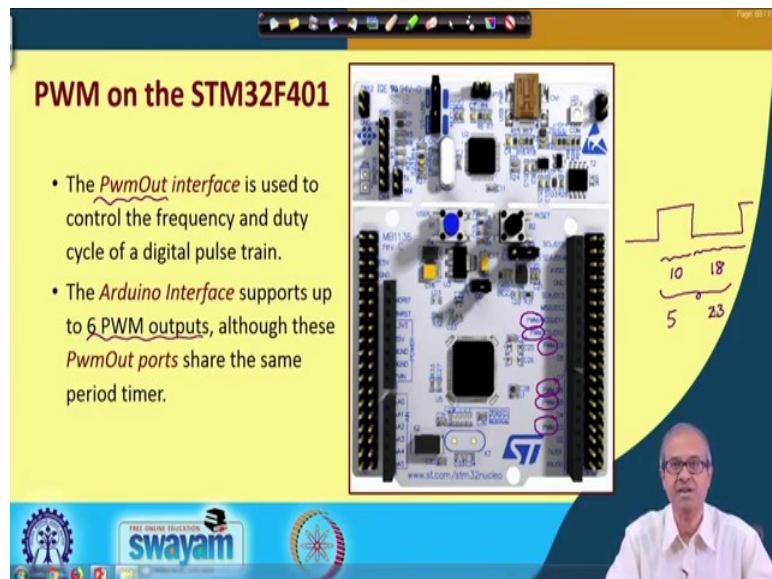
Some typical applications of PWM are listed here. Suppose you are trying to control a DC motor. For this there is often a control signal with the help of which you can adjust the power supply you are applying to the motor.

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 adjusting the speed of the motor. This is one application you think of.

Think of a gadget like heater or even you can think of microwave oven. When you adjust the power of the microwave oven what actually happens? 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 in an automatic heater control system. You turn ON and OFF

the heater, the time you are turning ON the heater will be 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)



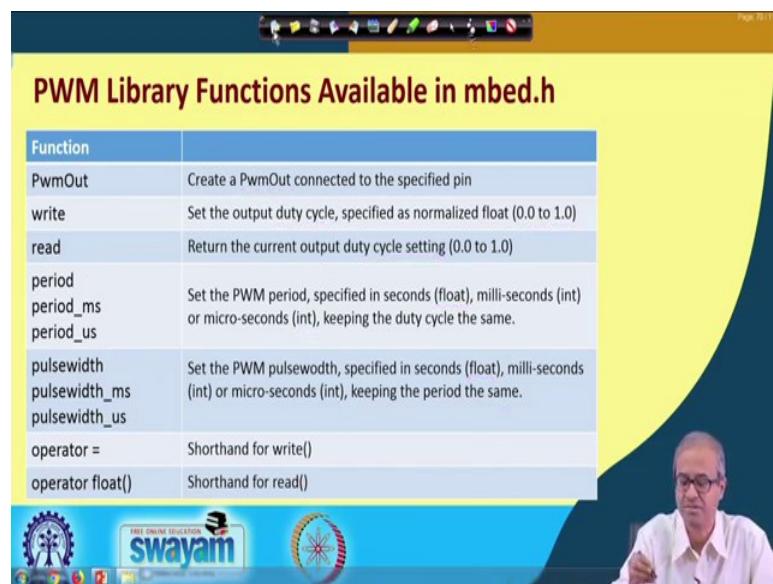
If you revisit the STM32F401 board, if you look at the Arduino connector you will see there are several PWM pins mentioned. There are six such PWM pins. There is a standard PWMOut interface that we shall be using in some kind of software development platform called mbed, this we shall be seeing later. Under mbed this PWMOut is a standard library that is provided, it helps us in developing applications that use a PWM.

Now, this interface has some functions that can be called to adjust the duty cycle of the PWM waveform that has been generated. Arduino interface supports up to 6 PWM outputs, but if you have access to these detailed pins there are many more. So, you can have more than 6 PWM output pins also.

Essentially this PWM signals, ON and OFF, are generated using some timers. You can put the output to 1, 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. 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.

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. These adjustments are done automatically as I adjust the duty cycle.

(Refer Slide Time: 16:19)



PWM Library Functions Available in mbed.h

Function	Description
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	Set the PWM period, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the duty cycle the same.
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	Set the PWM period, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the duty cycle the same.
pulsewidth	Set the PWM pulsewidth, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the period the same.
pulsewidth_ms	Set the PWM pulsewidth, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the period the same.
pulsewidth_us	Set the PWM pulsewidth, specified in seconds (float), milli-seconds (int) or micro-seconds (int), keeping the period the same.
operator =	Shorthand for write()
operator float()	Shorthand for read()

Page 70/101

FREE ONLINE EDUCATION **swayam**

Let us look at the functions that are available as part of the PwmOut library. This PwmOut is 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 D1, D2, D3, etc. Whatever PWM pin you will see it is written as PWM/D3. I will take some example.

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. This means it is not multiplied by 100, just on period divided by the total period, it is specified as a fraction. If you write it as 0.5, it will be equal ON and OFF period.

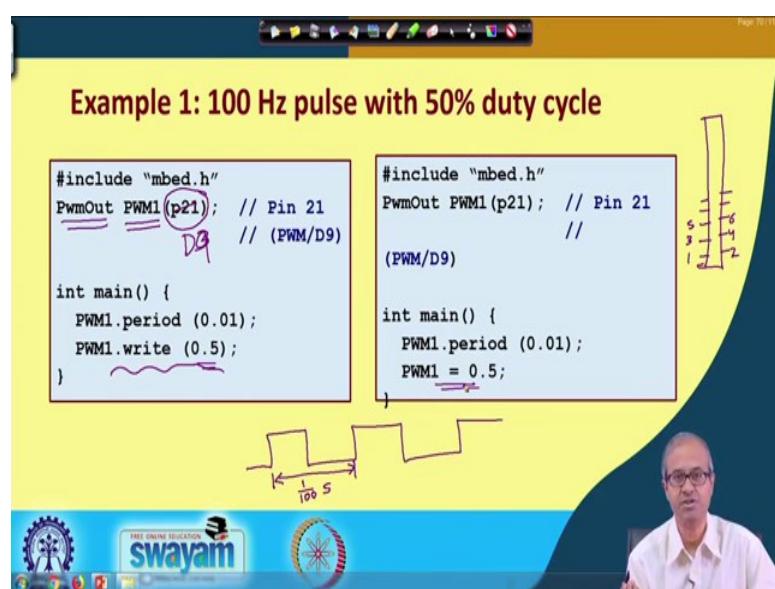
Similarly, you can read the current value of the duty cycle setting. Suppose, in a program you do not remember what was the duty cycle you had set, you can call this function read and know that. And, we can set the time period; there are three functions available: period, period_ms, period_us. The first one specifies the time period in seconds, second one specifies in milliseconds, third one in microseconds. The only point to note is that when you specify it in seconds, your parameter will be a floating point number, you can

write 2.5. But, when you specify in milliseconds or microseconds, the parameters will be integer. So, no fractional parts are allowed you should remember this.

You can set your period, by write of the duty cycle. Instead of specifying the duty cycle there is another way of having the control with yourself; you can specify the pulse width t_on. Here also you can specify in seconds, milliseconds or microseconds. Again in seconds it will be floating point, milliseconds and microseconds it will be integers.

Now, for write there is a shortcut. If you straightaway give the name of that object equal to something, which is equivalent to write. We will take some examples.

(Refer Slide Time: 19:39)



Suppose, we want to generate a 100 Hz pulse with 50% duty cycle. What is the meaning of 100 Hz? 100 Hz means $1 / 100$ second = 0.01 second will be the time period. I am showing two alternate codes. I have to include this mbed.h header because many of the functions I am using are all included there. This PwmOut is the PWM object I told you. We will have to create an instance of this class, you create an object you call it PWM1 and p21 is the name of the pin.

You will know p 21 with respect to the detailed connector where you see there will be one connector on either side. 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/D3. So, instead of p21 you can also write D3 straightaway.

In the main function this is a C code I am writing. I am calling the two functions period and write, 0.01 means 100 Hz, and I am specifying the duty cycle as 0.5.

The second code is the same one where instead of calling write I am using the shortcut. If I write straight away $\text{PWM1} = 0.5$, this means actually we are writing the duty cycle 0.5, this is just a shortcut. This is how you can generate a PWM waveform.

(Refer Slide Time: 22:14)

Example 2: 1 KHz pulse with 80% duty cycle

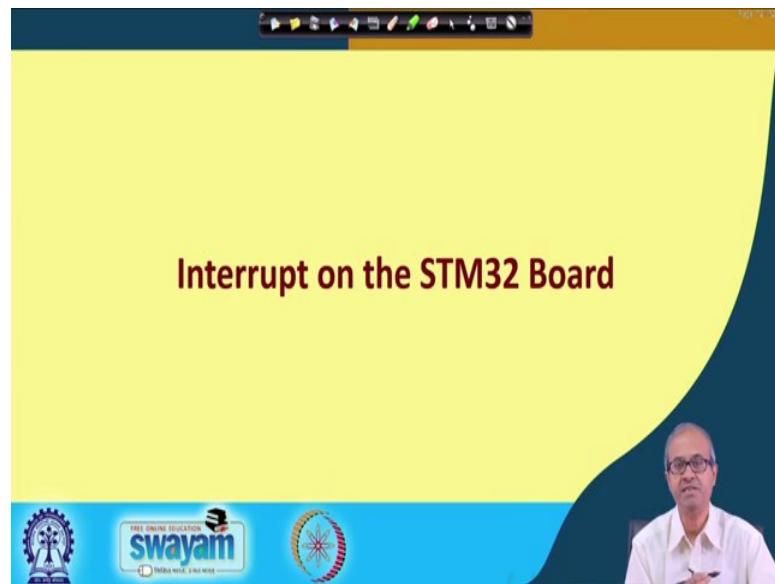
```
#include "mbed.h"
PwmOut PWM1(p21);      // Pin 21
                      // (PWM/D9)

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

The waveform diagram shows a square wave with a period of 1 millisecond (labeled '1 msec'). The duty cycle is 80%, meaning the high time is 0.8 milliseconds and the low time is 0.2 milliseconds.

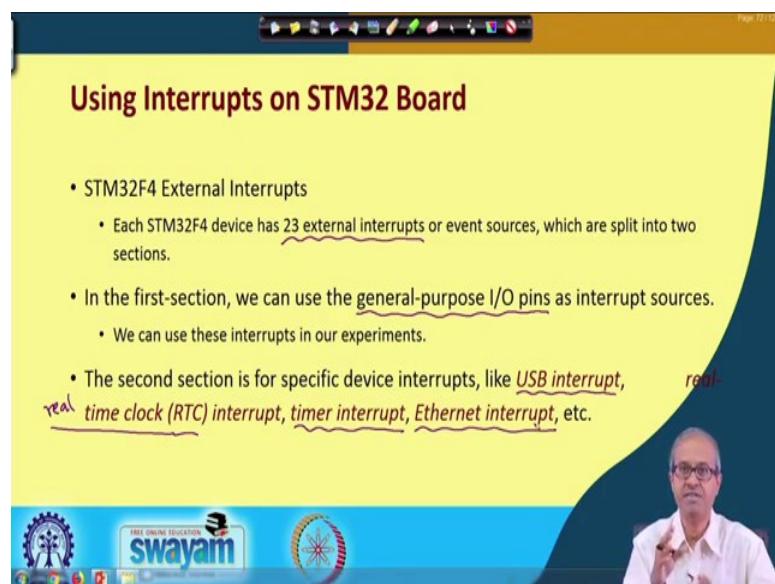
Take another example. This is a 1 KHz pulse with 80% duty cycle. Here the waveform will look something like this. 1 KHz means the time period will be 1 milliseconds, and 80% will be the duty cycle. In the same way you specify a PWM pin, you specify the period in milliseconds. And duty cycle 0.8. It will be generating a waveform like this. This we shall be seeing in the demonstration later.

(Refer Slide Time: 23:12)



Now, let us see what are the kinds of interrupts that are available on the STM32 board. In many applications wherever you are interfacing some devices, there will be some devices that can be generating some interrupt signals from outside. I mentioned that in ARM processors there are many interrupting source; in fact, any of the port lines you can use as an interrupt input.

(Refer Slide Time: 23:39)



Let us look into a little more detail.

For the STM32F4 general family of board, there are 23 external interrupts that are supported. These 23 external interrupts 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 which can be used as interrupt inputs. In the actual experiments you shall be using these pins only to connect interrupting sources.

But there is another section where some specific devices are generating interrupts. Like your USB interface can generate an interrupt, your real time clock that maintains the time of day can generate an interrupt, some timer which you are using can generate an interrupt, if you are having a network interface like Ethernet that interface can also generate an interrupt. These are device specific interrupts that fall in the second category, but first category is more general purpose. In our experiments we shall be using the first category.

(Refer Slide Time: 25:17)

General-Purpose I/O (GPIO) Pin Interrupt Lines

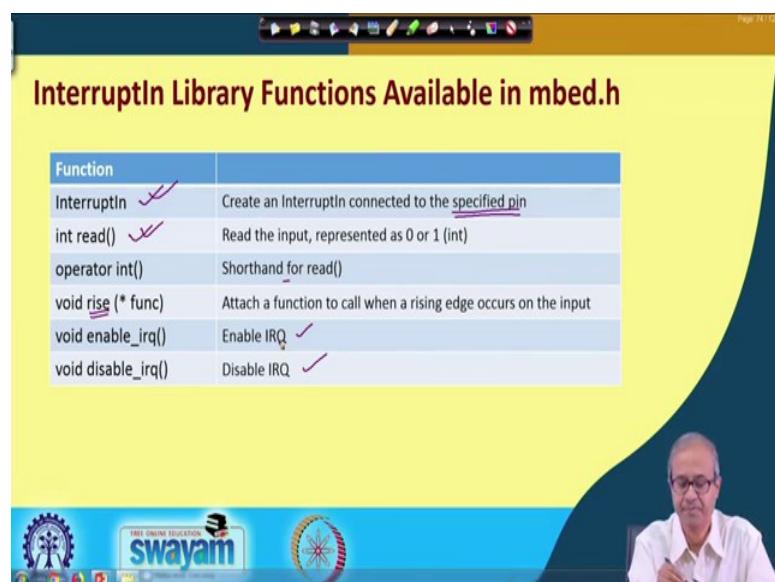
- The 16 interrupt lines are referred to as Line0 to Line15.
- Three 16-bit GPIO ports: PA, PB and PC.
- Line0 is connected to PA0, PB0, and PC0.
- Line1 is connected to PA1, PB1, and PC1.
- Line2 is connected to PA2, PB2, and PC2, and so on.
- We cannot use PAx, PBx, PCx as interrupt sources together for the same value of x.

In the first category these are called general purpose IO or GPIO interrupt lines. Under this category broadly speaking we say that there are 16 interrupt lines. These interrupt lines are referred to as Line0 up to Line15. You should remember that in this ARM board 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.

The point to note is that these 16 interrupt lines 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 PA2, PB2, PC2, and so on. 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. This is something you have to remember. There are 16 interrupting sources you can use at a time, but all the 48 port inputs you can use for the connection.

(Refer Slide Time: 27:28)



InterruptIn Library Functions Available in mbed.h

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

For the interrupt the library functions that are available to us are as follows. InterruptIn is the class, you create an object of this type. 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, it will come as either 0 and 1 of type integer.

If you just use the name of the object you are creating, that is a shortcut for read. There is a function called rise, where a parameter you give is a pointer to a function; this means if there is a rising edge on the interrupt input automatically that function will be called. This essentially functions like your interrupt service subroutine or interrupt handler.

When you are writing a C program you can 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. This will enable and disable interrupts if you want.

(Refer Slide Time: 29:04)

```

#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) {
    }
}

```

Let us show a simple example. On the left I am showing the code of a simple example using interrupts. Suppose this is your microcontroller board. Let us say on line D3 I have connected a switch. Typically we connect a 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. If this switch is open, high voltage or logic 1 will come to D3, if it is pressed it will be shorted to ground, and logic 0 will come.

And, on the other side in D2 we are connecting a LED. We are connecting a resistance, then you are connecting a LED, 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 at same voltage, and LED will not glow. This is how I have made the connections. What I want is that every time this switch is pressed the glowing status of the LED should change; that means, on off on off like that.

In the program, as usual we have included this mbed.h, then we have created an object of type InterruptIn, we have given the name switch, 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 port number D2.

This is my main function. Here we have first made a call to this rise, switch is the object you have created. switch.rise means I am calling this function in connection with the object switch and I have given &toggle means address of toggle; toggle is the name of a function. This means 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 where your main program is waiting.

Every time there is a switch press means an interrupt is coming what will happen? Toggle will just do led = !led. If led was 0, it will become 1; if it was 1 it will become 0. So, for every switch press toggle will be called and the status will change.

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.

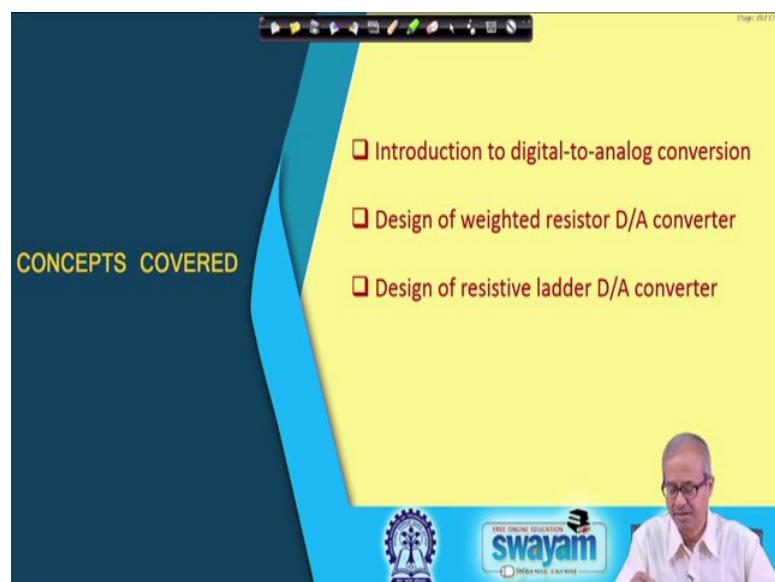
Embedded System Design with ARM
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 12
Digital to Analog Conversion

When you are trying to design and implement an embedded system, you need to interface with the outside world; there will be various kinds of sensors, there will be various kinds of actuators. So, whatever voltages are coming from outside they are very rarely digital in nature, most of them are analog. Similarly when you are activating the actuators, you often need to provide an analog control signal for those. For that reason you need to have devices called digital to analog converter and also analog to digital converter to interface with your microcontroller processing unit

The topic of this lecture is Digital to Analog Conversion. Here we shall study the different ways in which digital to analog conversion can be carried out.

(Refer Slide Time: 01:15)



In this lecture we shall be talking about some of the basics of digital to analog conversion and we shall be discussing two different alternate design styles to implement such D/A converters.

(Refer Slide Time: 01:37)

Interfacing with the Analog World

- a) Transducer/Sensor: converts physical variable to electrical variable.
- b) Analog-to-digital converter (ADC)
- c) Digital System (microcontroller)
- d) Digital-to-analog converter (DAC)
- e) Actuator

Physical variable → Transducer → ADC → Digital System → DAC // Actuator → To control physical variable

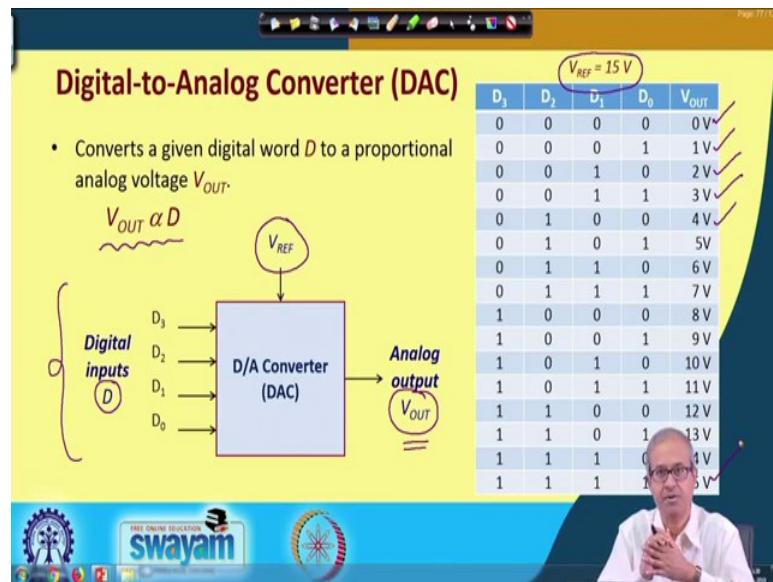
Page 70/101

FREE ONLINE EDUCATION **swayam**

To interface with the analog world, I am showing a schematic diagram here. The microcontroller can be sitting here, this is the actual processing unit or digital system. The physical variable that you are sensing from outside, you need to use some kind of a sensor, it is sometimes called a transducer to convert it into a voltage. And an A/D converter will convert this voltage into a digital input and this microcontroller can read the digital input directly.

And similarly when it wants to activate the actuator it provides with a digital output which through a D/A converter it is converted into a voltage; it is a analog voltage that is fed to an actuator. These are the basic components of a typical embedded system.

(Refer Slide Time: 02:57)



What is a digital to analog converter or a D/A converter? You look at this block diagram. In the input we apply a digital word. In this example, I am showing it is a 4 bit word $D_0 D_1 D_2 D_3$. Let us call this as D ; it is the decimal equivalent of this number. And in the output, we generate an analog voltage. Analog voltage means a continuous voltage which will be proportional to the digital value I am applying as the input. So, this V_{OUT} will be proportional to D , and often in this D/A converter, we apply some reference voltage which will determine what will be the maximum value of V_{OUT} .

In this table I have given a very small example. I have assumed the reference voltage to be 15 volts; this is a 4 bit D/A converter. So, there can be 0 0 0 0 up to 1 1 1 1, or 16 combinations, and the output voltage can be like this. 0 0 0 0 means output is 0 volts, 1 volt, 2 volts, 3 volts, 4 volts, up to 15 volts. The output voltage is clearly proportional to the decimal equivalent of the input digital word. In this case for every increase in D , there is a 1 volt increase in the output.

(Refer Slide Time: 04:53)

Resolution or Step Size

- Smallest change that can occur in $\underline{\underline{V_{OUT}}}$ as a result of a change in input $\underline{\underline{D}}$.
- Equal to the weight of the LSB, also called *step size*.
- Same as the constant of proportionality in $\underline{\underline{V_{OUT} \propto D}}$

$D: \underline{\underline{i \text{ to } i+1}}$

$V_{out} = \underline{\underline{k \cdot D}}$

0100

This increase in output for every one value increase in D is sometimes called resolution or the step size of a D/A converter. How do you define the resolution? This is the smallest change that can occur in the output voltage as a result of a change in D. Now D is a digital input. So, in D we are applying 0, 1, 2, 3 4. Whenever the value of D changes from any value i to the next value i+1, the change in the output analog voltage V_{OUT} is the step size or resolution. Sometimes we refer to this as the weight of the least significant bit because, when the least significant bit changes that also means an increase of 1.

So, V_{OUT} is proportional to D. The constant of proportionality is this step size. So, whatever value you are applying to D, that step size multiplied by D will be the actual voltage you will be getting.

(Refer Slide Time: 06:37)

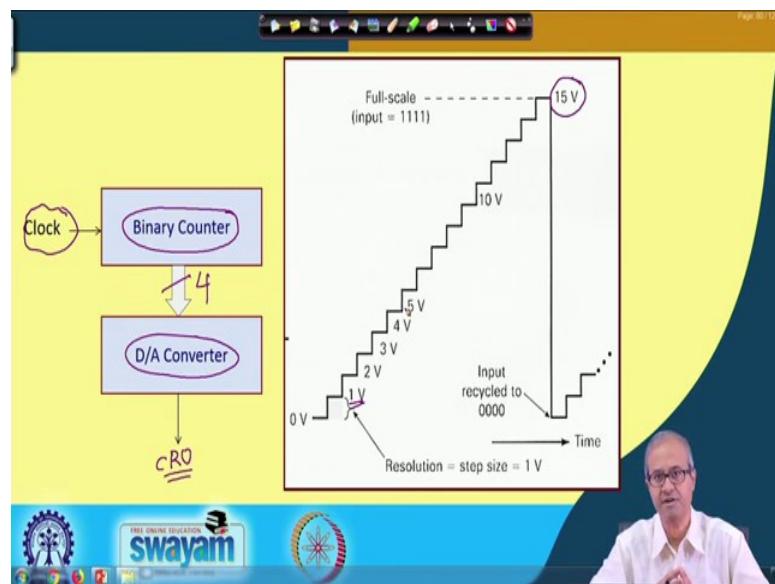
Resolution or Step Size

- Smallest change that can occur in V_{OUT} as a result of a change in input D .
 - Equal to the weight of the LSB, also called *step size*.
 - Same as the constant of proportionality in $V_{OUT} \propto D$
- Can also be defined as a percentage of the full-scale voltage:
Step-size or resolution $\Delta = \frac{V_{REF}}{2^N - 1}$, for an N -bit DAC
Percentage resolution = $\Delta / (\text{full-scale voltage}) \times 100\%$
 $= \frac{1}{2^N - 1} \times 100$

Resolution can also be defined as a percentage of the full scale voltage. The step size or resolution if you call it Δ , this can be defined as the full scale voltage V_{ref} divided by the total number of steps of the D/A converter. Like for a 4-bit converter you could go from 0 0 0 0 up to 1 1 1 1 which is 15. For an N -bit converter you can go up to $2^N - 1$. So, the total voltage divided by the number of steps will be the height of every step or resolution. This is how you can define the resolution or step size for an N -bit D/A converter.

If we want to express it as a percentage, what do you do? We divide this Δ by full scale voltage which is V_{ref} . It will be 1 divided by $(2^N - 1)$ into 100; this is the expression for percentage resolution. If you are asked to compute the percentage resolution of an N -bit D/A converter, you will be using this formula.

(Refer Slide Time: 08:03)



Suppose we have a D/A converter like this, and to the input of the D/A converter we have connected a binary counter which counts up and we apply a clock continuously. The counter will be counting up as 0, 1, 2, 3, 4 like that, let us say this is a 4-bit DA converter. So, there are 4 number of lines here and it is a 4-bit counter.

As the clock counts the counter will continuously go from 0 to 15 and then again it will go back to 0. If I observe the waveform at the output of the D/A converter, I will see a waveform like this. The output will go step by step from 0 to 15 and then again it will go back to 0, again 1 2 3 4 like this.

Now, depending on V_{ref} , the step height will be adjusted. If $V_{ref} = 15 \text{ V}$; that means, the full scale voltage is 15V then your step height will be 1 volt.

(Refer Slide Time: 09:45)

Types of D/A Converter

- We shall discuss two different designs of digital-to-analog converters.
 - a) Weighted resistor type DAC
 - b) Resistive ladder type DAC
- The first type is easier to analyze, while the second type is more practical from the point of view of implementation.

Now, talking about the types of D/A convertor, I had mentioned in the beginning that we shall be talking about 2 types of D/A converter that is most common. One is called weighted resistor type, other is called resistive ladder type. Now the point is that the first type is easier to build and understand. But there are some problems and issues in the design we shall see, but the second type is easier to build, but it is a little more complex in terms of the circuitry.

(Refer Slide Time: 10:25)

Weighted Resistor Type DAC

- For an n -bit DAC, it consists of n different resistance values of magnitudes $R, 2R, 4R, \dots, 2^{n-1}R$ respectively.
- The resistances help in generating currents inversely proportional to their magnitudes.
- The total current is added up by an operational amplifier, and is converted to the voltage output V_{OUT} .

V \rightarrow $\frac{V}{R}$

We first start with the weighted register type D/A converter. Let us consider the design of an n-bit D/A converter; in general there are n number of bits that are coming in the input. The first thing is that to construct this kind of a D/A converter, we need n different resistance values, and this is one of the main problems in manufacturing this kind of D/A converter. You see you can very accurately manufacture 1, 2 or 3 resistance values, but if I tell you require 16 different resistance values, you have to manufacture all of them very accurately, it becomes that much more difficult. This method requires n different resistance values to be used and the magnitudes will be some base value R, 2R, 4R; that means multiples of 2; this will go up to $2^{n-1} R$.

The different resistance values are actually responsible for generating some currents. Let us say you have a resistance R here and we apply a voltage. Suppose I ground this point, so the current that will be flowing will be given by V / R . If I use n different resistance values, the current will be determined by the value of the resistance, for R it will be having some value, 2R will be having half the current, 4R will be having half of that, and so on. The current values will be changing in multiples of 2.

If I have a mechanism to calculate the total current that is finally coming out, then that will give you a final DAC output, if you can convert it into a voltage.

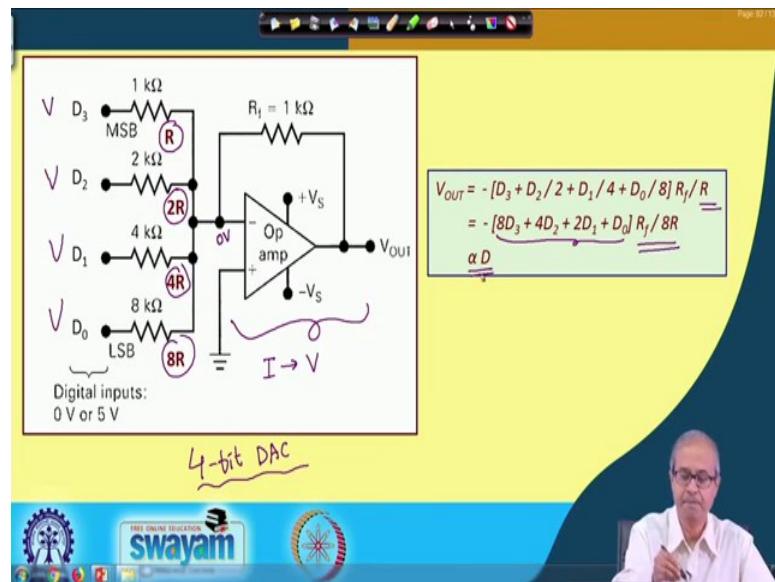
(Refer Slide Time: 12:51)

Weighted Resistor Type DAC

- For an n-bit DAC, it consists of n different resistance values of magnitudes R, 2R, 4R, ..., $2^{n-1} R$ respectively.
 - The resistances help in generating currents inversely proportional to their magnitudes.
 - The total current is added up by an operational amplifier, and is converted to the voltage output V_{OUT} .
- Main drawback:
 - A different-valued precision resistor must be used for each bit position of the digital input.

The main drawback I already mentioned, you need to use so many resistance values.

(Refer Slide Time: 12:57)

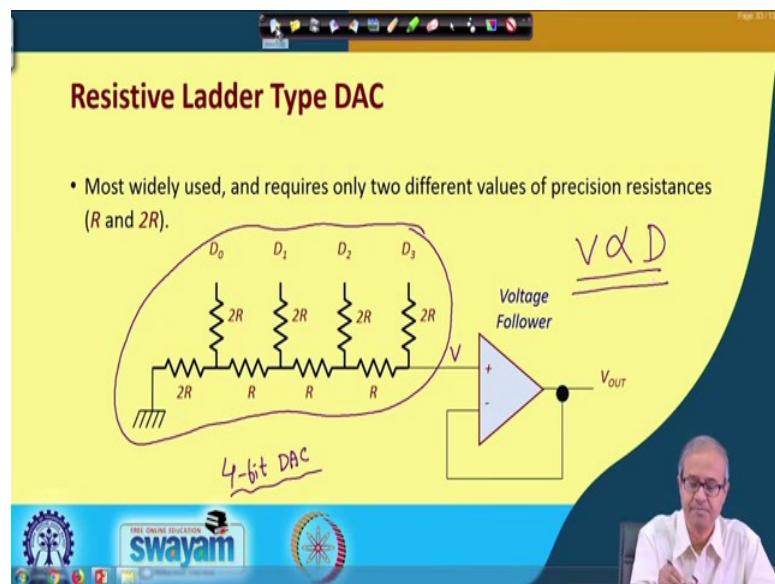


This is how this circuit looks like. I am showing a 4-bit D/A converter of weighted register type. You see the values we are using are R , $2R$, $4R$ and $8R$ --- just as an example I am showing it to be 1 kilo ohm, 2 kilo ohm, 4 kilo ohm and 8 kilo ohm. And the digital inputs are applied like this: $D_0 D_1 D_2 D_3$, least significant bit is connected to the highest resistance; most significant bit is connected to the lowest resistance. On the output side, we are using an operational amplifier. I am not going to the detail how an operational amplifier works, but basically this amplifier converts the current into a proportional voltage.

So, the currents that are flowing they will be V / R , $V / 2R$, $V / 4R$, and $V / 8R$. The expression for V_{OUT} is shown. Clearly, V_{OUT} is proportional to D .

This is how the weighted register D/A converter works. This is actually quite simple to understand. Each of the resistances is generating a current that is proportional to the weight of that particular bit and all the currents are added up by the operation amplifier, and it is converted into a voltage V_{OUT} .

(Refer Slide Time: 15:45)



Let us now come to the other type, resistive ladder. Here also I am showing a schematic diagram of 4-bit D/A converter. The first thing you note is here is that the number of resistances required are much more. In the earlier method, we were requiring only 4 resistances, but here if you need 8 resistances. But the good thing is that the values of the resistances are only of 2 types, R and $2R$, you do not need to use many different values of the resistances.

It is not that this circuit generates a current that has to be converted to a voltage; rather it directly produces a voltage proportional to the digital input. And in the output again, we are using an operational amplifier circuit, but we are connecting this op-amp in a different way as you can see. This is called a voltage follower circuit. It does not convert anything to anything, input is voltage it remains a voltage, but it actually acts like a buffer, so that when the output is connecting to some other circuit that circuit should not disturb the operation of this D/A converter circuit. Let us see how this voltage V which is generated here, is proportional to the digital input D .

(Refer Slide Time: 17:47)

Calculation

- Let us calculate the voltages at the opamp input when exactly one of the D_i inputs is at 1 (say, $+V$ volts).
- Case 1: Input is 1000

GND GND GND $+V$

$2R$ $2R$ $2R$ $2R$

$2R$ R R R

V_x

$2R$ $2R$ R

GND

FREE ONLINE EDUCATION **swayam**

We calculate the voltages at the at the op-amp input. Let us say we assume that at a time one of the input is at 1, other 3 inputs are at 0. Let us start with this assumption. The first case let us assume that the input is 1 0 0 0; that means, just one bit is 1 and the all other bits are 0. Let us see what happens here. The circuit looks like this. In this circuit the rightmost input is the most significant bit and the other 3 are the lower significant, these are all 0's.

Let us try to analyze this circuit. You see here these two resistances $2R$ and $2R$, they are connected in parallel to ground. This is equivalent to a single resistance R connected to ground. In this way let us do some simplification.

(Refer Slide Time: 19:17)

Calculation

- Let us calculate the voltages at the opamp input when exactly one of the D_i inputs is at 1 (say, $+V$ volts).
- Case 1: Input is 1000.

$$V_x = \frac{V \cdot 2R}{2R + 2R}$$
$$V_x = V/2$$

This becomes R, then that R and this R will come in series and the net resistance will again become 2R. So, again this 2R and this 2R will come in parallel, that will become R. That R and this R again in series will become 2R, this 2R and 2R in parallel will become R. This R and this R will become 2 R. So, finally, you will be having a circuit like this. This is the equivalent circuit --- 2R connected to $+V$ and this side equivalently 2R to ground.

This is a voltage divider circuit. The output voltage $V \cdot 2R / (2R + 2R) = V / 2$.

(Refer Slide Time: 20:39)

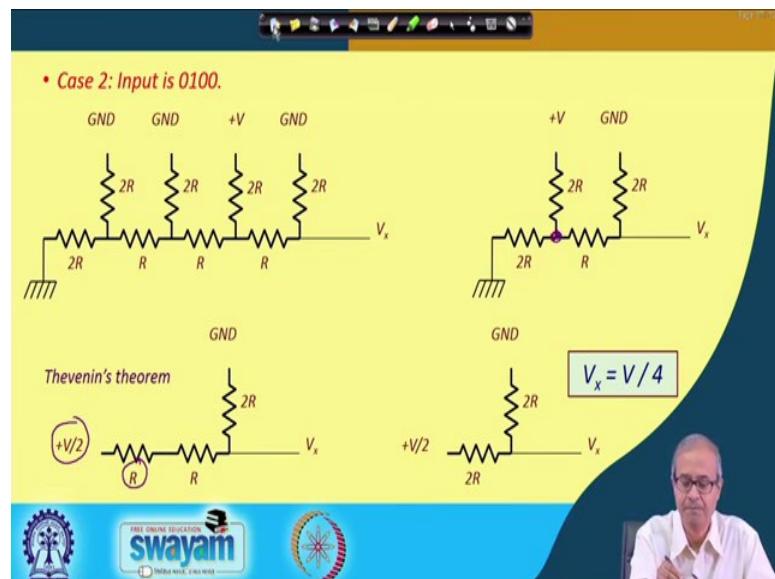
Case 2: Input is 0100.

$$V_x = \frac{V}{2}$$

Next let us assume that this second bit is 1. That means, I am applying a $+V$ here, all other 3 are ground. Following the same argument what we did for the previous case, these 2 resistances are coming in series. They become R , that R and this R in series becomes $2R$. Again this $2R$ and this $2R$ in parallel becomes R . R and R in series becomes $2R$. So, we come to a point when our equivalent circuit looks like this. I have $+V$ here, $2R$ here, around the left equivalent resistance is $2R$. See here I cannot combine them in parallel because one of them is connected to ground other is connected to $+V$. In the earlier cases both were connected to ground, that is why you could use the parallel combination formula.

Here we apply something called Thevenin's theorem. Thevenin's theorem says that if we have a circuit like this let us look at this point. This whole circuit can be replaced by a voltage source which is equal to the voltage that is coming here; you forget the remainder of the circuit, forget this part of the circuit only this part. So, the voltage will be $V / 2$, and assuming this voltage is 0, you connect it to ground and look at the equivalent; it is $2R$. So, you assume that there is a voltage $V / 2$ with a resistance $2R$ in series.

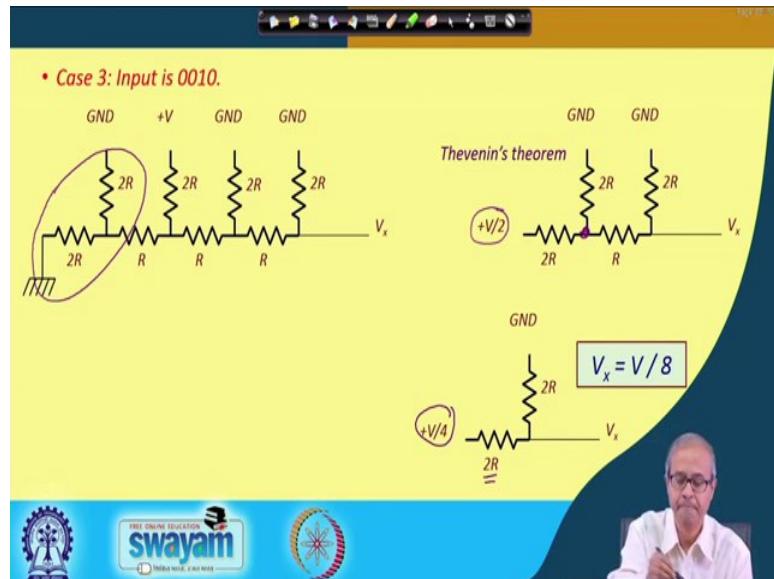
(Refer Slide Time: 23:03)



So, again R and R come in series becomes $2R$, and $2R$ and $2R$ in parallel; this is $V / 2$ here. It will become something like this: $V / 2$ here, $2R$ here, and $2R$ here to ground. Again this is a voltage divider circuit, one side $V / 2$ one side ground. So, the output will

be $V / 4$. For the earlier case when the MSB is 1, it was $V / 2$. Now when the second MSB is 1, it is $V / 4$; it is becoming half.

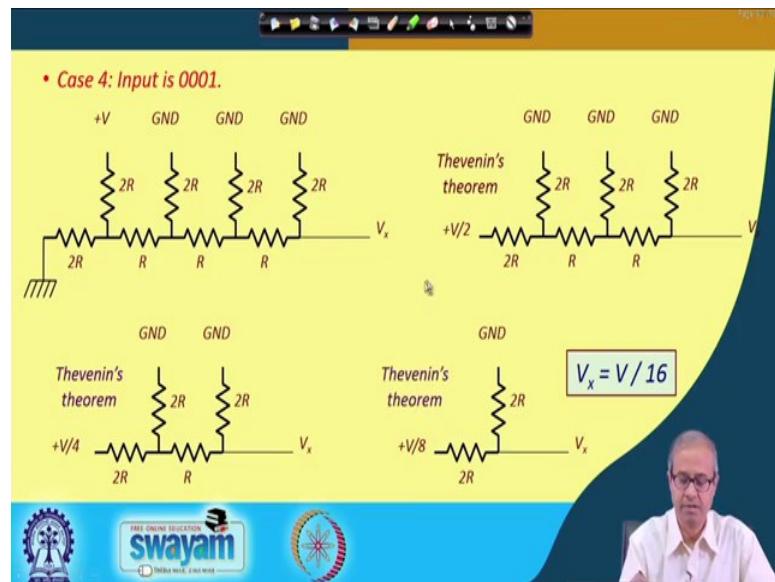
(Refer Slide Time: 24:11)



Let us look at the next bit. So, what happens when I apply 0 0 1 0? Again similarly the first 2 resistances here, you can connect in parallel and then do a series you get an equivalent circuit like this. So, you have $V / 2$ here and you combine this again apply Thevenin's theorem here. At this point you get $V / 2$ here, and $2R$ here and again you apply Thevenin's theorem.

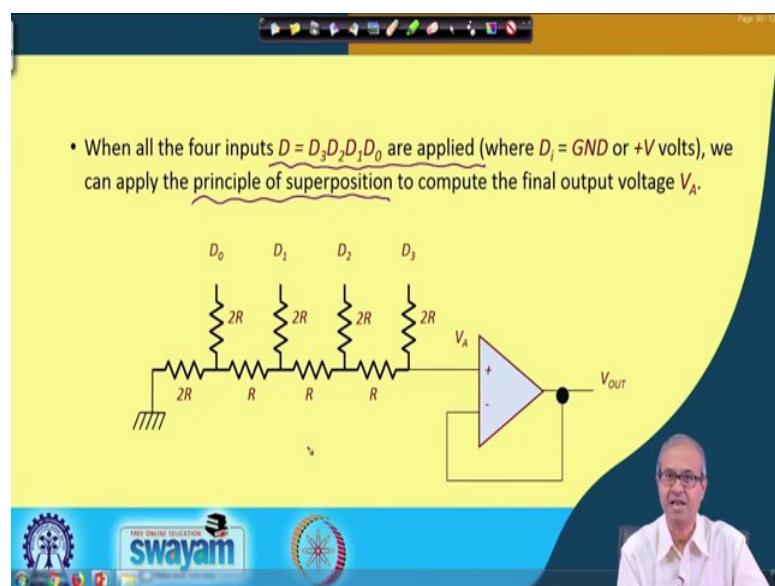
Here you get $V / 4$ here and $2R$ here. At the end you have a scenario where you have a circuit like this; again a resistance divider $V / 4, 2R, 2R$ to ground. So, the voltage will become $V / 8$. So, you see $V / 2, V / 4, V / 8$; it is progressively becoming half. Let us see whether the same thing happens for the last bit where the input is 0 0 0 1.

(Refer Slide Time: 25:19)



Here from the first step itself you have to apply Thevenin's theorem because there is ground on one side and $+V$ on other side. So, you apply $V/2$ here and $2R$ to replace this, you get a circuit like this. We apply Thevenin's theorem here again, you get this $V/4$ and $2R$. You apply Thevenin's theorem again you get $V/8$ and $2R$ and finally, you will see that V_x becomes $V/16$. So, you see as the bits are changing MSB to LSB the weight of each of the bit is becoming $V/2$, $V/4$, $V/8$, and $V/16$.

(Refer Slide Time: 26:19)



Now, the question is when all the 4 bits are applied together, what will happen? When the bits are applied together, there is a well known theorem in circuit theory called principle of superposition. It says that when multiple inputs are applied to a linear circuit; resistance is linear, then the output will be the same as the sum of the outputs where you are applying the inputs one at a time. So, whatever you are applying you may imagine that one input is being applied at a time, you calculate, add all the voltages up you will be getting the final voltage. Using principle of superposition you can use this circuit to carry out the final calculation that will give you the final output.

(Refer Slide Time: 27:23)

$$\begin{aligned}
 V_A &= [D_3 \cdot (V/2) + D_2 \cdot (V/4) + D_1 \cdot (V/8) + D_0 \cdot (V/16)] \\
 &= [8D_3 + 4D_2 + 2D_1 + D_0] \cdot (V/16) \\
 &= D \cdot (V/16)
 \end{aligned}
 \quad D_i :: \begin{cases} 0 - 0V & i = 0 \\ 1 - 5V & i = 1, 2, 3 \end{cases}$$

You will be getting an expression like this.

Clearly, V_A is proportional to D .

(Refer Slide Time: 28:11)

For an N -bit DAC, the contribution of the k -th input D_k on the output voltage will be $V / 2^{N-k}$.

Thus, $V_A \propto D$.

$$\begin{aligned} V_A &= [D_3(V/2) + D_2(V/4) + D_1(V/8) + D_0(V/16)] \\ &= [8D_3 + 4D_2 + 2D_1 + D_0](V/16) \\ &= D.(V/16) \end{aligned}$$

FREE ONLINE EDUCATION **swayam** India Niye, Jai Niye

So, for N -bit D/A converter in general if you look at the k -th bit, the contribution will be $V / 2^{N-k}$ in general.

(Refer Slide Time: 28:33)

D/A Converter in STM32F401

- This development board does not support any DAC channels.
- We can use the PWM feature for analog control of external devices.
- Or else we can connect a DAC externally (using resistances or chip).
- Other boards of the family (e.g. STM32F405) supports 12-bit DAC interface.
- Two channels are supported with DMA capability and triangular wave generator.

FREE ONLINE EDUCATION **swayam** India Niye, Jai Niye

The point to note is that for the STM32F01 board that we are using, this board does not have any support for D/A converter, but I told you the board already has PWM capability and using PWM also we can do something which is very much similar to D/A conversion. We can convert the duty cycle into an equivalent average voltage which is like the output of a D/A converter.

There are some other families like where a 12-bit D/A converter is built into the board.

With this we come to the end of this lecture where we had discussed the process of digital to analog conversion. In the next lecture, we shall be starting our discussion on the reverse, analog to digital conversion.

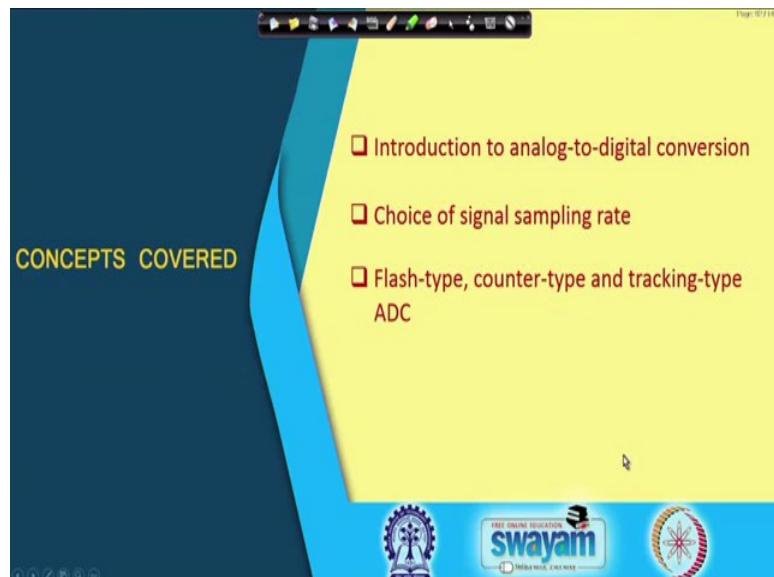
Thank you.

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

Lecture - 13
Analog to Digital Conversion (Part I)

In this lecture, we shall be starting our discussion on Analog to Digital Conversion. If you recall in our previous lecture we discuss the reverse process, how to convert a digital value into an equivalent and proportional analog value.

(Refer Slide Time: 00:42)



In this lecture we shall first be talking about some of the basics of A/D conversion, then there is something called signal sampling rate, we shall be talking about that very briefly, and we shall be looking at some alternate A/D converter designs.

(Refer Slide Time: 01:06)

The slide has a yellow header with the title 'Introduction'. The content is as follows:

- An analog-to-digital converter (ADC) takes an analog input V_A as input, and generates a digital word D as output, where $D \propto V_A$.
- Various types of ADC are possible:
 - a) Flash-type ADC ✓
 - b) Counter-type ADC ✓
 - c) Tracking-type ADC ✓
 - d) Successive approximation ADC ✓
 - e) Dual-slope ADC ✓

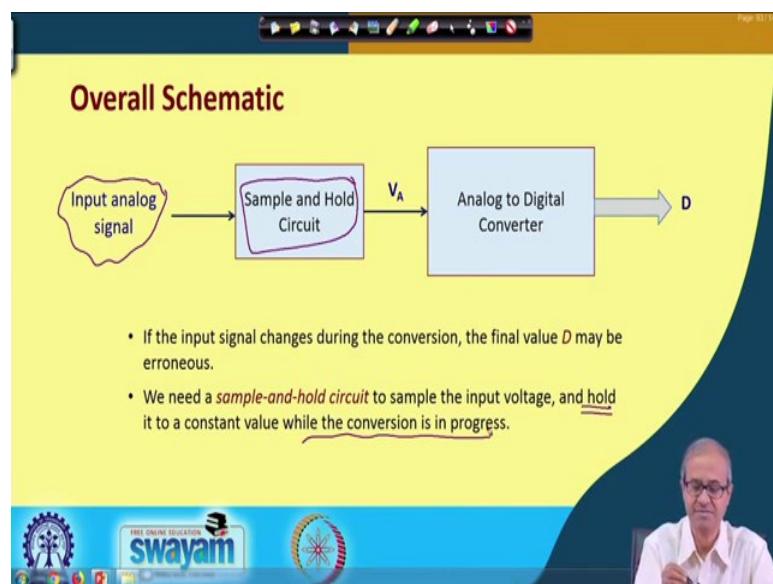
Hand-drawn annotations on the slide include a block diagram of an ADC with an input V_A and an output D , and several checkmarks next to the listed ADC types.

The footer of the slide features the 'swayam' logo and other educational icons.

Let us start with the basic introduction to A/D converter and the various types that are possible. An analog to digital converter in short you call it an A/D converter, it takes an analog voltage V_A as input and produces digital value at the output D , where D is proportional to V_A .

Various types of A/D converter designs are possible, some of them are listed here --- flash, counter, tracking, successive approximation and dual slope. During the lectures we shall be discussing the first four of these, the last one we shall not be discussing. The first four are most commonly used.

(Refer Slide Time: 02:17)



Before we start the discussion on the different types of A/D conversion, let us look at the bigger picture, what we are actually trying to achieve using this A/D conversion. There is some analog signal which is coming from outside, this can be a voice, this can be a temperature waveform. Let us say, in an industrial plant we are trying to implement some control circuitry, where we are monitoring the temperature, pressure, humidity of a oven or some plant.

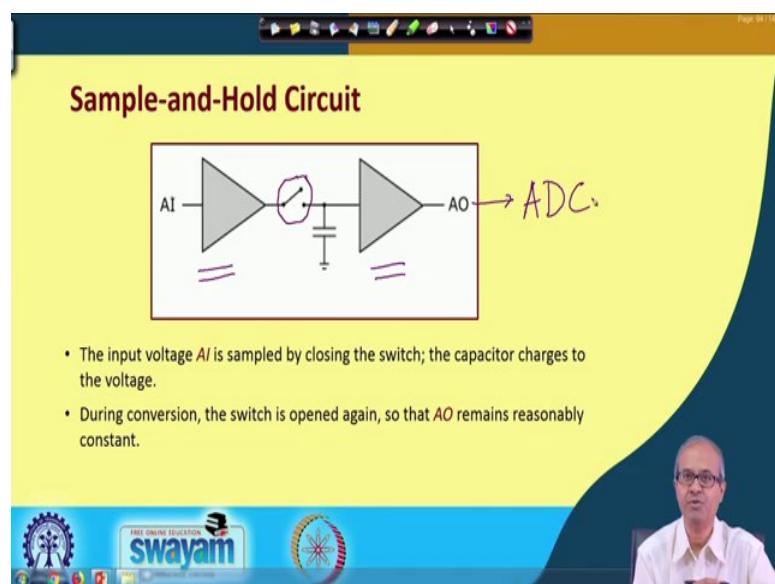
Those variables are being continuously monitored or sensed. It will be like a waveform, changing up and down. It is not necessary that they will remain constant. And here you have the A/D converter which will be converting the voltage that is equivalent to the sensed value into a proportional digital output.

But you see you have another functional block out here, which is mentioned called sample and hold circuit. Why do we need sample and hold circuit? The reason is very simple, suppose my A/D converter takes 5 milliseconds to complete one conversion. My input waveform is coming, I am reading it, and converting it -- the process takes 5 milliseconds.

Now, suppose my input waveform is changing very rapidly, even within the 5 millisecond time the value of the input waveform is changing. So, what will happen? During conversion if the input itself is changing, the final result may become erroneous. I expect that when A/D converter is doing the calculation the input should be held at a

stable value. The sample and hold circuit does just that; it samples the input at a particular time and holds it to that value while A/D conversion is in process. This is the main purpose of sample and hold circuit.

(Refer Slide Time: 05:04)



I am not going into the detail circuit diagram what is that inside the sample hold circuit, but schematically I am showing you what are the basic building blocks. There are two buffers, this is your analog input, this is your analog output, and there is a switch which can be controlled electronically.

Whenever the switch is closed this capacitor will get charge to that voltage. The switch will remain closed for certain time, enough for the capacitor to get charged, then the switch is again opened. Now the capacitor does not have any path to discharge because this is a buffer, the input resistance of this buffer is typically very high. So, the capacitor cannot discharge. The analog output you can feed to your A/D converter. This voltage will be maintained at a stable level while A/D conversion is in progress.

(Refer Slide Time: 06:34)

The slide is titled "How Fast We Must Sample?". It contains a bulleted list of points and a graph of a band-limited signal. The graph shows a horizontal axis with two points labeled f_{\min} and f_{\max} . A wavy line representing the signal is shown between these two points. The text on the slide includes:

- A very important decision.
 - Guided by Nyquist Theorem.
- What is Nyquist Theorem?
 - A band-limited analog signal that has been sampled can be perfectly reconstructed from an infinite sequence of samples if the sampling rate f_s exceeds $2f_{\max}$ samples per second, where f_{\max} is the highest frequency in the original signal.
 - If the analog signal does contain frequency components larger than $f_s/2$, then there will be an aliasing error.
 - Aliasing is when the digital signal appears to have a different frequency than the original analog signal.

The next question is what should be the rate of sampling? Should we sample once every second, once every millisecond, once every microsecond, what should be the best sampling rate. Well sampling rate depends on the characteristic of the input signal and there is a nice theorem called Nyquist theorem that gives us a guideline in this regard. What Nyquist theorem says is that for a band limited analog signal, band limited signal means a signal that has frequency components within a range, let us say f_{\min} to f_{\max} .

Suppose we are sampling it, the signal can be perfectly reconstructed at the receiving end provided we carry out sampling at greater than $2f_{\max}$. If you do that, then at the receiving end you can reconstruct all the frequency components accurately. This is what Nyquist theorem says and this gives you a guideline what should be the required sampling rate. This of course depends on an input signal, what is the value of f_{\max} .

But if you do not satisfy this condition, then your receiving end might get wrong inference regarding the frequency components present in the signal. I am not going into detail, but this is the basic idea on the result of sampling.

(Refer Slide Time: 08:55)

Resolution

- The least significant bit of an analog-to-digital converter (ADC) gives the resolution.
- Related to full scale if the ADC is linear.

• $\text{LSB} = A / 2^n$ for an n -bit ADC.

• For a linear 8-bit ADC with a 1V full scale input,

$$\text{Resolution} = 1.0 / 2^8 = 3.9 \text{ mV}$$
$$\text{Percentage Resolution} = \text{Resolution} / (\text{Full-scale voltage}) \times 100 \%$$
$$\frac{3.9 \times 10^{-3}}{1} \times 100 = 0.39\%$$

There is another thing called resolution of an A/D converter, just like in a D/A converter we had resolution. For A/D converter also we have resolution, this is given by the least significant bit. The minimum change in the analog input which will result in a change in the digital output by +1 is resolution.

If the characteristic of the A/D converter is linear then it changes linearly with changes in the input. The resolution of the LSB will be defined as A , A is the full scale voltage divided by 2^n .

Let us take an example. Suppose I have an 8-bit A/D converter and my full scale voltage is 1 volt. So, the resolution will be $1V / 2^8 = 3.9 \text{ mV}$. This can also be expressed as a percentage by multiplying by 100. So, in this case this will be 0.39 %. In this way you can calculate resolution and percentage resolution.

(Refer Slide Time: 11:18)

The slide has a yellow header and a blue footer. The header contains the title 'Conversion Time and Bandwidth'. The footer features the 'swayam' logo and other educational icons. A video feed of a speaker is visible in the bottom right corner.

- How often can a conversion be done?
 - A few ns to a few ms depending on the technology.
- Input bandwidth
 - Maximum input signal bandwidth
 - Sample and hold input circuitry
 - Conversion frequency.

Conversion time and bandwidth are two other parameters. Conversion time is how much time it takes for the A/D converter to convert a given input analog voltage into the digital equivalent. Depending on the type of converter it can be very fast, it can take few nanoseconds, or it can be quite slow of the order of milliseconds.

And input bandwidth is the maximum frequency of the input that can be supported, it depends on two different components of the circuit. One is how fast is the sample and hold circuit, and the other is how fast is the A/D converter.

(Refer Slide Time: 12:17)

The slide has a yellow header and a blue footer. The header contains the title 'ADC Transfer Characteristics'. The footer features the 'swayam' logo and other educational icons. A video feed of a speaker is visible in the bottom right corner.

• This curve is for ideal ADC.

• Possible errors:

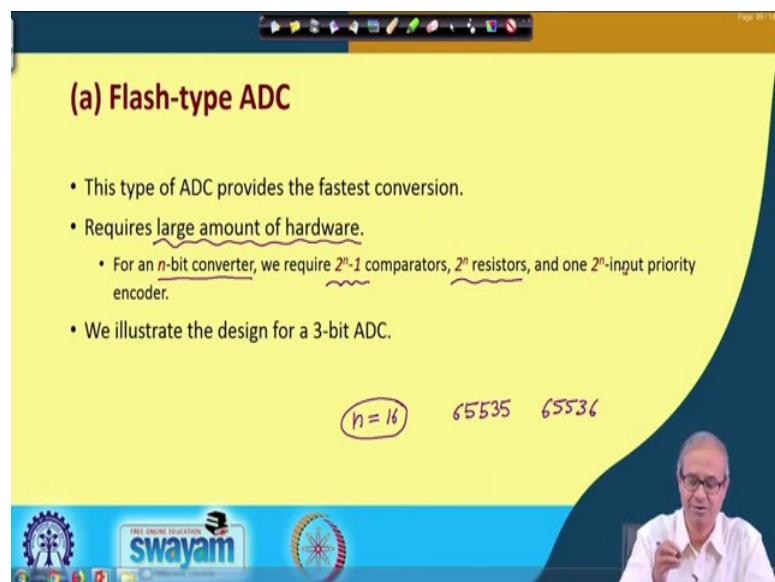
- Offset
- Non-linearity*

The graph shows an ideal ADC transfer characteristic. The x-axis is labeled 'Vin' and ranges from -2 to 12. The y-axis is labeled 'ADC count' and ranges from 0 to 12. A smooth, linear curve is drawn from (0,0) to (10,10). A stepped line represents the ideal digital output, which follows the curve but is quantized at integer values from 0 to 10.

Talking of the ADC transfer characteristics, as the input changes there is also a stepwise increase just like a D/A converter. It goes up like this, this curve is for ideal case because this is a straight line, but in general there can be offset error or nonlinearity error. Well offset error means instead of this you may see that your waveform is either like this or like this.

Nonlinearity means the step height may not all be equal, for one step you may see that it is going big, second step may be small, third step may be big, fourth step may be small, this is called nonlinearity. Such errors may be there in an actual implementation.

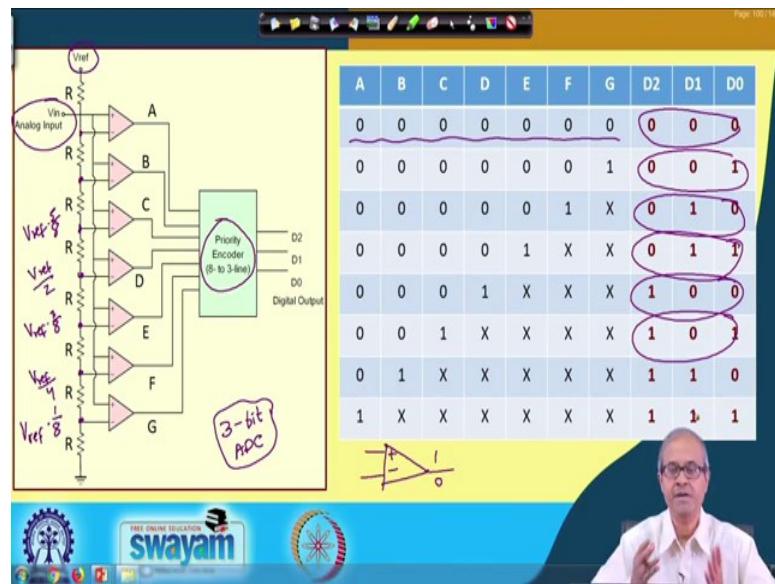
(Refer Slide Time: 13:13)



Now let us come to the different types of A/D converter and how they work. The first type of AD converter is the flash type A/D converter. Well this has nothing to do with flash memory, flash means very fast, this is the fastest type of A/D conversion that is possible. But the drawback of this converter is that it requires enormous amount of hardware, how large? Let us have an estimate, suppose we want to design an n -bit A/D converter.

Let us say $n = 16$. I will be needing 2^n-1 comparators, for $n = 16$, it will be 65535. And we need one huge priority encoder that will be having 65536 inputs, which is impractical to build. Because of this you can only build very small flash A/D converters, for larger values of n it becomes really impractical. Let us see how flash AD converter looks like and how it works.

(Refer Slide Time: 15:09)



Here on the left hand side I am showing the diagram of a 3-bit flash A/D converter. In the first stage there is a reference voltage V_{ref} that is used and there is a chain of resistances to ground. These resistances serve the purpose of providing several reference voltages.

The lowest one has a resistance R below and $7R$ above. So, what will be the voltage here? $V_{ref} \cdot R / 8R$. This will give a voltage of $V_{ref} / 8$. If you move to the next one this will be $2R$ and this is $6R$. So, $2R$ divided by $(2R + 6R)$. So, it is $V_{ref} / 4$. Then you look at this, it will be $V_{ref} \cdot 3 / 8$ and so on.

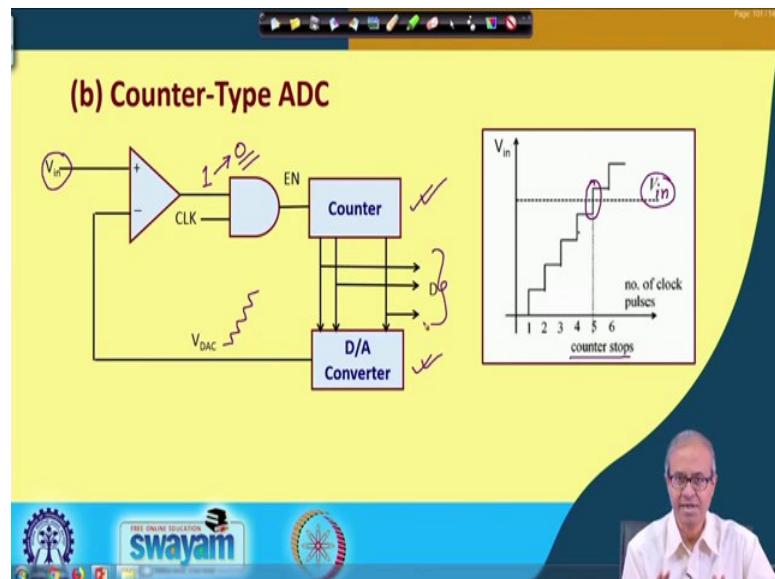
You see it provides a range of a reference voltages equally separated. And there are a series of voltage comparators. Suppose I have a comparator like this + and -, I have two inputs I apply two voltages. If the + input is greater than the - input, then the output will be 1; if - is greater, output will be 0, this is how comparator works. So, depending on the analog input voltage V_{in} , A B C D E F G values will change.

The priority encoder will be converting these binary combinations into equivalent digital output values.

What we get in the output will be the required output digital value, , the delay will be the delay of a comparator plus delay of the priority encoder. There is no iteration or loop

anywhere, that is why this kind of converter is very fast, but as I told you it requires enormous amount of hardware.

(Refer Slide Time: 19:23)



Let us look into something that is more practical to build. You see D/A converter is easier in that respect, you need only resistances. You can built a 10-bit, 12-bit, 16-bit, 32-bit D/A converter without any trouble.

The methods that we talk about now use a D/A converter as our central block, and using that we are trying to realize an A/D converter. This method is called counter type A/D converter. We have a D/A converter out here, the input of the D/A converter is fed from a binary counter. It counts 0, 1, 2, 3, 4 like that and this counter is not counting always, it will be counting only when this clock will be coming.

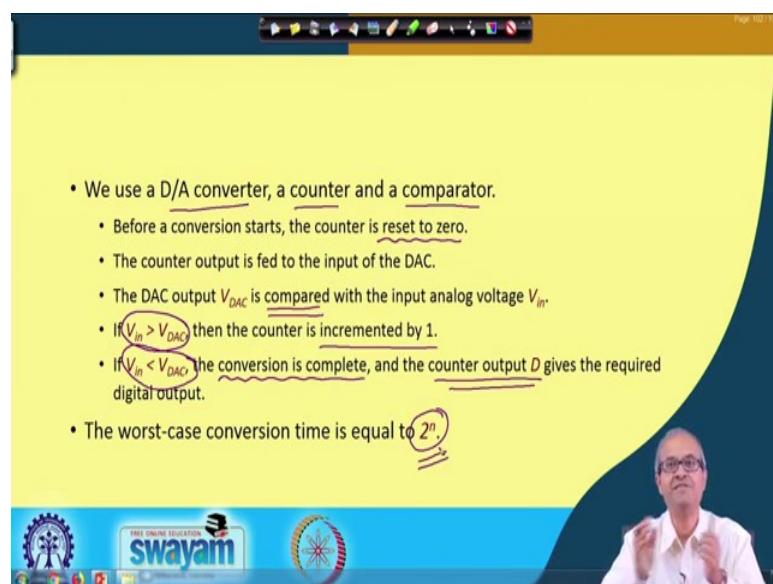
And this clock will be coming provided the other input is at 1, this is an AND gate if the other input is 0 then the clock will not come. As long as the other input is 1, this counter will go on counting. As the counter is counting up you think of this V_{DAC} . What will happen to the output of the D/A converter? It will also increase slowly in steps.

As the counter increases the - input of this comparator will go on increasing, and there will be a point when this - input will be crossing the analog input voltage. As soon as it crosses, this value will become 0 and the counter will stop counting. In this diagram we have showed exactly this thing, suppose this is our V_{in} that you are applied this dotted

line, and the counter will start from 0 it will continuously go on incrementing and as soon as it crosses the counter will stop.

Whatever is the counter value, will be the digital output. This is how you are implementing an A/D converter using a D/A converter. This is called counter type AD converter because you are using a binary counter here.

(Refer Slide Time: 22:12)

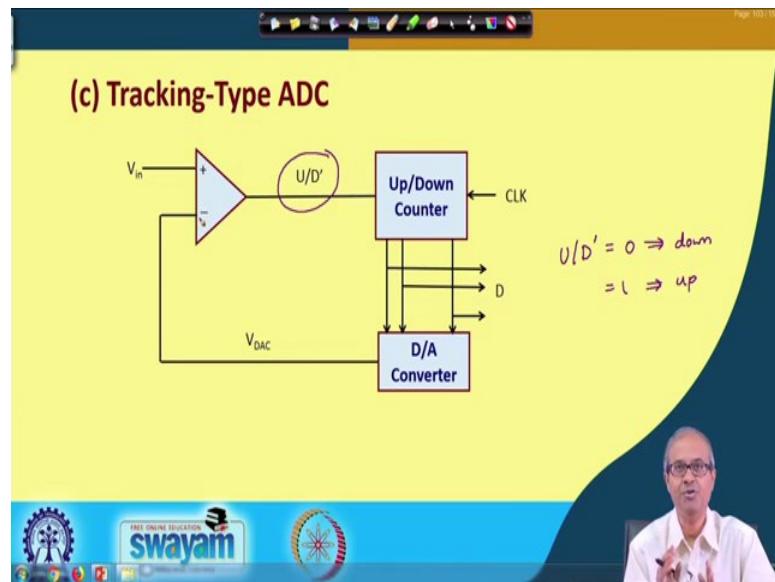


To summarize exactly what we are doing here, the steps are written. You need a D/A converter, you need a counter, and you need a voltage comparator. Before the conversion starts the counter is reset to 0, and as we have seen the output of the counter goes to the input of the D/A converter.

The D/A converter output is fed to the - input of the comparator, and the analog voltage input is connected to the + input of the comparator. So, we are making a comparison, if you find that the input voltage is greater than the DAC output then the counter is incremented by 1; that means, the AND gate is enabled. But as soon as the reverse happens, the output of the DA converter crosses V_{in} , this will mean that the conversion is complete, we stop, and whatever is the counter output will be the required output of the A/D converter.

One problem with this method is that in the worst case for an n-bit counter I may have to count through all 2^n steps to find where V_{in} is. So, the maximum number of clock pulses required maybe 2^n . But it is very simple we can built very large converters.

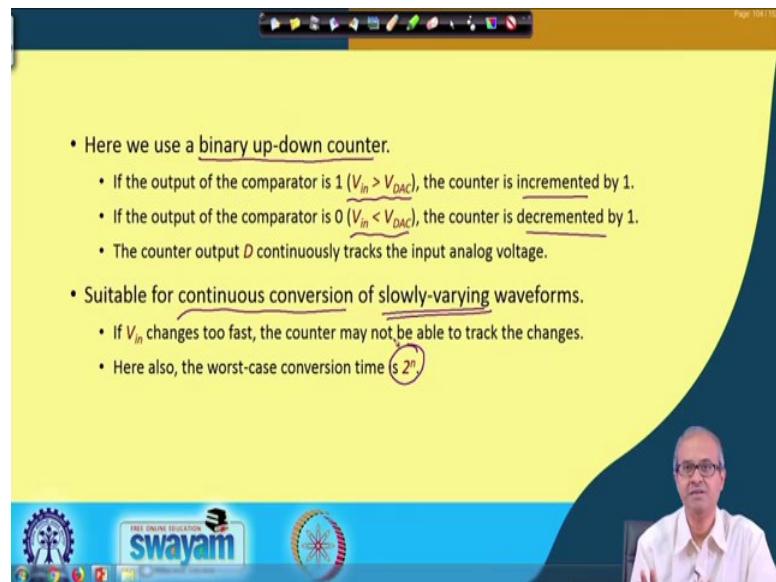
(Refer Slide Time: 24:00)



Now with a small modification you can make an improvement. Here we have something called tracking type A/D converter. You see the circuit looks quite similar, but we have remove that AND gate, and we have replaced the binary up counter by an up/down counter. Up/down counter is a binary counter which can either count up 0, 1, 2, 3, 4 or it can count down 4, 3, 2, 1. And how we decide whether it is going to count up or down? There is a control input called a U/D' , if this input is 0 this means the counter will count down, if it is a 1 this will mean the counter will count up.

Here also the principle is very similar, you start by initializing the counter to 0 and then the comparator will be comparing D/A converter output with V_{in} . If the input is greater than this, the U/D' output will be 1. So, the counter will be counting up, if D/A converter become greater the counter will be counting down. So, the D/A converter output will try to track the input voltage. Here you may assume that there is no sample and hold circuit. The input waveform is also changing slowly and D/A converter output is trying to track it, it is either increasing or decreasing depending on which direction the input is changing and this is happening continuously; this is how tracking AD converter works.

(Refer Slide Time: 26:00)

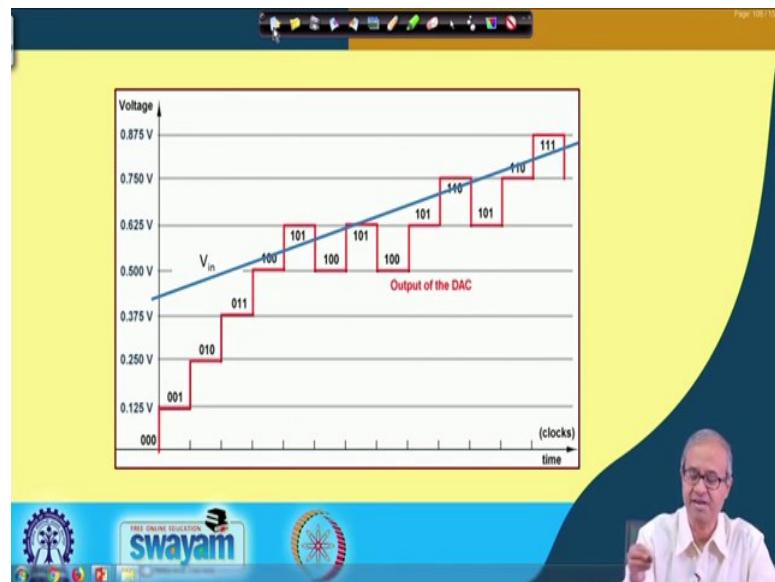


- Here we use a binary up-down counter.
 - If the output of the comparator is 1 ($V_{in} > V_{DAC}$), the counter is incremented by 1.
 - If the output of the comparator is 0 ($V_{in} < V_{DAC}$), the counter is decremented by 1.
 - The counter output D continuously tracks the input analog voltage.
- Suitable for continuous conversion of slowly-varying waveforms.
 - If V_{in} changes too fast, the counter may not be able to track the changes.
 - Here also, the worst-case conversion time is 2^n .

Now here we use a binary up down counter, and if the output of the comparator is 1; that means, V_{in} is greater than V_{DAC} we increment the counter, if it is reverse we decrement the counter. So, here there is nothing like conversion is complete we are continuously doing the conversion we are tracking the input voltage. For applications where the input is changing very slowly and we will need to continuously track the input you can use this kind of AD converter.

This is suitable for continuous conversion of very slowly varying waveform, but the worst case conversion is still 2^n because if the input is very high you will have to count through all 2^n steps to reach V_{in} . The worst case delays complexity you are not improving here.

(Refer Slide Time: 27:09)



To show how the input can track, suppose this blue line I am showing, this blue line let us say is my input waveform.

And this red one is the output of the D/A converter which is trying to track. You see how it changes, it is going up it is out here it is crossing. So, the next step it will be counting down. Next step it is seen that again V_{in} is up again it will go up, again it will go down, again it will go up, again up, down, up, up. And you see the DAC output is trying to track the input waveform as it is changing, this is the main characteristic of the tracking type A/D converter.

With this we come to the end of this lecture. Here we have discussed three different designs of A/D conversion: flash type, counter type and tracking type. In the next lecture we shall be talking about another type of A/D converter, which in fact is faster than the counter type or the tracking type. In the sense that for n -bit conversion you require only n number of clock pulses and which is very much practical; for a 16-bit AD converter you need no more than 16 clock pulses rather than 65000 clock pulses. This we shall be discussing in our next lecture.

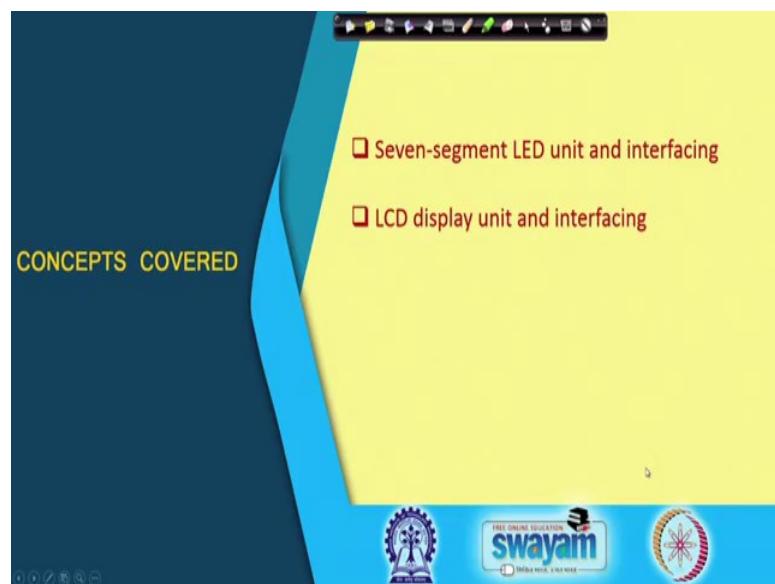
Thank you.

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

Lecture - 14
Analog to Digital Conversion (Part II)

We continue with the discussion on Analog to Digital Conversion.

(Refer Slide Time: 00:27)



In this lecture we shall be first talking about another kind of A/D converter, which is similar to counter type or tracking type converter that we discussed in the last lecture, but is much more efficient in terms of the conversion time. And we shall be talking about what kind of A/D conversion facilities are there in the ARM development board that we shall be using.

(Refer Slide Time: 01:03)

(d) Successive Approximation Type ADC

- The principle of operation is analogous to binary search.
- Suppose we are searching for a number (here V_{in}) in a sorted list.
- We look into the middle of the list – effectively the size of the list reduces by half.
- For 2^n steps, the number of steps required is only n .

128
64
32
16
8
4
2
1

$\log_2 128$

Sorted

SWAYAM

This method of A/D conversion is called successive approximation type A/D converter. The principle of operation is analogous or similar to binary search. For those of you who are familiar with programming have some basic knowledge and understanding of data structure, you may have come across the binary search algorithm.

Let me tell you what binary search is for the sake of those who do not know it. Just assume that I have a list of numbers which are stored in an array, and these numbers are sorted in either ascending or descending order.

Now, I want to search for a number. I want to find out whether a number, let us say 45, is present or not. Well if the array was not sorted, then I would have to search it from the beginning to the end. But because it is sorted I can adapt a very intelligent strategy. I can start with the middle of the array; I compare whether the middle element is less than or greater than the element I want to search. If I find the middle element is less than that, it means my element must be on the right hand side, or if it is other way around, then it must be on the left hand side.

So, you see in one comparison I have reduced the size of the list to half. I repeat the process for the half that I have selected. Suppose it is in the right half. I again probe in the middle, depending on the probe either I have to see in the left half or the right half; I repeat this process.

Suppose I have 128 numbers in the array. After one comparison I reduce the list to half, it becomes 64. After another search, it becomes 32, after another search it becomes 16, like this 8 4 2 and 1. Finally, when there is only 1 element, I can tell that whether it is there or not. So, 7 comparisons are required. What is 7? It is nothing, but $\log_2 128$.

This successive approximation technique is like implementing binary search in hardware in performing the A/D conversion.

(Refer Slide Time: 04:33)

(d) Successive Approximation Type ADC

- The principle of operation is analogous to *binary search*.
 - Suppose we are searching for a number (here V_{in}) in a sorted list.
 - We look into the middle of the list – effectively the size of the list reduces by half.
 - For 2^n steps, the number of steps required is only n .
- What modifications are required?
 - We use a successive approximation register (SAR) instead of a counter.
 - The SAR emulates binary search.
 - For example, in 4 bits, it starts with 1000 (i.e. 8).
 - If the input is smaller, the next value is set as 0100 (i.e. 4).
 - If the input is larger, the next value is set at 1100 (i.e. 12).

$(\log_2 2^n = n)$

You see for 128 it will need $\log_2 128$ comparisons. So, if there are 2^n number of elements, I will be needing $\log_2 2^n = n$. I need n number of steps. So, if I can implement this I need only n clock pulses and not 2^n clock pulses.

(Refer Slide Time: 05:17)

(d) Successive Approximation Type ADC

- The principle of operation is analogous to *binary search*.
 - Suppose we are searching for a number (here V_{in}) in a sorted list.
 - We look into the middle of the list – effectively the size of the list reduces by half.
 - For 2^n steps, the number of steps required is only n .
- What modifications are required?
 - We use a successive approximation register (SAR) instead of a counter.
 - The SAR emulates binary search.
 - For example, in 4 bits, it starts with 1000 (i.e. 8).
 - If the input is smaller, the next value is set as 0100 (i.e. 4).
 - If the input is larger, the next value is set as 1100 (i.e. 12).

$1000(8)$

0100

1100

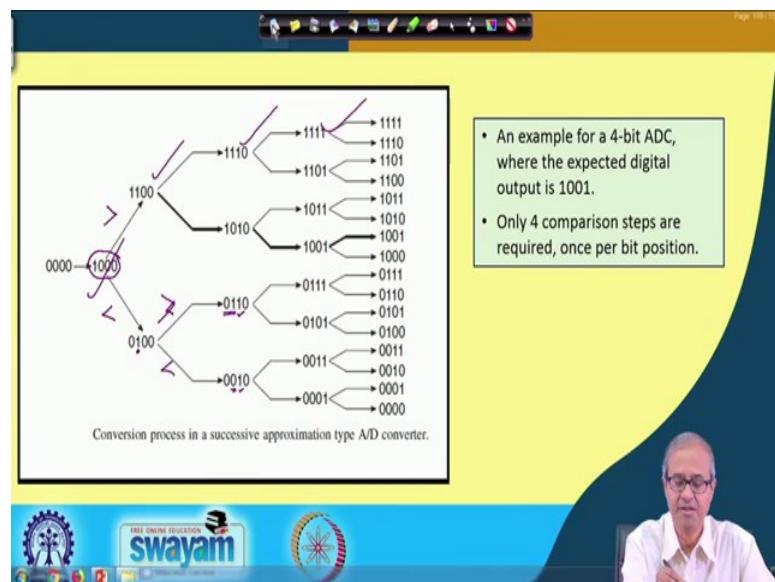
FREE ONLINE EDUCATION **swayam**

We use something called a successive approximation register instead of a counter. Consider a 4-bit A/D converter.

Let us say we initialize this register with 1 0 0 0. This 1 0 0 0 means 8, which is approximately the middle of the range 0 to 15. I start with the middle point, I compare whether it is less or greater. Suppose I say that the input is smaller; then it will be on the left half 0 to 7. What is the middle point of 0 to 7? 4. So, what is 4? 4 is 0 1 0 0. And on the right hand side if it is greater; that means, 9 to 15, middle point of it is 12, 12 is 1 1 0 0.

So, what we are actually doing? We started with 1 0 0 0. If it is less than this 1 is made 0 and the next bit is made 1, but if it is the other way round this 1 remains 1, I do not change, but the second bit only I am making 1. This is what we are doing repeatedly and this emulates binary search. This is the role of the successive approximation register.

(Refer Slide Time: 07:07)



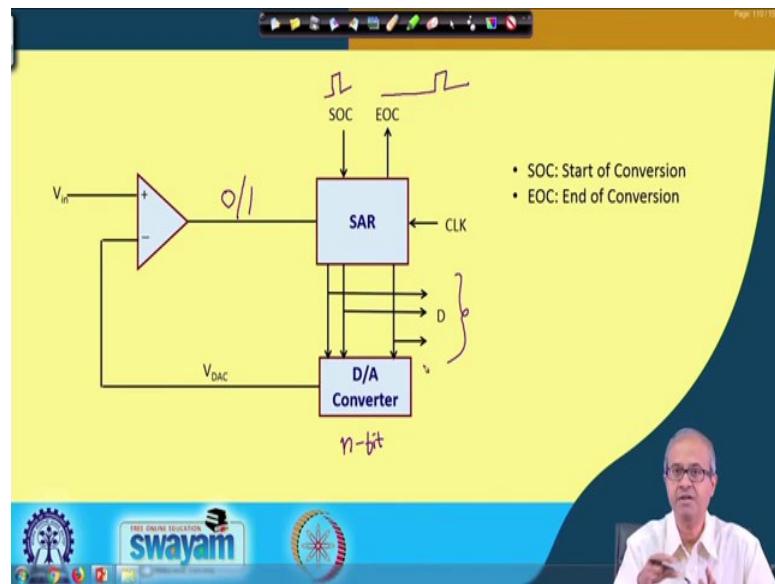
You see this diagram that I am showing. On the left hand side it shows exactly what I just now told with the example. In the SAR we start with 1 0 0 0, we compare with the analog input voltage whether it is less than or greater than.

If the if the input is less than that the SAR output; then I go here. I make this bit as 0, I make the next bit 1: so 0 1 0 0. If it is other way round, I make it 1 1 0 0, and I repeat this process. You see here also I again make a check whether it is less than or greater than. If it is less than, in the last bit which I had made 1, I make it 0 and the next bit I set to 1.

But on the other side if it is greater than, I am not changing, the next bit I am changing to 1, and this process repeats. Always I am looking at the next bit, the immediate bit I am changing it back to 0 and setting the next bit to 1, or I am not disturbing the present bit just changing the next bit to 1.

So, in this way I make 1 comparison, 2 comparison, 3 comparison and 4 comparison and I get finally my result whatever will be my output of the A/D converter. So, here only 4 comparison steps are required.

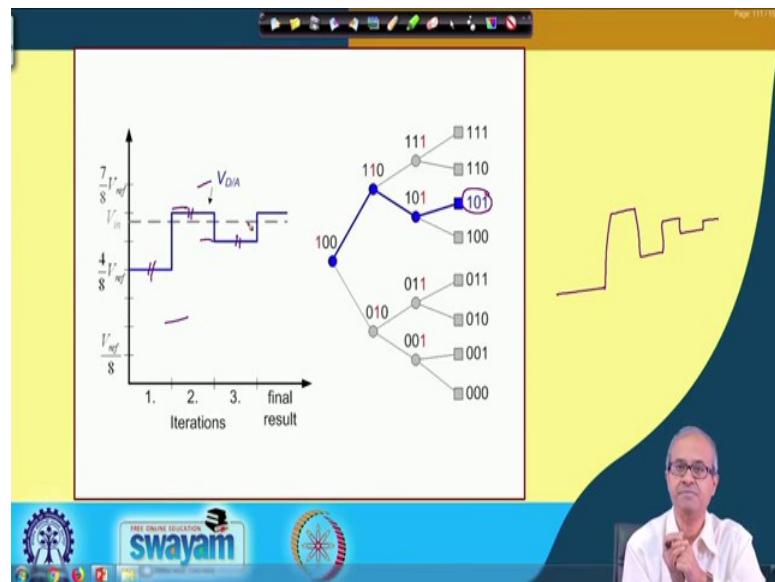
(Refer Slide Time: 08:57)



Block diagram-wise this is how a successive approximation A/D converter looks like. There is still a comparator and a D/A converter, very similar to a tracking type ADC; just the up down counter is replaced by a successive approximation register and this implements actually the logic, which we mentioned depending on whether the output of the comparator is 0 or 1 it takes a decision.

And whenever you want to start the process of conversion, there are typically two interfaces, SOC and EOC. If you apply a pulse in SOC the conversion will start and internally there will be 8 clock pulses. Suppose you are using an n -bit converter. You need a D/A converter of n bits, after n clock pulses this EOC signal will be made high. So, any device outside will know that my A/D conversion is over. Now I can read the value of D . This is how it works. It is simple in concept, but very flexible and very fast; this requires only n number of clock pulses.

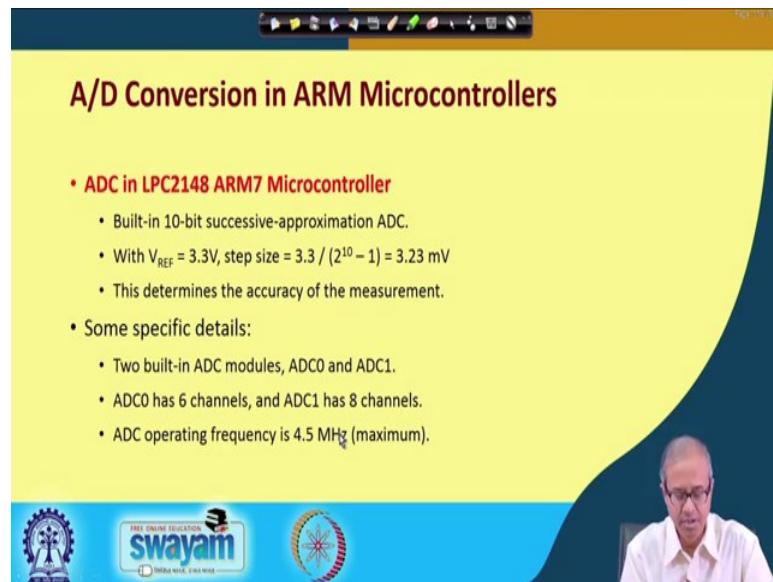
(Refer Slide Time: 10:21)



One small example is shown here for a 3-bit converter. See as this binary search goes on, how the output of the D/A converter changes? You start with the middle. This is my middle point then you either go here or here, let us say I have to go up. So, this is my next point. Next step you either go up here or here, let us say you go here like this. You start with the middle point, then maybe you will be going here, then you will be going here then we will be going here like this; you will be converging to the point.

This is a small example I have shown here, for input that corresponds to 1 0 1. It gets converged to that point, but if you look at the output waveform of the D/A converter, the waveform will look like this.

(Refer Slide Time: 11:39)



A/D Conversion in ARM Microcontrollers

- **ADC in LPC2148 ARM7 Microcontroller**
 - Built-in 10-bit successive-approximation ADC.
 - With $V_{REF} = 3.3V$, step size = $3.3 / (2^{10} - 1) = 3.23 \text{ mV}$
 - This determines the accuracy of the measurement.
- Some specific details:
 - Two built-in ADC modules, ADC0 and ADC1.
 - ADC0 has 6 channels, and ADC1 has 8 channels.
 - ADC operating frequency is 4.5 MHz (maximum).

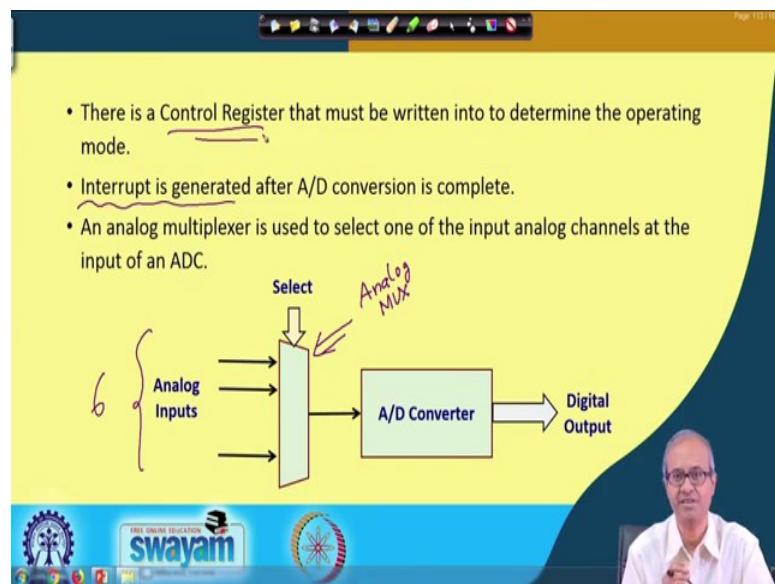
Now, talking about the ARM microcontrollers, what kind of A/D conversion facilities are there? You see D/A conversion is easy, you only need some resistances to make a D/A converter, but A/D conversion is more difficult. You need to have some special circuitry to do A/D conversion.

Let us look at some of the older generations of ARM. ARM7 was one of the previous generation microcontrollers. There was a built in 10-bit successive approximation A/D converter. They all implemented successive approximation technique because it is efficient, because it requires less amount of hardware.

Now, since it is a 10 bit converter, if I connect a 3.3 volt supply to the reference voltage, then the step size will be full scale voltage divided by $(2^{10} - 1)$. If you make a calculation this comes to 3.23 millivolts. This determines to what level of accuracy you can make the measurement. This resolution is a measure of accuracy.

Inside the processor there are two such ADC modules, they are called ADC0 and ADC1. ADC0 has 6 channels and ADC1 had 8 channels. 6 channel means you could connect 6 inputs for conversion; 8 channel means you could connect 8 inputs. And the frequency of the clock was maximum 4.5 MHz.

(Refer Slide Time: 13:57)



• There is a Control Register that must be written into to determine the operating mode.

• Interrupt is generated after A/D conversion is complete.

• An analog multiplexer is used to select one of the input analog channels at the input of an ADC.

6 Analog Inputs → Analog MUX → A/D Converter → Digital Output

Now, let us see how these channels work. Here you have an A/D converter and this block that I am showing is an analog multiplexer. You know what is the digital multiplexer is; if there are multiple inputs, one of the inputs are selected based on the select lines. Analog multiplexer is similar, the difference is that the inputs are analog voltages; one of the input will be selected at the output depending on what you are applying at the digital select input lines.

Now, in ADC0 there are 6 analog input channels. By appropriately selecting it, you can select one of them for conversion. Multiple of them can be used simultaneously, but you will have to switch from one to other at a very fast rate; you convert one channel then convert second channel, then convert third channel. Again come back to the first.

Now, this you can possibly use again because of the Nyquist theorem I talked about. The A/D converter may be working at 1 MHz speed, but you may not be requiring that kind of a sampling rate depending on your input waveform characteristic. May be your sampling rate will be 1 KHz only. So, multiple channels can be multiplexed on the same A/D converter and you can use them simultaneously. Here there are 2 such channels ADC 0 and ADC1 supporting 6 and 8 channels respectively.

After A/D conversion is completed an interrupt signal is generated. This is how it works. And there is a control register that has to be initialized appropriately. This was what was there in ARM7.

(Refer Slide Time: 16:07)

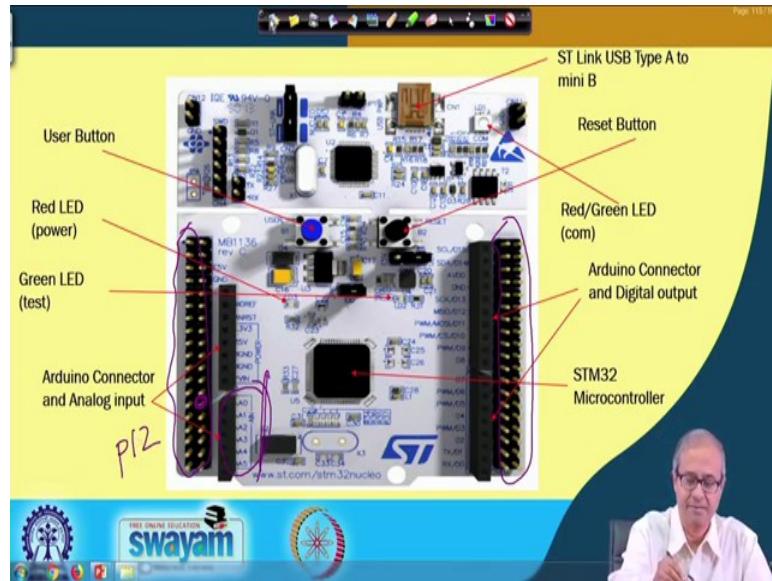
• ADC in ARM Cortex-M4

- STM32F4xx microcontrollers have up to 3 in-built 12-bit A/D converters, each of which has 19 channels.
- With $V_{REF} = 3.3V$, step size = $3.3 / (2^{12} - 1) = 0.806 \text{ mV}$
- There are 16 external channels, connected to I/O pins.
- 6 of these are connected to dedicated Arduino connector pins, A0 to A5.
- There are 3 internal channels:
 - V_{bat} : voltage on battery pin.
 - Temp sensor: used to measure difference in temperature.
 - V_{ref} : voltage reference for the A/D converter.

Now, coming to the board that we would be using, STM32 that is based on the ARM Cortex-M4 microcontroller. In this board there are 3 built-in 12-bit A/D converters and each of them can support up to 19 channels; that means, the analog multiplexer has so many inputs. For 12 bit if your reference is 3.3 volt, your step size will be much smaller. So, your accuracy will be higher, 0.8 millivolts.

There are 16 external channels that are connected to the input/output pins and out of the 16 channels, 6 of them are available on the Arduino connector pins. These are called A0 to A5. And in addition there are 3 internal channels that are meant for checking the health of the system. These 3 internal channels are connected to the voltage of the battery; if the board is running on battery you can read what is the battery voltage, then there is a built in temperature sensor in some of the boards. You can read the temperature of the board also and also you can read, what is the current reference voltage. So, all these internal things you can monitor; these are called internal channels.

(Refer Slide Time: 17:59)



Now, coming back to this board again, you see in this Arduino connector, the six analog input lines are available here, but more number of analog input pins are supported. If you want more you will have to access them from these extension pins. In the STM32 extension pins, you have access to all the signal pins, but here you have access to only the Arduino compatible connector pins.

When you are requiring A/D converter in the experiments that we shall be showing, we shall not be using too many channels. We can directly connect to the Arduino input connectors. In that case those pin numbers we can refer to as A0 A1 A2 A3 A4 A5. We can also give the pin numbers; these will be 1 2 3 4 5 6 7 8 9 10 11 12. We will be calling them as P12 (say).

So, you can either call them by name or you can call them by the pin number. When you write a program, you can use any of those names as you want.

With this we come to the end of this discussion on A/D conversion. Earlier we had looked at D/A conversion. In the next lectures, we shall be talking about some of the common sensors and actuators that are very important in embedded system applications, and many of them we shall be actually demonstrating through the hands-on sessions. Before using them, it is always good to know what the sensors are, how they work and some basic idea as to how they can be interfaced with the microcontroller. These discussions we shall be continuing in our next lectures.

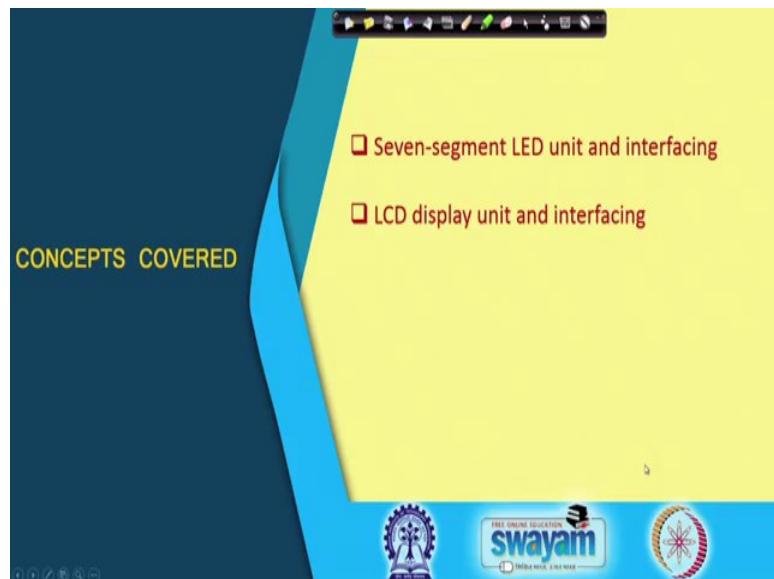
Thank you.

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

Lecture – 15
Output Devices, Sensors and Actuators (Part I)

In this lecture, we shall be starting with a brief discussion on the various output devices, sensors and actuators, which will be required for the experiments and demonstrations that we shall be showing you using the STM32 and Arduino boards.

(Refer Slide Time: 00:45)



In this lecture we shall be talking about 2 different output devices, some of their characteristics. One is the very familiar 7 segment LED display unit that many of you know of. And, the second is the liquid crystal display (LCD) display unit.

(Refer Slide Time: 01:07)

What is 7-Segment LED Display?

- It is a display device consisting of 8 LEDs used for number display purpose.
- Seven line segments and a dot point.
- It can display 0-9 for Decimal and 0-F in Hexadecimal.

A b C D E F

swayam

Let us start with 7 segment LED display unit. What is a 7 segment display unit? Many of you may already be familiar with this kind of device. It is a display device that consists of 8 light emitting diodes, which are arranged in a particular fashion. There are 7 of these LEDs, which are arranged in the form of a 8-like pattern and there is a dot point. These kind of displays are very useful for displaying numeric characters 0 to 9 by suitably activating these individual segments, which are all light emitting diodes or LEDs, which are shaped in the form of a bar. By glowing or a not glowing them selectively, you can display any arbitrary numeric character.

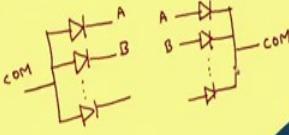
Not only the numeric character 0 to 9, you can also display the hexadecimal characters A, B, C, D, E, F. For example, you can display A like this, you can display b in lower case like this, C can be display like this, d can be displayed again in lower case like this, E can be displayed like this, and F can be displayed like this. So, in addition to 0 to 9, you can also display the hexadecimal characters. Therefore, the 7 segment units are very useful as a hexadecimal display device.

This is how it looks like, there are 7 segments that are numbered A B C D E F G and there is a dot point. A typical display unit looks like this; there are 10 pins, 5 on this side, 5 on the other side, and these 10 pins have connections to all the individual segments and also there is a common terminal, which is available on both the sides.

(Refer Slide Time: 03:27)

Two Types of 7-segment Display Units

- Common Anode
 - The common point (COM) connects to a positive voltage source, and a segment pin should connect with ground to glow the LED.
- Common Cathode
 - The common point (COM) connects to ground, and a segment pin should connect with positive voltage source to glow the LED.



SWAYAM

Now, depending on how the LEDs are connected inside, these 7 segment display units can be of 2 types, either common anode or common cathode. Common anode means the anode terminals are all connected together and this is your common point.

But if it is the other way around means you have the LEDs like this and if the cathode terminals are connected together and this is the common terminal, you call this a common cathode kind of display. So, depending on how they are connected internally, you can have 2 different kinds of display units, common anode or common cathode.

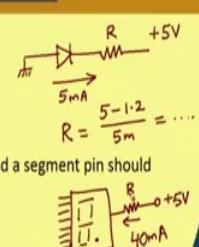
(Refer Slide Time: 04:47)

Two Types of 7-segment Display Units

- Common Anode
 - The common point (COM) connects to a positive voltage source, and a segment pin should connect with ground to glow the LED.
- Common Cathode
 - The common point (COM) connects to ground, and a segment pin should connect with positive voltage source to glow the LED.

Point to note:

- To glow a LED, about 5-10 mA current should be made to flow.
- Current limiting resistance is used (1 for common, or individual for segments).
- Drop across LED is typically 1.2 V.



SWAYAM

Now the point to note is that, when you are actually displaying a 7 segment display unit about 5 to 10 milliampere current is required to be passed through a LED for it to glow at full intensity; this is the minimum amount of current you have to pass. And typically, we use some current limiting resistance, which will limit the total flow of current through an individual LED. The other point to note is that, when an LED is glowing it is forward biased, the voltage across it is typically 1.2 volt; for some LED, it can be even more.

If we look at one individual LED, what we mean to say is that you need to have a current limiting resistance connected in series to it. And let us say on one side I connect 5 volts, on the other side I connect ground, and I want a current of, let us say, 5mA to pass through it. Then the value of this resistance can be very easily calculated, as $(5V - 1.2V) / 5mA$. This will give you the value of the resistance.

Sometimes when you have a 7-segment display unit, there are 8 LEDs inside. The best thing will be to connect a one resistance to each of these individual terminals, so that the current through each of the segments can be fixed at 5mA, but for the sake of interfacing to make it simple, instead of using 8 resistances what you can do is you use a small resistance and let us say for common anode, you can connect it to 5V and whichever segment you want to glow you connect it to ground.

Then this resistance will determine the total current that is flowing. In the worst case, when all the 8 segments are glowing this current will be divided among these 8 segments. So, in total you should allow 40mA of current to flow through this resistance, accordingly you should choose the value of the resistance. Like this you can select the value of the resistance to be used.

(Refer Slide Time: 07:49)

A Typical Interface

- Consider a **common anode** display unit.
- If a segment line is set to GND, it will glow.
- To display **0** we apply $A = B = C = D = E = F = \text{GND}$, and $G = \text{Vcc}$.
- To display **5** we apply, $A = C = D = F = G = \text{GND}$, and $B = E = \text{Vcc}$.
- To display **F** we apply $A = E = F = G = \text{GND}$, and $B = C = D = \text{Vcc}$.
- To display arbitrary characters, the seven segment lines can be driven from microcontroller port lines.
- For **common cathode** display, the convention will be just the reverse.
- GND for OFF, Vcc for ON.

When you have a display unit like this let us assume that I have a common anode display, common terminal is connected to 5V. To glow a segment the corresponding segment pin has to be connected to ground.

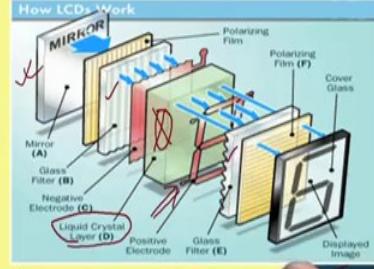
Let us say if I want to display the character 0. So, what is 0? 0 will look like this. All the segments other than G will be glowing. So, only G will be connected to Vcc and all other segments will be connected to ground. Similarly, let us say when you want to display 5, the character 5 is like this. So, the segments B and E will not be glowing. So, B and E you connect to Vcc, and all others to ground.

Similarly, for the character F B, C and D are not glowing. So, BCD you are connecting to Vcc like this. If it is common cathode just the convention will be reversed.

(Refer Slide Time: 09:55)

What is Liquid Crystal Display (LCD)?

- It is a type of display screen used in numerous applications.
 - Very low power consumption as compared to LED.
- LCD units are very thin and is composed of several layers.
 - Two polarized panels, with a liquid crystal solution sandwiched between them.
 - Light is projected through the layers and is colorized as it passes, thereby producing the visual image.



How LCDs Work

MIRROR

Polarizing Film

Polarizing Film (F)

Cover Glass

Mirror (A)

Glass Filter (B)

Negative Electrode (C)

Liquid Crystal Layer (D)

Positive Electrode

Glass Filter (E)

Displayed Image

swayam

So, this is how you can interface a 7 segment LED display unit to the microcontroller. I have told you the basic way of interfacing. Now let us come to liquid crystal display unit, which has become very popular today. In many display devices, we see LCD display units. The LCD display units are used as the display screen in numerous applications starting from many of our mobile phones, our laptop screens, touch screens, many control appliances everywhere.

Some of the older kind of LCD screens cannot display color; those are called monochrome screens, they can display only in black and white. The main advantage of this LCD screen is that they consume very low power as compared to LEDs. I told you to glow every LED, you need of the order of 5 mA of current, but in LCD the current requirement is of the order of microamperes. And the way LCD units are manufactured they are also very thin in terms of form factor.

I am not going into the details of how LCDs are constructed; just like to tell you that LCD has a number of layers inside it, they are all sandwiched together. The central thing is a layer, which is shown here in the middle, this is a liquid crystal layer. There is a special material that is called liquid crystal; depending on a voltage applied, it either allows light to pass through or it blocks light. It is typically polarized and there are some glasses on both sides, this is a glass layer, there is a glass layer, the liquid crystal is sandwiched between them. And let us assume that it is a 7 segment display unit of LCD,

there will be an electrode, which will be here which will be in the shape of a 7 segment display, these individual electrodes can be applied a current and activated.

Depending on the applied current, there will be some polarization of the light that will be flowing through it and depending on the polarization some light will be blocked and some light will pass through; there is a mirror in the backside, which reflects the light so that it is visible. Also for good intensity there is often a LED backplane, where there is a some kind of illumination in the back side so that the display becomes very clear.

So here I mentioned it very briefly. It consists of several layers, there are 2 polarized panels with a liquid crystal solution sandwiched between them as I have shown. And light is projected, when light flows through the layers for a color LCD, they are polarized in some way, which represent colors. It is a very sophisticated kind of engineering that is done so that light comes out with particular color properties.

(Refer Slide Time: 14:05)

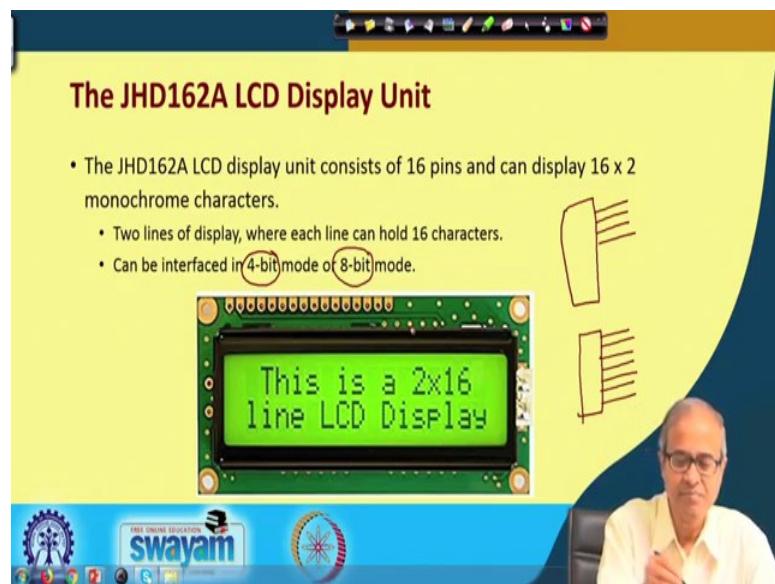
Working Principle

- When an electric current is applied to the liquid crystal molecules, they untwist.
 - Results in a change in the angle of light.
 - The polarized layers are responsible for either blocking the light or allowing it to pass.
- A reflecting mirror is arranged at the back of the unit.
- An electrode plane is used to allow the current to flow in selected areas.
- The entire arrangement is placed inside a sealed casing, with necessary electronics for controlling the device.

When some electric current is passed through one of the electrodes, this electric current will influence the liquid crystal molecules. They will change their orientation, and depending on the change in orientation, the polarized light will either flow or not flow through them. This will result in a change in angle of light. So, you can either block the beam of light passing through a particular segment or you can allow it to pass, in this way a segment is either glowing or not glowing.

So, you are not directly passing a current through the liquid crystal, indirectly it is happening and as I said there is a reflecting mirror at the end, electrode plane and the whole arrangement is placed inside a sealed casing. When you see an LCD display unit, you see everything in a nicely packaged case, very thin, all these layers are engineered and put inside that sleek frame.

(Refer Slide Time: 15:19)



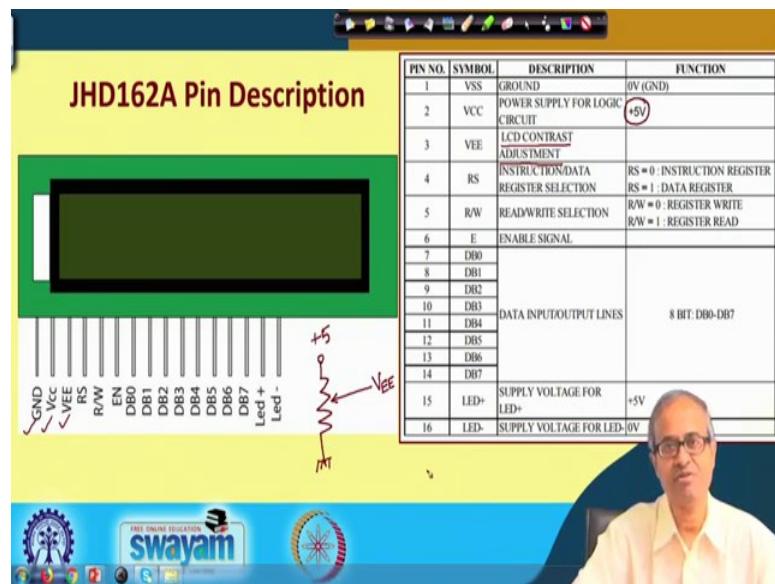
Let us consider one particular LCD device that we shall be showing as part of our demonstration, this is a very convenient output device, which can be used to display alphanumeric characters. Alphanumeric means not only the numbers, but also alphabetic characters abcdefgh everything. The part number is JHD162A and this is the picture of it, you can see on one side there are some connector pins, a total of 16 pins are provided and this display unit can display 2 lines of characters, 16 characters each.

It is a 16 x 2 display unit, and there are 2 different modes. You can either have a 4 bit mode or you can have an 8 bit mode. In 4 bit mode, you connect only 4 data lines with the microcontroller, and in the 8 bit mode, you will be connecting all 8 data lines to the microcontroller. Here, among the 16 pins, 8 of them are data lines. So, if you connect all 8 of them to the microcontroller, it is easier to send the data, because all characters are typically encoded in some character code like ASCII, they are 7 bit or 8 bit.

On the flip side you have to connect 8 wires. So, the number of wires becomes more that is why this device also provides with an alternative of 4 bit mode, where you connect

only 4 bits of data lines to the microcontroller, and the microcontroller will be sending the 8 bits of data in 2 cycles, first 4 bits and then the other 4 bits. If you are using a driver that supports 4 bit mode, you need not have to worry about it, you connect to only 4 lines and the driver will take care of the rest.

(Refer Slide Time: 17:53)



Let us look at the pin description; there are 16 pins. On the right the pins and their brief descriptions are mentioned. The first 2 are ground and power supply, typically the ground is connected to 0V and power supply is connected to 5V power supply. The third input VEE, this you can use for contrast adjustment, by applying a voltage you can adjust the contrast level of the display.

The way you connect is like this, you use a variable resistance called a potentiometer. On one side you can connect 5V, on the other side you connect ground, and this is a potentiometer, this tapping point can be changed either by rotating a knob or there is a screw by rotating a screw, and the middle point you connect to VEE. By suitably adjusting the potentiometer, you can apply any voltage between 0 and 5V to VEE and depending on your viewing angle, you can adjust the contrast accordingly.

There is a pin called RS, RS stands for register select. When you are sending some data or instruction to the display unit, it can be either the data you want to display or you see it is not only a data sometimes, you are telling the display unit that look we want to display the characters in 8 bit mode or 16 bit mode. So, some instructions can also be

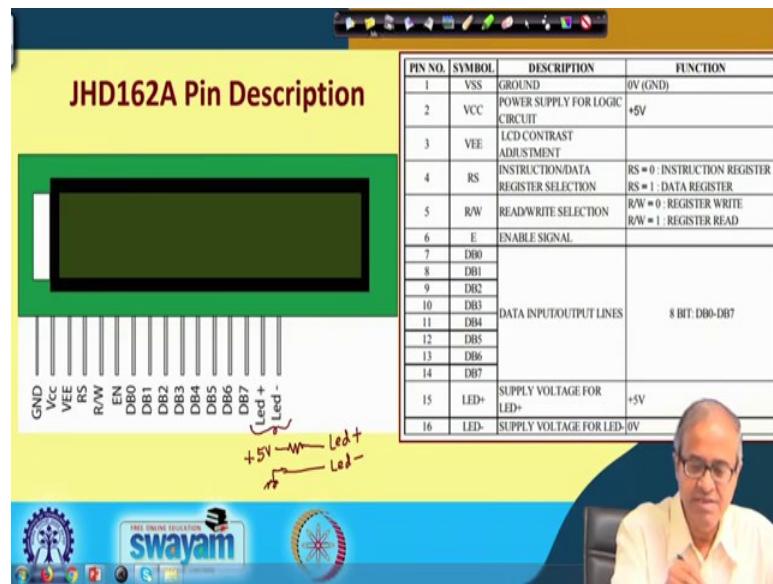
written to this display unit. It can be either instruction or data; this RS actually tells you what you are actually writing.

When you are interfacing with the microcontroller, sometimes microcontroller will be sending an instruction; sometimes it will be sending a data. This RS pin has to be connected to some data line of the microcontroller so that it can change it to 0 or 1 as per the requirement. The next pin is a read/write pin, this tells you whether you are writing a data to the LCD unit or you are reading some status information from the LCD unit. If it is 0, it means write, if it is 1 it means read.

In our interfacing application, we would only be writing, we would not be reading the status. We can directly connect it to ground you can make it 0 so that, it is always in the register write mode. The next pin is an enable signal, you see this enable signal will activate this display unit. Whenever you are enabling it only then it will be working. Typically this enable pin is also connected to the microcontroller, so that whenever it wants to send the data, it enables it and then it writes the data.

If it is 1, it is enabled, if it is 0, it is not enabled. Then come the 8 data lines DB0 to DB7. When you are interfacing in the 8 bit mode, you will be connecting 8 lines from the microcontroller port to these 8 pins, but when you are using the 4 bit mode then you need to connect only 4 of these lines D4, DB5, DB6 and DB7, you only connect the high order 4 lines to the microcontroller; and as I said the microcontroller will be outputting the data in 2 cycles, 4 bits at a time.

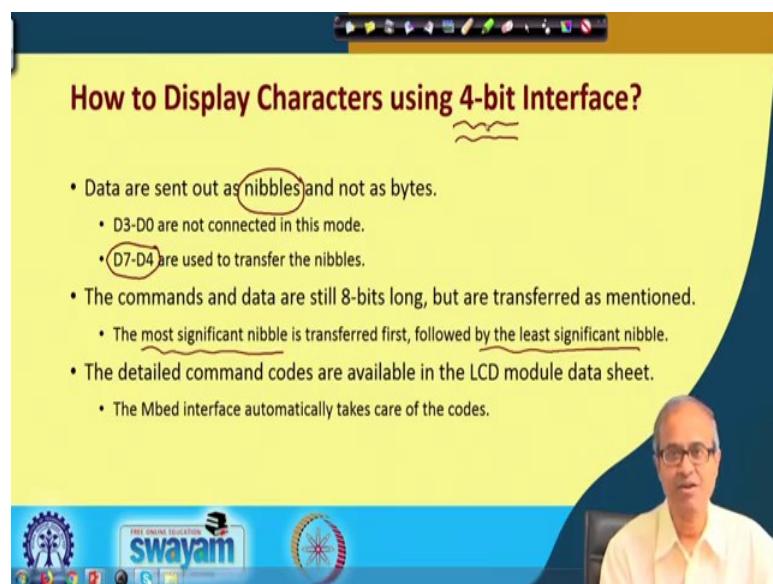
(Refer Slide Time: 22:33)



And the last 2 pins this LED+ and LED-, these are used to control the brightness level. You can connect a voltage across LED+ and LED- to make the display bright, typically what you do with 5V, you can connect a resistance, you can connect it to the LED+ input and the other input you are connecting to LED-, this you can ground, this is how you can connect.

This tells you about the different pins and exactly how you can connect them.

(Refer Slide Time: 23:31)

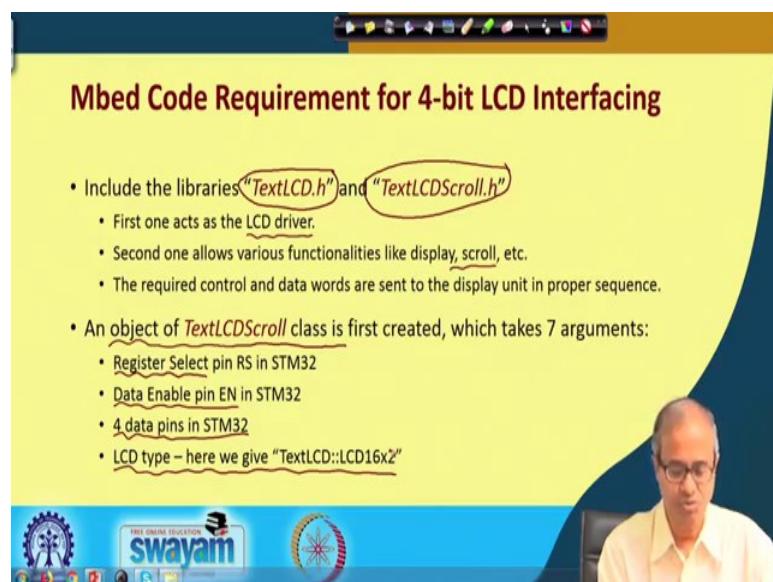


Now talking about the interfacing issue; in our experiments because of simplicity, we shall be using the 4 bit mode, because only 4 data lines have to be connected. Now, I told you in the 4 bit mode data are written into the display device 4 bits at a time, 4 bit is called a nibble. So, data are sent as nibbles.

DB7 to DB4 are connected to the microcontroller, but the commands and data that are actually to be written are still 8 bit wide, and so they will be written in 2 cycles. First the higher order 4 bits are sent, then the lower order 4 bits are sent. With the STM32 board, we will be using a programming environment called mbed.

The mbed programming environment allows us to develop our applications in C, and it provides with a host of library functions and drivers, which you can use. In that library function, this 4 bit interfacing will be taken care of and while writing the program, we need not have to worry about anything, the driver will automatically take care of; that means, writing the high order 4 bit, then low order 4 bit next and so on and so forth, when to write instruction, when to write data those will be taken care of by the driver.

(Refer Slide Time: 25:31)



We shall be using these 2 libraries TextLCD and TextLCDScroll. The first library is actually the LCD driver, it is responsible for sending the low level data and the instructions 4 bits at a time, and the second one allows you with scrolling features in the display, if you are trying to display more than 16 characters in a row, the display will automatically scroll from left to right.

I shall show an example; we first create an object of this TextLCDScroll class, it will create an object that will indicate the LCD unit, this is an object oriented programming concept. Now when you specify this object, 7 arguments have to be specified, because you see with this LCD you have to make some connections with the microcontroller, these 7 things are required to make the connections. What are these things?

First you have to connect the register select input. I told you have to select whether it is the instruction or a data. Then data enable, you have to enable the display unit. Then the 4 data lines, this makes it 6, and lastly LCD type here, we have to specify what kind of LCD display unit is there. You see there are other display units, which is 16 by 4, there are 4 lines, but the display unit that we are interfacing here consists of only 2 lines.

(Refer Slide Time: 27:47)

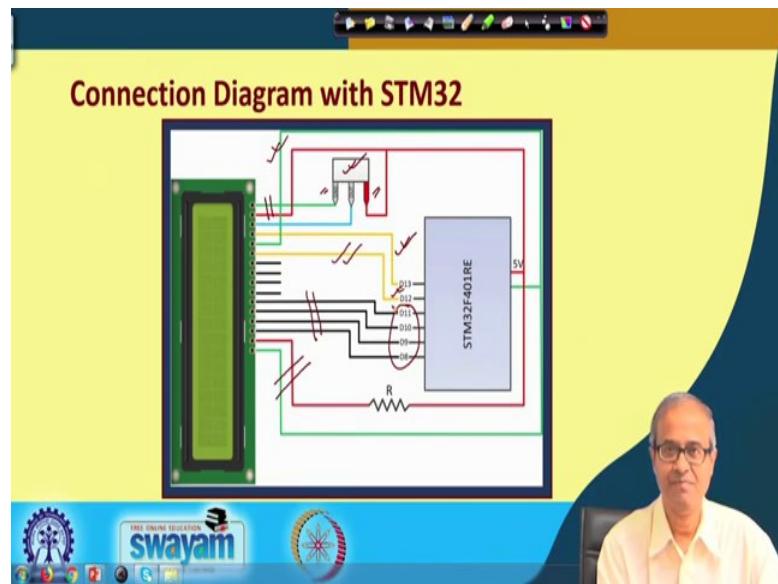
Basic Functions in the TextLCDScroll Class

- **cls()**
 - Used to clear the screen.
- **setSpeed (n)**, where $n \geq 1$
 - Used to set the scrolling speed of the text (text scroll per second).
- **setLine (line_number, "Text")**
 - Here line_number can be 0 or 1, and indicates first and second line respectively.
 - If the number of characters displayed in each line exceeds 16, then the will scroll through the screen.

The different functions that are supported by this TextLCDScroll class are shown. `cls` means clear the screen, it is made blank. `setSpeed(n)`, where n is an integer greater than or equal to 1, means when you are scrolling the screen, what will be the speed of scrolling? 1 means 1 character per second, 2 means 2 characters per second, 3 means first of 3 characters per second, you can control the speed of scrolling.

And `setLine` is a function, where you can tell what text you want to display and in which line number. Line number can be either 0 or 1, 0 means the first line, 1 means the second line. And as I said earlier, if the text you are trying to display on a line is greater than 16 then automatically the text will start to scroll.

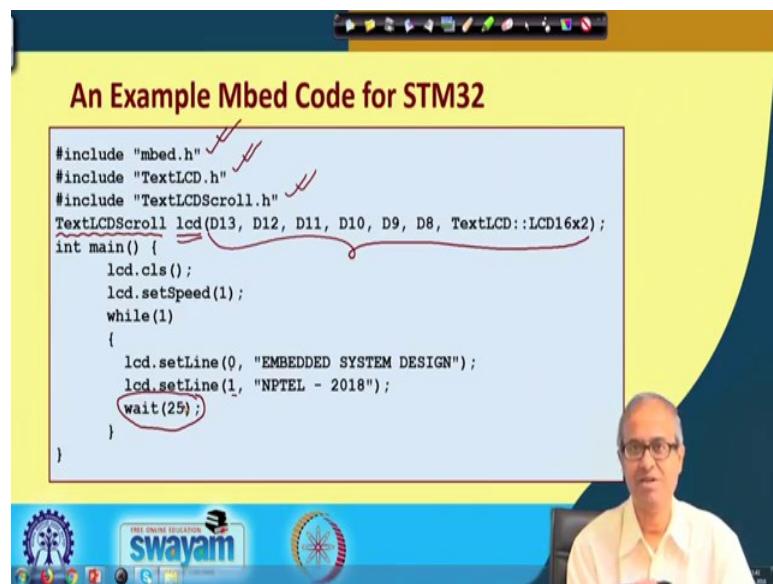
(Refer Slide Time: 28:53)



Here I am showing a very typical interface with the STM32 microcontroller board. You see here I am connecting the LED+ and LED- connections. Here, the first two are the power supply and ground. You see here, I am using a potentiometer on one side, I am connecting ground on one side, I am connecting 5 volt the middle point, I am connecting to the VEE terminal that is the contrast control.

Then, we have the registers select this yellow line, which is connected to one of the microcontroller pins, then the read/write is permanently connected to ground, because we are only writing, then here we have the enable. Enable is also connected to one of the microcontroller lines, because microcontroller has to enable it. Then the high order 4 lines are connected to some microcontroller pins. This is how you make the connection microcontroller to the display unit.

(Refer Slide Time: 30:11)



And I am showing a sample mbed code to display some characters. You see at the beginning you have to include some headers, because you are using the bed programming environment, mbed.h is mandatory. You have to include this and these are the two additional libraries, you are using TextLCD and TextLCDScroll. As I told, you have to create an object of type TextLCDScroll, you give a name lcd, and these are the 7 things you specify.

Here you specify which pins you are connecting to. First one is the register select; that means, you are telling you are connecting register select to your STM32 D13 pin. Then next one is your enable, you are connecting it to the STM32 D12 pin. Then the 4 data lines, you are connecting to D11, D10, D9, D8 and then the last one says what kind of display unit is this? And in the main program, you see it is very simple, we are calling this cls function first.

We have to give the name of this object lcd dot name of the function this is how you call. So, the display is cleared then I am setting speed of scrolling to one character per second. Then in a loop, while(1) means it is an infinite loop. We are displaying something on first line, something on second line, we are displaying EMBEDDED SYSTEM DESIGN, which is more than 16 character. So, there will there will be slow scrolling, 1 character per second, second line NPTEL - 2018, this is fixed, less than 16 this will not scroll.

And here, this is not required actually, I am waiting for 25 seconds. This is how you can interface and program an LCD by connecting with the STM32 microcontroller board. This will give you an idea, how to do it. With this we come to the end of this lecture. We shall be continuing by talking about some other sensors and actuators, and output devices in the next lectures.

Thank you.

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

Lecture – 16
Output Devices, Sensors and Actuators (Part II)

We shall be talking about some of the sensors that we can use for interfacing and for building some embedded system applications in this lecture.

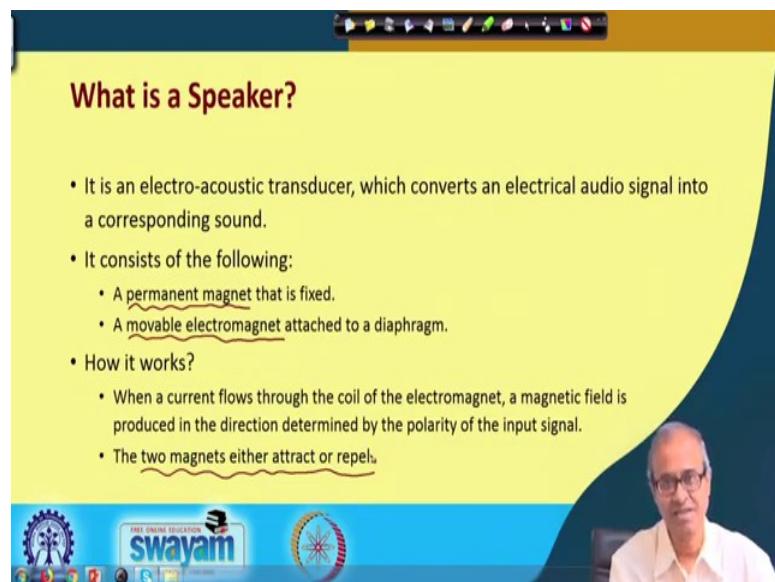
(Refer Slide Time: 00:41)



Here we shall be very briefly talking about some of the sensors like speaker, temperature sensor, LM35, light dependent resistor, microphone that we shall be showing as part of the demonstration. We shall not be going into detail, but very brief working principle so that you actually understand how the thing is working .

First let us talk about a speaker.

(Refer Slide Time: 01:13)



What is a Speaker?

- It is an electro-acoustic transducer, which converts an electrical audio signal into a corresponding sound.
- It consists of the following:
 - A permanent magnet that is fixed.
 - A movable electromagnet attached to a diaphragm.
- How it works?
 - When a current flows through the coil of the electromagnet, a magnetic field is produced in the direction determined by the polarity of the input signal.
 - The two magnets either attract or repel.

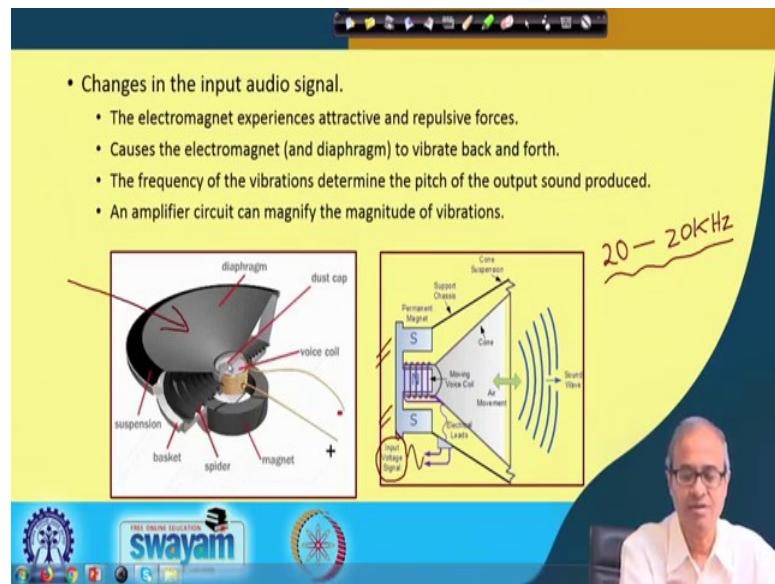
A speaker is a device through which we can generate some sound.

A speaker essentially an electro-acoustic transducer, which means it converts an electrical audio signal into a corresponding sound; that means, you give an electrical signal waveform as the input and the speaker will generate a sound as the output. Now, what is there inside a speaker? Inside a speaker there is a permanent magnet and there is a movable electromagnet.

What is an electromagnet? Electromagnet is constructed out of some ferromagnetic material with a coil around it; whenever there is a current flowing through the coil that material becomes a magnet. The idea is like this. There is a permanent magnet and the coil around the ferromagnet is movable, whenever there is a current flowing this will become a magnet. So, it will either be attracted towards the permanent magnet or it will be repelled from the permanent magnet depending on the kind of current we are passing, and there will be a vibration this is the basic idea.

When current flows through the coil of the electromagnet due to the magnetic field produced this material becomes a magnet and the two magnets either attract each other or there will be a repulsive force there will move away from each other. This will happen depending on the polarity of the current, which direction the current is flowing.

(Refer Slide Time: 03:17)



Just looking into the diagram you see here the permanent magnet is shown in the back this is the permanent magnet. And in the coil, coil is wound around a middle point this can be connected or attached to the permanent magnet, or this can be a separate movable part also, there can in two different kind of speakers you can manufacture.

And when there is a current flowing depending on the input voltage signal there will be some current flowing in the coil. So, this coil will be moving forward and backward depending on the attractive or repulsive force and there is a thin diaphragm connected to that coil. In a speaker you must have seen there is a diaphragm like this on top -- a thin soft material that also starts moving back and forward, and there is a vibration.

Now, if that vibration is in a frequency range that our ear can hear we will be hearing a sound. Now, you know that the typical audio range is 20 Hz to 20 KHz, human ear is able to hear between this frequency range. If there is a frequency with which the diaphragm is vibrating in this range, we will be able to hear that sound; this is how it works.

The electromagnet as well as the diaphragm will be moving up and down, there will be a vibration and the frequency of vibration will generate a sound. In most audio systems we also have a good amplifier circuit with very low noise so that we get a very clear and nice sound.

(Refer Slide Time: 05:45)

What is LM35?

- The LM35 series are precision integrated-circuit temperature sensors with an output voltage linearly proportional to the centigrade temperature.
- Does not require any external calibration and provides accuracy of $\pm 0.25^\circ\text{C}$ at room temperature.
- It is a low-power device, and draws only $60\mu\text{A}$ current from the power supply.
- It can operate over the range -55°C to 150°C .
- Operates for supply voltages from 4V to 20V.

LM35

1 4-20V
2 OUT
3 GND

The analog output increases by 10mV for every degree rise in temperature

swayam

A photograph of a man with glasses and a white shirt, identified as the speaker, is visible on the right side of the slide.

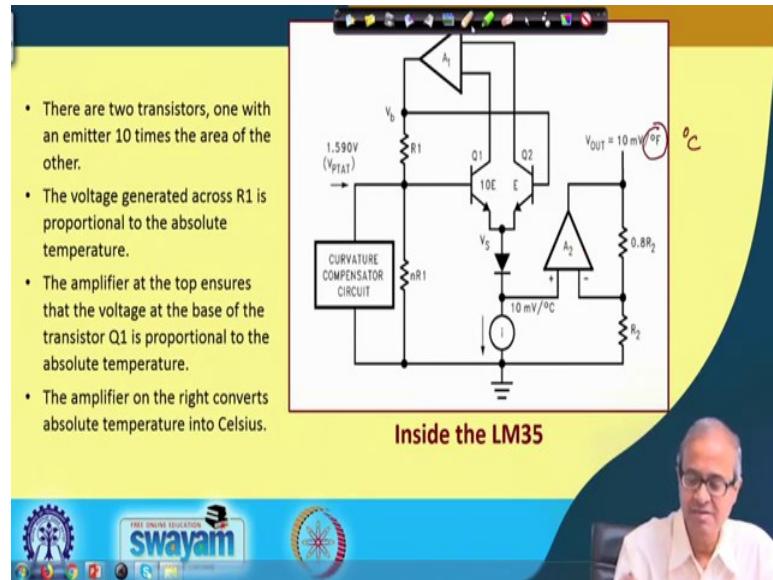
Next let us talk about one kind of temperature sensor. This is a very popular temperature sensor that many of us use in our experiments; this is called LM35. This LM35 is a very convenient kind of a temperature sensor; it looks like this. There are 3 pins; pin number 1 and pin number 3 are the power supply pins. Pin number 3 you connect to ground and pin number 1 you can connect a positive power supply; depending on the type number of LM35 this power supply voltage can vary between 4 and 20 volts.

And the middle pin number 2 will be generating some analog voltage that will be proportional to the external ambient temperature. There are two versions of LM35 available, some of them can directly generate a voltage proportional to the temperature in degree centigrade or Celsius, there is another version that can also give in degree Fahrenheit.

As I said the output voltage is proportional to the temperature in these sensors. The kind of sensor that we shall be showing you, it will be proportional to the degree centigrade the temperature. And this has all this circuitry built inside it, you do not need any kind of external calibration. And it has an accuracy of $+\text{-} 0.25^\circ\text{C}$ that is often sufficient in most applications. And it also consumes very low power, 60 microampere current during operation. So, it is also a very low-power device, and it can operate over a wide range of temperatures from -55 to $+155$ degree Celsius. The power supply range is also from 4 to 20 volts.

The other point to note is that the middle point is generating an analog output voltage and it is proportional. So, what is the constant of proportionality? The constant of proportionality is such that there will be an increase in 10 millivolts in the output voltage for every degree centigrade rise in the temperature, this is how this device has been built.

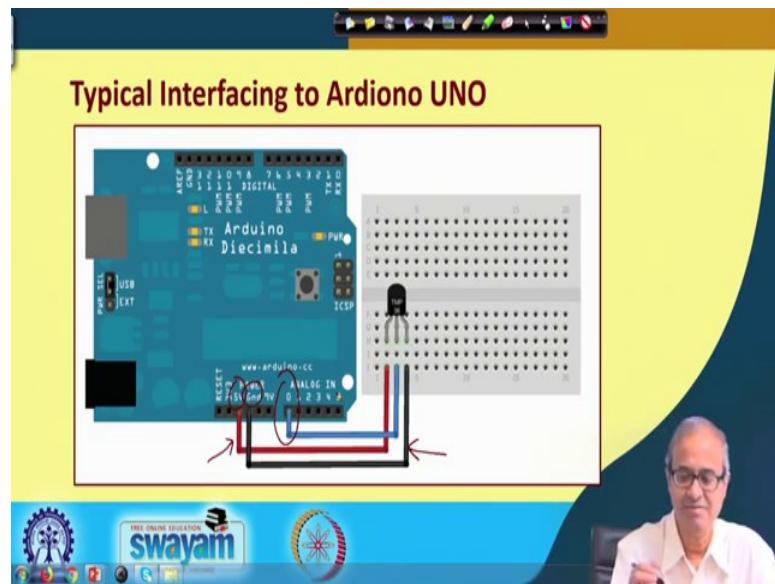
(Refer Slide Time: 08:41)



Internally I am not going into the detail explanation, this device looks like this, you see there are two transistors, some diodes, there are some amplifiers, there is a constant current source. And these two transistors are very peculiar, one of them has an emitter which is 10 times larger than the other. You see I am not trying to explain how this circuit works, but the idea is that the currents that are flowing through the transistors there is a strong dependence on the ambient temperature that is how the temperature sensing is done.

And the difference in the currents that are flowing into these two transistors is amplified and the voltage across this resistance R1, is actually becoming proportional to the absolute temperature. And on the other side this absolute temperature value, this voltage through some other circuitry is being converted into a voltage that can be made proportional to either degree Fahrenheit or degree centigrade. So, there are two families of LM35 device I told you, one gives you the output voltage proportional to degree centigrade other proportional to degree Fahrenheit.

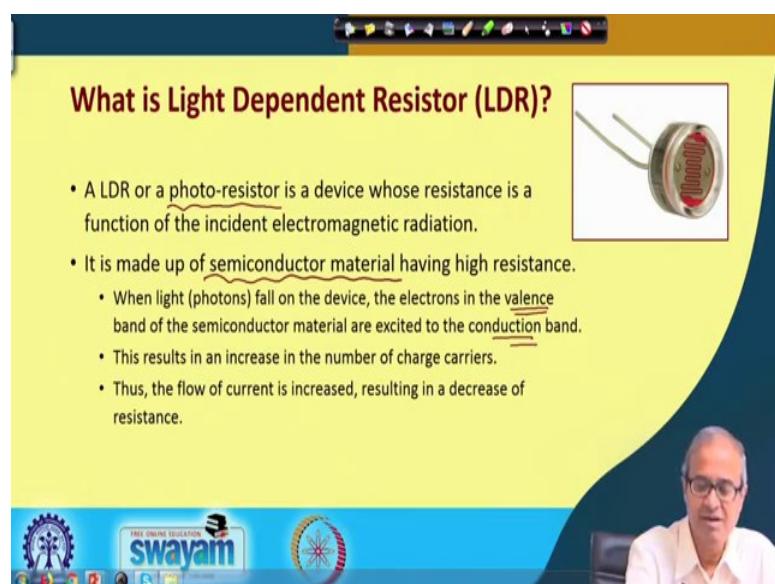
(Refer Slide Time: 10:17)



But to use it is extremely simple. Here I am showing a diagram where with an Arduino UNO board we are trying to connect a LM35. The leftmost terminal we are connecting to 5V, this is the terminal which is generating 5V we are connecting it directly to 5V, the rightmost wire this black one is connected to ground.

And the middle one that is supposed to generate the analog output proportional to the temperature is connected to one of the port lines. You see we have connected to one of the analog inputs, because it is an analog voltage. You can read the value of the analog voltage through built-in A/D converter, you can read the data and you can make a calculation what is the actual temperature outside.

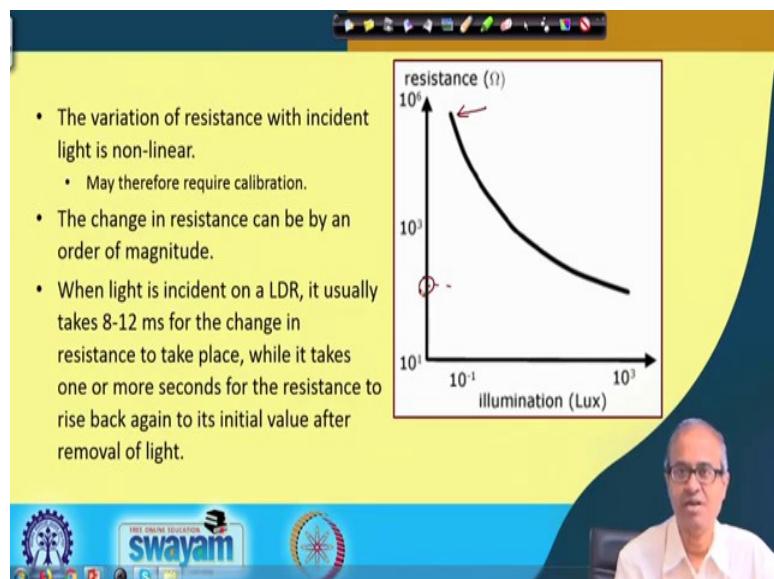
(Refer Slide Time: 11:23)



Now, let us talk about a sensor called light dependent resistor (LDR) that can sense light. LDR is a very small device. It looks like this, it is a two terminal device and there is a transparent window, on top you can see some this kind of zigzag pattern inside it this is how an LDR looks like. This LDR is sometimes also called a photo resistor; it is a resistance whose value changes depending on the light incident on it. This is how LDR works. Depending on how much light has been incident, the value of the resistance changes. You can measure the resistance that will give you an idea about the intensity of light that is falling on it.

Internally it is made out of some semiconductor material, because semiconductor materials have this kind of photo resistive property. When light falls on them the resistivity changes. When light or photons fall on the device the electrons inside the device move from one state to the other, typically they are called valence band, they move to the conduction band. Valence band to conduction band if they move; that means, electrons become more mobile resulting in a reduction in resistance because flow of electrons result in a flow of current.

(Refer Slide Time: 13:23)



And the other property is that as the illumination increases the resistance decreases exponentially. Unlike LM35 this is not a linear device; the resistance is not proportional to the intensity level, it follows this kind of exponential curve.

And the other point to note is that the difference can be quite significant. You see in this typical curve here we have a resistance of the order of mega ohms, and on this point we have a resistance of the order of hundreds of ohms. So, from mega ohm to hundreds of ohms is a huge difference. If you have a proper circuitry to sense this change in resistance, you can very faithfully and accurately sense the level of light intensity.

The other point you note is that these devices are not lightning fast; they take a little time to generate the output. Like when light falls on these devices it typically takes of the order of tens of milliseconds for the device to respond and the output to settle down. And when light is withdrawn again you put it in a dark place, then it can take a couple of seconds for the resistance to go back to that high resistance state. This you should remember; that means, the response time is not of the order of microseconds or very lower milliseconds.

(Refer Slide Time: 15:23)

How to Convert LDR Resistance to a Voltage?

- The resistance R_1 can be chosen suitably.
- The range of analog output voltages over the expected variation in light intensity must be estimated a priori.

Now, talking about interfacing very typically we use some kind of a resistance divider circuit. Like we can use an LDR, we can use a resistance R_1 with a 5V supply, we can feed it to one of the analog input pins of the microcontroller.

So, you will have to first find out the range of analog output voltage in the expected environment. You may be using the LDR to check whether the light intensity has fallen below a threshold. So, you will have to do an experiment and find out what are the resistance values in the dark state and in the light state. So, this R_{LDR} value is changing.

Accordingly you will have to choose the value of R1, so that the change in the voltage output can be significant. You can read this analog input in the microcontroller and take a decision what to do with that.

(Refer Slide Time: 17:03)

What is a Microphone?

- A microphone is functionally the opposite of a speaker.
 - Converts sound waves into electrical signals.
 - Very similar in design to a speaker.
- How does a microphone work?
 - The diaphragm (much smaller as compared to a speaker) moves back and forth when the sound waves hit it.
 - The coil, attached to the diaphragm, also moves back and forth.
 - The permanent magnet produces a magnetic field that cuts through the coil. As the coil moves, an electric current flows through it.
 - Sound energy gets converted into electricity.

FREE ONLINE EDUCATION
swayam

A video of a man speaking is visible on the right side of the slide.

Now, the last kind of sensor that we shall be talking about in this lecture is a microphone. Microphone is a kind of a sound sensor, some sound is being generated that is captured by the microphone and it is converted to electronic format.

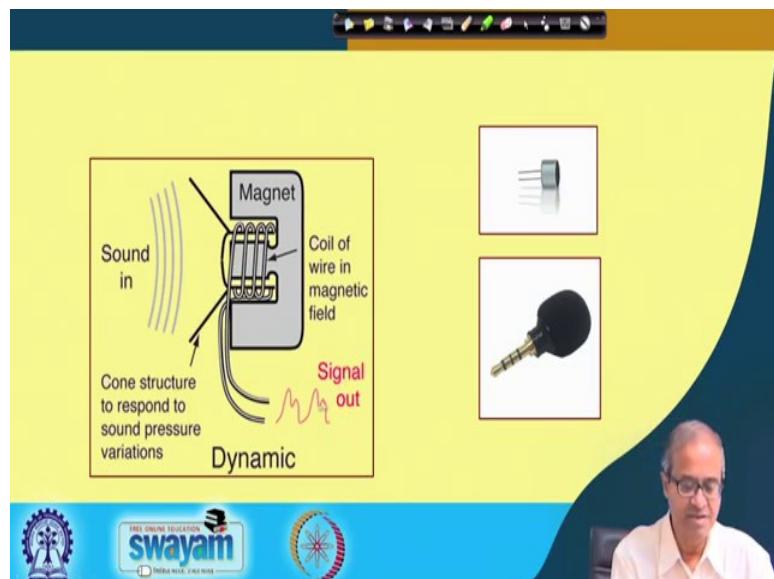
So, essentially what is a microphone? In terms of functionality internally the arrangement is very similar to that of a speaker, but it works in exactly the opposite mode. A speaker converts an electrical signal to sound; microphone converts sound into electrical signals. The design is very similar; here also there is a diaphragm. But, microphones are much smaller than a speaker, typically speakers are large in size.

In contrast microphones are very small in size; there is a very small diaphragm inside. When you speak on it the diaphragm vibrates, there is also a coil around the diaphragm around that permanent magnet, now you are not passing a current through the coil rather you are allowing the diaphragm to vibrate in response to the sound.

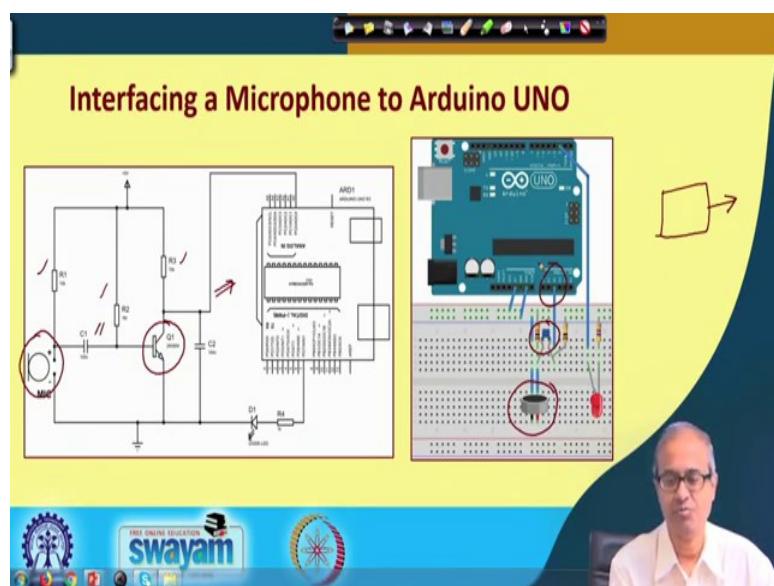
Now, as the coil vibrates inside a magnetic medium, a voltage will be induced in the coil and you can measure that voltage.

This is actually how it works. The diaphragm moves back and forth, the coil that is attached to a diaphragm will also move back and forth. Just like a speaker there is a permanent magnet there will be magnetic field in which the coil is sitting. So, as the coil moves an electric current will be flowing through it, because of which a voltage will get induced. So, sound energy gets converted into electrical energy. This is how a microphone works.

(Refer Slide Time: 19:51)



(Refer Slide Time: 20:43)



Now, interfacing is also not quite difficult. I am showing a typical interface there can be simpler ways of doing it also. You see here this is the symbolic diagram of a microphone here I am showing you a transistor level amplifier which consists of some resistances and some capacitances. Whenever you are speaking sound is being converted into a voltage, which is in the 0 to 5 volts range. Here the typical interface is shown where you can see the microphone sitting here and the other circuitry you can see here, this is your transistor and you have made the connection with this Arduino UNO.

Let me also tell you when you want to buy a microphone of this type for some experiments, you will find that there are some products available that already have this circuitry built into it. So, if you buy it you need not have to construct this circuit, directly this unit will give you a voltage that will be proportional to the sound.

With this we come to the end of this lecture. In the next lecture we shall be continuing with some more of these devices that we will be using to show you or demonstrate the interfacing experiments.

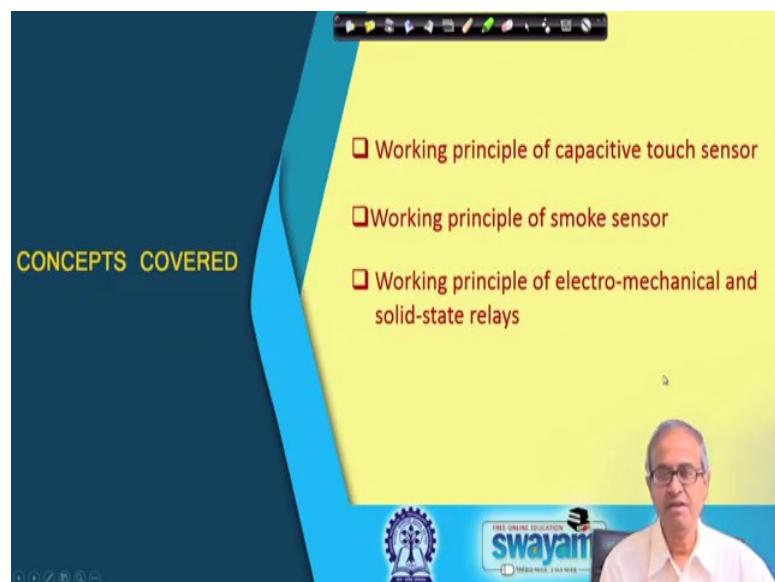
Thank you.

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

Lecture – 17
Output Devices, Sensors and Actuators (Part III)

We continue with our discussion on Sensors and Actuators.

(Refer Slide Time: 00:29)



Specifically, we shall be talking about capacitive touch sensor as an input device, smoke sensor also as an input device, and then we shall be talking about different kinds of relays.

First let us talk about capacitive touch sensor that can detect physical touch. We are all familiar with many of the touch-activated devices, like our mobiles are touch sensitive. We use our fingers to navigate on the screen; there are many other input devices where you can touch the screen and feed our inputs; there is no separate keyboard. This is one technology --- the capacitive touch sensing using which we can implement such kind of a sensing device.

(Refer Slide Time: 01:29)

What is Capacitive Sensing?

- Capacitive sensing is a technology based on capacitive coupling that can detect and measure anything that is conductive or has a dielectric different from air.
- The working of a touch sensor is similar to that of a simple switch.
 - When there is contact with the surface of the touch sensor, the circuit is closed inside the sensor and there is a flow of current.
 - When the contact is released, the circuit is opened and no current flows.
- Two types capacitive and resistive.
- Many applications in human interface devices:
 - Trackpads, touchscreens, touch switches, etc.

Now, first let us see what this capacitive sensing is all about. This is basically a technology based on capacitive coupling. What is a capacitance? When two materials like parallel plates are brought close together, there will be a capacitance. When you bring your finger close to a material, your finger and that material will form some kind of a capacitance because there will be some moisture in your finger, your body is also conductive. So, if you bring your finger that will also affect the value of the capacitance. The touch sensor that is built out of this capacitive sensing is similar to a switch. Whenever you make a touch some switch is closed, when you remove the touch the switch is open.

So, instead of a normal push button switch, you can also use a touch kind of a switch. And broadly speaking this kind of touch sensors can be either capacitive or resistive. Of course, the capacitive touch sensors are much more flexible and better in terms of performance. As I told you there are many applications where we use touch sensors; like track pads. Many of the laptops have no separate mouse; there is a flat surface, where you have to move your finger to move the cursor on the screen; that is called a trackpad.

(Refer Slide Time: 03:25)

Principle of Operation

- The capacitance of a parallel plate capacitor = $\epsilon_0 * \epsilon_r * A / d$, where ϵ_0 is the permittivity of free space, ϵ_r is the relative permittivity of the dielectric material, A is the area of the plates, and d is the distance between them.
- The capacitance will increase if a conductive object touches or approaches the sensor electrode.

Diagram illustrating the principle: A parallel plate capacitor with area A and separation d is shown. The capacitance is given by $C = \epsilon_0 A / d$. As a finger approaches the top plate, the capacitance increases to C_0 , and further to $C_0 + C_r$.

swayam

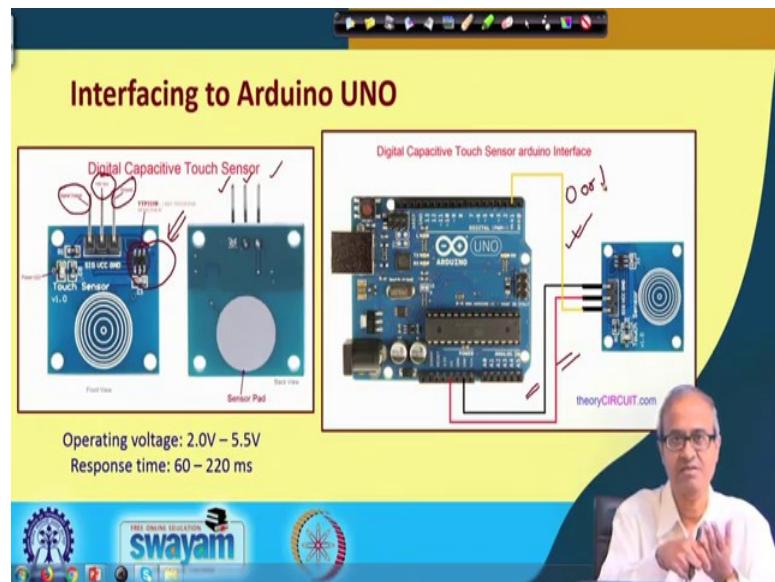
A video feed of a teacher is visible on the right side of the slide.

This diagram shows how a parallel plate capacitor looks like. There is a plate whose area is A , there are two parallel plates, the separation is d , and there is a material in between called dielectric, which has a dielectric constant. The value of the capacitance is given by this expression.

Now, ϵ_0 is defined as the permittivity of free space, and ϵ_r is the permittivity of the material between the two plates. Now when you are bringing a finger near to it, there will be a coupling through your finger. If you put your finger, these two plates will get a coupling via a capacitive effect. Essentially a touch sensor works in this way.

So, when the object touches or even you are bringing your finger in very close proximity, the value of the capacitance will increase. And if you have an appropriate circuitry to detect the change in capacitance, you can detect the touch.

(Refer Slide Time: 05:19)

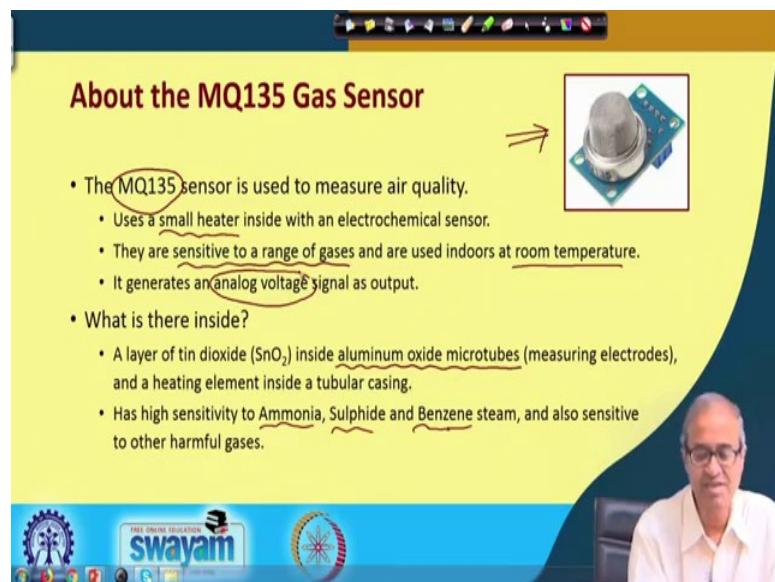


One kind of device we shall be using as part of the experiment looks like this. You see here there are circular patterns, this is your sensor pad; if you put your finger on top of this circular pattern area, the value of the capacitor changes. And in this board itself there is some electronic circuitry. You see here there is a small IC that has all the sensing and conditioning circuitry inside it. So the interfacing becomes very simple; there are three pins --- one is your + 5 volt power supply, other is ground and the third one is analog signal output; depending on your touch what is the output voltage that you can read.

When you are interfacing such a sensor to your microcontroller board, it becomes very easy. You connect the signal output to one of the port pins. Now this port pin is not an analog output pin, it is a digital output pin. It says whether you have touched it or not touched it, 0 or 1.

Next let us talk about smoke sensing. There are many applications where you need to install a sensor. For example, in your home you want to install a sensor to check whether there is a leakage of your LPG gas in the cylinder or not. The sensor should be able to respond to LPG gas fumes, there has to be some kind of a sensor in that way.

(Refer Slide Time: 07:26)



About the MQ135 Gas Sensor

- The MQ135 sensor is used to measure air quality.
 - Uses a small heater inside with an electrochemical sensor.
 - They are sensitive to a range of gases and are used indoors at room temperature.
 - It generates an analog voltage signal as output.
- What is there inside?
 - A layer of tin dioxide (SnO_2) inside aluminum oxide microtubes (measuring electrodes), and a heating element inside a tubular casing.
 - Has high sensitivity to Ammonia, Sulphide and Benzene steam, and also sensitive to other harmful gases.

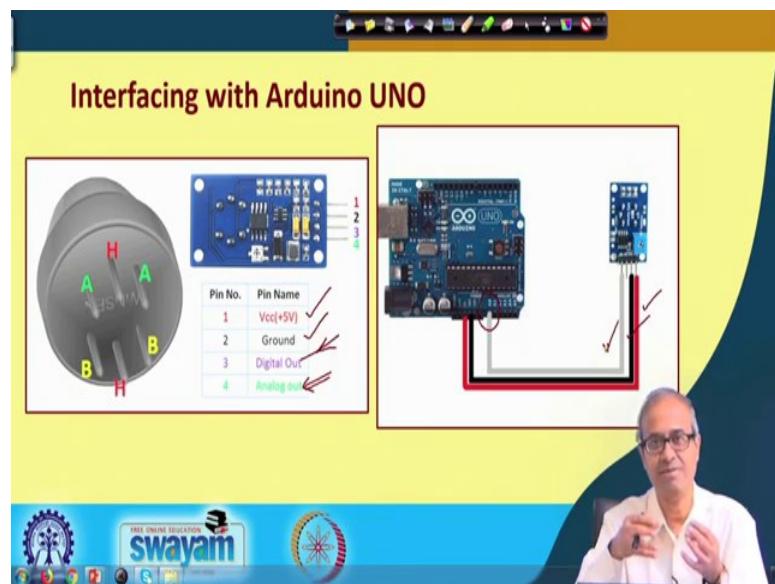
FREE ONLINE EDUCATION **swayam**

Here you see a sensor on the right side. In the experiment we will be showing you this kind of a sensor. You see there is a small mesh kind of a mask where you have to give that gas, you can put it in the environment where you are sensing the gas. It can also check your breath; you can exhale on top of it. It can check whether your breath that is coming out contains means alcohol vapor or not. And the name of this sensor is MQ135 that we shall be using.

Internally there are a lot of things. Although it is very small in size, there is a very small electric heater that heats up the material inside. And there are some aluminum oxide micro tubes, very thin tubes inside it, and there is a heater coil in the middle. So, when it heats up the smoke that enters through this graded surface, will react with those tubes. There is a thin layer of tin dioxide, which is put inside the micro tubes and acts as the main sensor.

The property of this tin dioxide material varies with the kind of gas that it is being exposed to. This kind of a sensor can respond to gases like ammonia, sulphide, benzene and also alcohol and other harmful gases, but depending on the type of gas, how much change there will be in the output voltage will vary, it varies from gas to gas. These sensors are sensitive to a range of gases and you can use them in room temperature. It can directly generate an analog voltage signal as output, and it is very easy to interface.

(Refer Slide Time: 09:58)



Like I am showing this device, it has four pins; one is the power supply 5 volts, ground, then this is analog out. Here you get a continuous voltage depending on the gas, but you can also have another digital out signal, that will tell you whether there is a gas or no gas. There is a threshold level, which can be set and depending on that you can get either a 0 or 1.

If you do not require the digital out you want only the analog out, then you can only connect these three pins. The analog out you can connect to one of the analog input pins and 5 volts and ground.

This circuit has all the required electronic circuit inside it, so that when you interface it, it becomes very easy for the user. That is the big advantage.

Now, let us come to actuators. We only talked about different kind of sensors from where we can read the data, but now we want to also control some devices. We may want to turn on or turn off a heater, we want to turn on or turn off our AC machine or anything you can think of. Most of these devices are high power devices, you cannot directly control them from your microcontroller in terms of voltages. So, you need some kind of a device which can switch high power electric or circuit lines, these are called relays; you need relays for those kind of applications.

(Refer Slide Time: 12:25)

What is Mechanical Relay?

- It is a device that can turn on or turn off power supplied to another device.
- For switching, we need to apply a small amount of power (to an electromagnet).
- This allows high-power circuits to be controlled by low-power devices.

Relay Circuit Diagram:

```
graph LR; Input((Input)) -- "To Ground" --> Gnd(( )); Input -- "To 0-5V Digital Output" --> IO(( )); IO --> Relay((Relay)); Relay --> Load((Load)); Load --> Gnd; Load --> Diode((Diode)); Diode --> Gnd; Load --> Power(( )); Power -- "Pulse (Optional)" --> Load;
```

Relay Components:

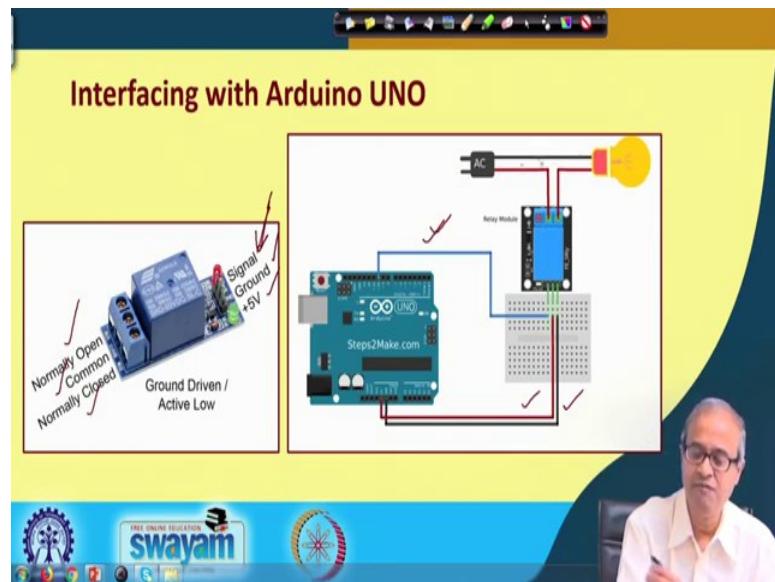
- Two physical relays shown: one with internal components visible and one in a sealed case.
- A blue relay component shown with a diode connected to ground.

Relays can be of two types, mechanical and solid state. First let us talk about mechanical relay. Sometimes these are also called electromechanical relays, because you are controlling a mechanical device, using electrical signals. On the left you see some pictures of this kind of electromechanical relays, where through a transparent material you can see what is inside. But some of these are also available in a sealed envelope, you cannot see inside, like the the one on the right. This is the relay that we shall be using in our experiment.

Now, internally you see what it is there. There is a small electromagnet inside and there is a switch with one movable terminal spring-loaded switch. When the electromagnet is activated, it becomes a magnet, the switch is attracted and the circuit closes and if you withdraw the current the circuit again opens. Now this circuit which closes and opens is a high power circuit, this can be used to drive a high load.

But on the other side the current that you are passing through this electromagnet to pull and release the switch, this can be a very low power circuit, this can be working at 5 volts power supply. You are using a very low power signal to control a higher power circuit. These two circuits are isolated, they are not directly connected; there is no physical connection between these two circuits.

(Refer Slide Time: 14:23)



For controlling you connect one point to ground, 5 volt, and here you are applying a signal to turn it on and off. There are two kinds of relays, one is called ground driven means if you make it 0 it will be on. And there is another kind which is not ground driven it is voltage driven, when you make it 1 it will be on. And on the output side you see there are three terminals that are provided, these are called normally open (NO) normally closed (NC), and the common terminal.

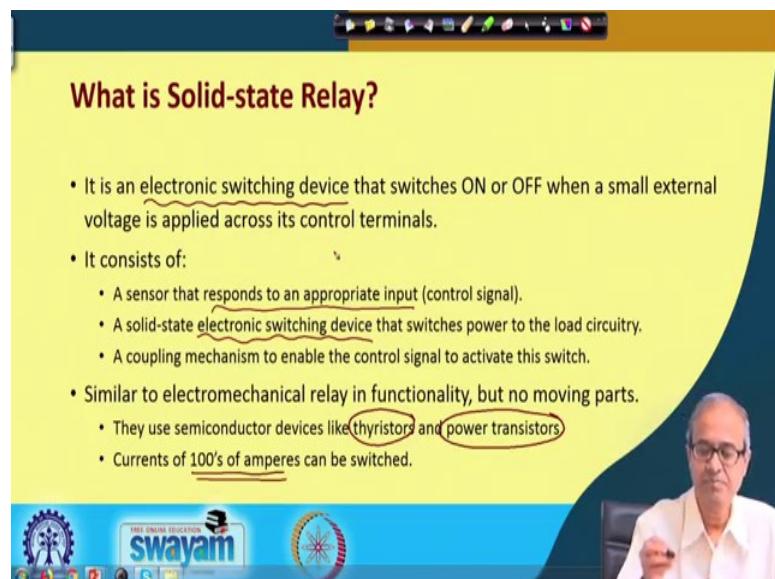
NO means normally open terminal and common are normally open internally, when this switch is open. So, it is open when the switch is opened, it gets closed when the switch is closed, but NC is the reverse. If you connect a device between NC and common, then when the switch is open this circuit is normally closed, but when you activate the switch this circuit will be open.

I have shown a very simple interface with this kind of module. On one side you are controlling this from your Arduino board, Vcc, ground, and through a digital port pin you are controlling the relay. And on the other side I am using the normally open connection. This I am connecting to a circuit, let us say a bulb to the AC mains.

When I am activating it and the relay becomes on, the bulb will glow, and if it is off the bulb will be off this is how it works. The interface is very simple, we shall be seeing this kind of interfacing experiments later. Now there is another version of a relay that is not

based on electromagnets pulling and releasing the switches mechanically, but these are called solid state relays.

(Refer Slide Time: 16:58)



What is Solid-state Relay?

- It is an electronic switching device that switches ON or OFF when a small external voltage is applied across its control terminals.
- It consists of:
 - A sensor that responds to an appropriate input (control signal).
 - A solid-state electronic switching device that switches power to the load circuitry.
 - A coupling mechanism to enable the control signal to activate this switch.
- Similar to electromechanical relay in functionality, but no moving parts.
 - They use semiconductor devices like thyristors and power transistors.
 - Currents of 100's of amperes can be switched.

Solid state means everything is built inside a chip using semiconductor device.

This you can say is an electronic switching device, there is no mechanical moving parts like in an electromechanical relay. Here the kind of devices you require for the switching are high power semiconductor devices like thyristors or power transistors, they can switch very high currents at high voltages. For example, hundreds of amperes of currents can be switched.

Inside there will be some electronic circuitry, which will be responding to some appropriate input that will be sending a control 0 or 1, and this thyristor or power transistor whatever is there will be switched on or off, and that will be turning the power on or off to the device that is being controlled.

(Refer Slide Time: 18:17)



This is how solid state relay works. These are some of the pictures of solid state relays. You see this is the middle one, it says that it works from 24 to 380 volts AC. These are much smaller in size because the semiconductor made of solid state devices, they are very small. The one on the left you can see is 4 centimeters total in size.

With this we come to the end of our discussion on various kinds of output devices like LEDs and LDRs, various kind of sensors and actuators. Now during the actual hands on and demonstration sessions, we shall be showing you all these sensors and actuators in use. How to use them, how to interface them, and how to write a program in the microcontroller to control them in the way we want. These we shall be discussing in our subsequent lectures.

Thank you.

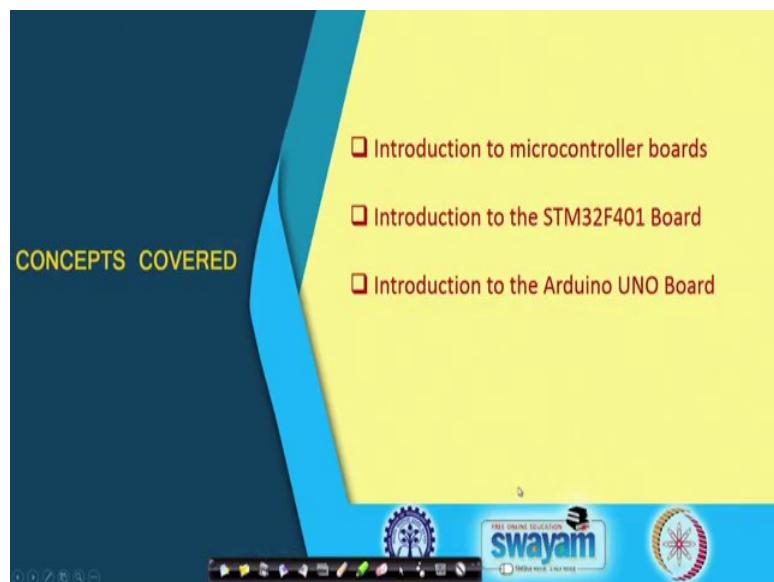
Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 18
Microcontroller Development Boards

Welcome to week 4 of the course. In this week I will take you to a tour of various Microcontroller Boards and specifically I will be discussing about two boards; one is STM board and another is Arduino board. And I will also show you how you can program using these two boards specifically.

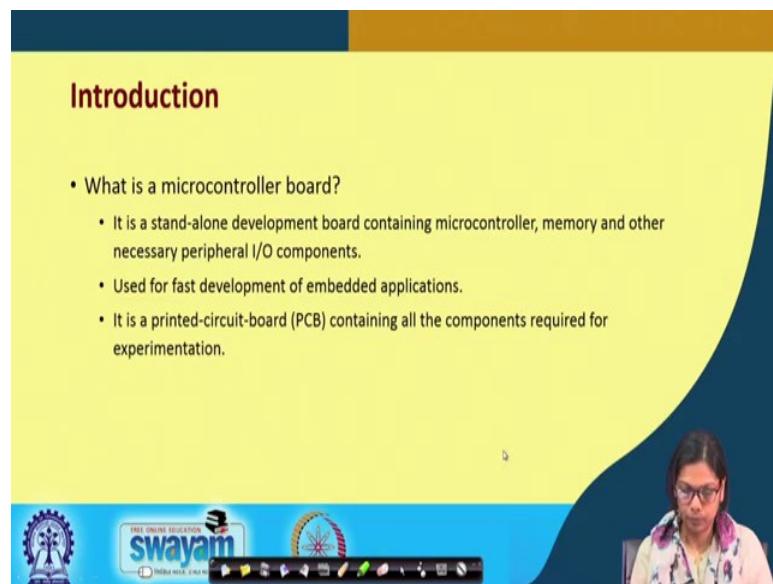
There are other development boards as well, but in this course we will be taking the opportunity to take these two specific boards into consideration and we will be discussing the experiments based on these two boards.

(Refer Slide Time: 01:09)



The concepts that will be covered in this lecture are like this. I will introduce microcontroller boards, we already know what a microcontroller is. I will be discussing about two boards; one is STM32F401 board and another one is Arduino UNO. All the experiments that we will be doing in this course will be based on these two boards.

(Refer Slide Time: 01:44)



Introduction

- What is a microcontroller board?
- It is a stand-alone development board containing microcontroller, memory and other necessary peripheral I/O components.
- Used for fast development of embedded applications.
- It is a printed-circuit-board (PCB) containing all the components required for experimentation.

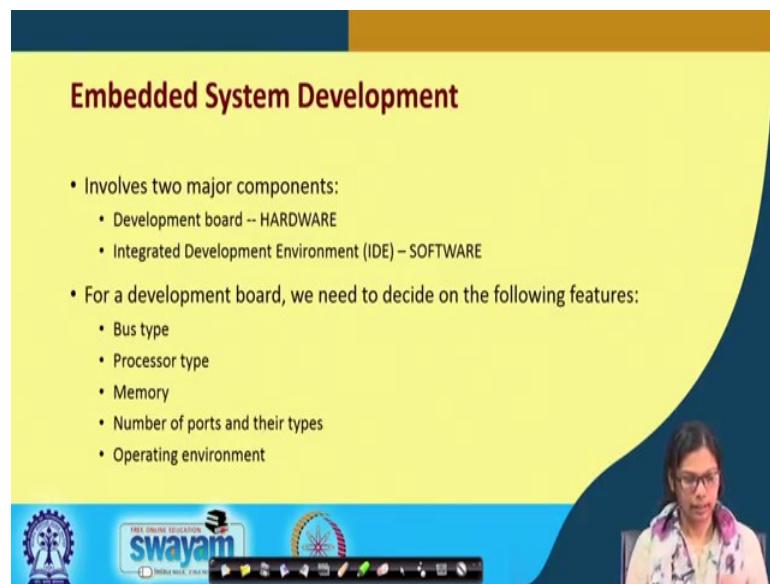
Let me talk about what is a microcontroller board. We can define it as a standalone development board containing microcontroller, memory and other necessary peripheral I/O components. Why it is used? It is used for fast development of embedded applications. And, it is generally built on a printed circuit board that is called PCB containing all the components that are required for the experimentation.

Microcontroller boards are used for fast development of embedded applications. Nowadays we see lot of embedded applications. If you think of a small example, let us say a smoke detector, what is it? It is a system that will detect smoke, whether it is present inside a particular room or a particular area. In that case what do you require? You require various other input output devices.

But more specifically you need a microcontroller board through which you will be connecting a sensor that will read some analog values, which is in terms of smoke inside the room, and it will convert digitally to some value. And then we will do some kind of operation to find out whether there is smoke inside the room or not. Based on that detection various things can be done; one thing is that a buzzer could be on, and then you can actually hear the sound and can make out.

The microcontroller boards are so powerful that all input output ports come along with it, such that you can actually connect directly with a sensor, with the buzzer etc. to make the complete system.

(Refer Slide Time: 04:48)



Embedded System Development

- Involves two major components:
 - Development board -- HARDWARE
 - Integrated Development Environment (IDE) – SOFTWARE
- For a development board, we need to decide on the following features:
 - Bus type
 - Processor type
 - Memory
 - Number of ports and their types
 - Operating environment

The slide is part of a presentation on Swayam, as indicated by the logo at the bottom. A woman in a pink shirt is visible in the background of the slide.

When we talk about this embedded system development, this involves two major components. One is the development board, that is the hardware that is required, and another is Integrated Development Environment, we call it IDE that is the software.

So, when we think of designing a system these two things are required. We need a hardware that is the board, but how do we program it? We need to have some software associated with it through which we can program it. We will also look into what are the various kinds of IDE that we will be using for the two boards. Now when we try to design a system, we always think of what kind of development board we should use.

In that regard which development board to use is based on the following features; one is the bus type another is the processor type. What kind of processor we are using? The memory; memory is one of the vital important parameter that is required for any design, because if you want to dump a very large program and you have limited memory you need to think how will you do it.

So, memory is an important factor that is to be decided when you develop an embedded system, number of ports and their types. We need input output ports; we also need analog ports. We need to decide upon that or what kind of ports are there in that particular board and the operating environment. In some development boards we use some integrated development environment, but for some the compiler is already available online.

The only requirement is that you need to have internet connection because you will be using an online compiler. To compile your code into the development board you need that particular software.

(Refer Slide Time: 08:07)

Some Common Development Boards

- STM32F401 Nucleo
 - Based on ARM Cortex-M4, 96KB SRAM, 512KB Flash
 - 84MHz clock
- Arduino Uno
 - Open-source microcontroller board based on Microchip ATmega328P, 2KB SRAM, 32KB Flash, 1KB EEPROM
 - 16MHz clock
- Raspberry Pi 3 Model B
 - Quad-core 1.2GHz Broadcom 64-bit CPU
 - 1GB RAM, wireless LAN, Bluetooth, Ethernet, USB, etc.
- PIC 16F877A
 - 8-32KB Flash, 10-80MHz clock

These are some common development boards. The one we will be using is STM32F401 Nucleo board, it is based on ARM Cortex-M4, and it has got 96 KB of SRAM. It has got 512 KB of flash memory, clock speed of 84 MHz. Another very popular board is Arduino UNO, which is a open source microcontroller board based on Microchip ATmega328P; it has got 2 KB of SRAM and 32 KB of flash, it has also got 1 KB of EEPROM. It has a 16 MHz clock.

If you see the features of STM board and Arduino UNO, you can clearly make out that STM is much powerful as compared to Arduino. You see the kind of memory it is having; 96 KB compared to 2 KB, 512 KB of flash compared to 32 KB of flash, clock speed of 84 megahertz compared to clock speed of 16 megahertz.

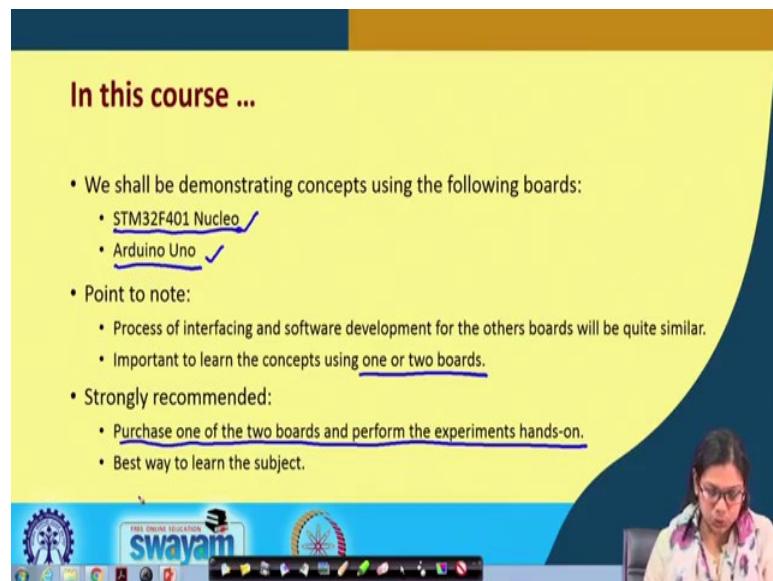
Now I should also tell you one important thing. The applications that we will be showing can be run using both Arduino UNO board as well as STM board. We can use any one of the boards for our development.

But when you think of a very huge application that you are trying to build, which requires higher memory, in that case STM board will be preferred. Now, there are even

more powerful boards one of which is Raspberry Pi 3 model B, which consists of quad core 1.2 GHz Broadcom 64-bit CPU, along with that it has got 1 GB of RAM, wireless LAN, Bluetooth, Ethernet and USB. You can see that it can be used as a computer itself. It has got high-end features that can be used for sophisticated application development although we will not be using Raspberry Pi in this particular course.

Another microcontroller board is PIC microcontroller board. This is one of the model which is 16F877A; the feature is like this. It has got 8 to 32 KB flash and the clock speed can range from 10 to 80 MHz. So, when you think of the developing very simple kind of applications, you can think of using PIC based microcontroller systems.

(Refer Slide Time: 12:53)



In this course ...

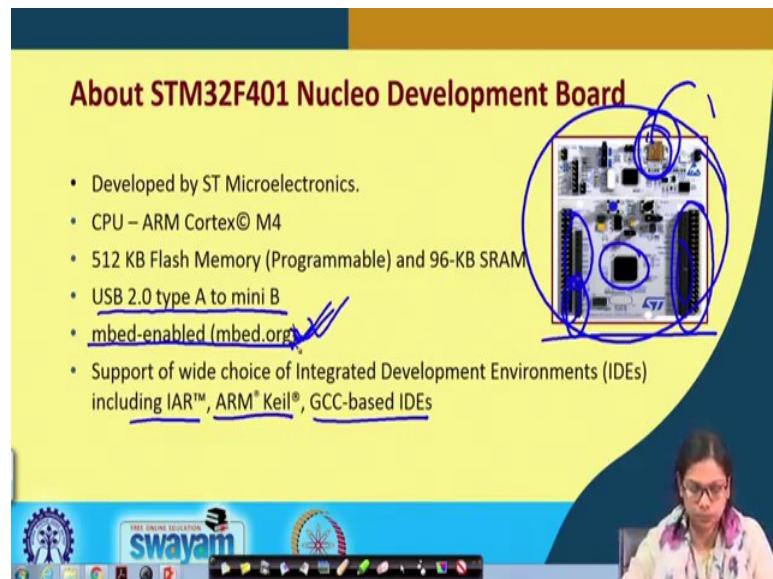
- We shall be demonstrating concepts using the following boards:
 - STM32F401 Nucleo
 - Arduino Uno
- Point to note:
 - Process of interfacing and software development for the others boards will be quite similar.
 - Important to learn the concepts using one or two boards.
- Strongly recommended:
 - Purchase one of the two boards and perform the experiments hands-on.
 - Best way to learn the subject.

As I have already told you, in this particular course we shall be demonstrating the concepts using the following boards. The first one is STM32F401 Nucleo board and another one is Arduino UNO. Now, the process of interfacing and software development for the other boards will be quite similar. As we will be only discussing about STM board and Arduino UNO, if you want to interface any other board the process will not be very different.

It is important to learn the concept using at least one or two boards, such that you know the basic process. And what is strongly recommended for this course is you purchase at least one of the two boards and perform the experiments. We will be coming across many hands on experiments in this particular course.

You should go ahead and purchase at least one of these boards and you can do the experiments along with us.

(Refer Slide Time: 14:53)



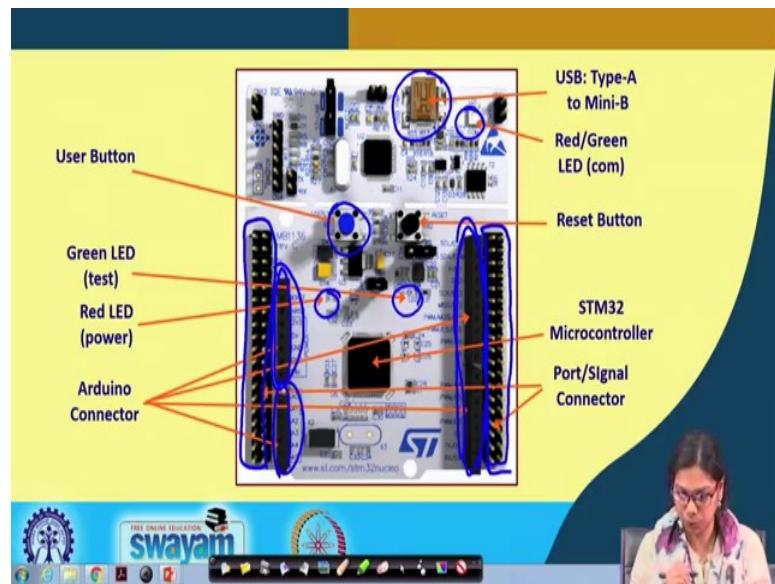
Now, I will move on with STM32F401 Nucleo development board. This is the board that we will be using. It has got some digital pins, some analog pins, there is a power, this is the USB connection through which you can connect through USB port.

This board is developed by a company called ST microelectronics. The CPU inside this particular board is ARM Cortex-M4; it has got 512 KB of flash memory that is programmable and 96 KB of SRAM. As I have already told you that this is the USB connector. Here a USB 2.0 port.

For connecting from here to your PC you require this USB. Now this particular board is mbed-enabled. So, what we do is that we go to a website mbed.org, I will give you the exact URL for it, and there we actually write the code and dump the code into this particular board. And then you connect through these IO pins or you connect through the analog pins, you can do various other things.

In this particular course we have used this online mbed compiler. There is a wide range of choice of IDE, so you can also use ARM Keil or GCC based IDE's including IAR.

(Refer Slide Time: 17:49)



This is the diagram of this board. There is a blue color button that is the user button, this is the reset button that will be used to reset the device. Here is the connector I was talking about, type A to mini B, and there is a red/green LED. Whenever you will dump a code through this USB, you will see that this red/green LED will glow. These are the Arduino connectors and there are many other ports. You can see these ports; these ports are basically signal connector that we can use. You can see that apart from these connectors there are many other connectors that can be used, but in our experiment we have mostly used these Arduino connectors.

(Refer Slide Time: 19:59)

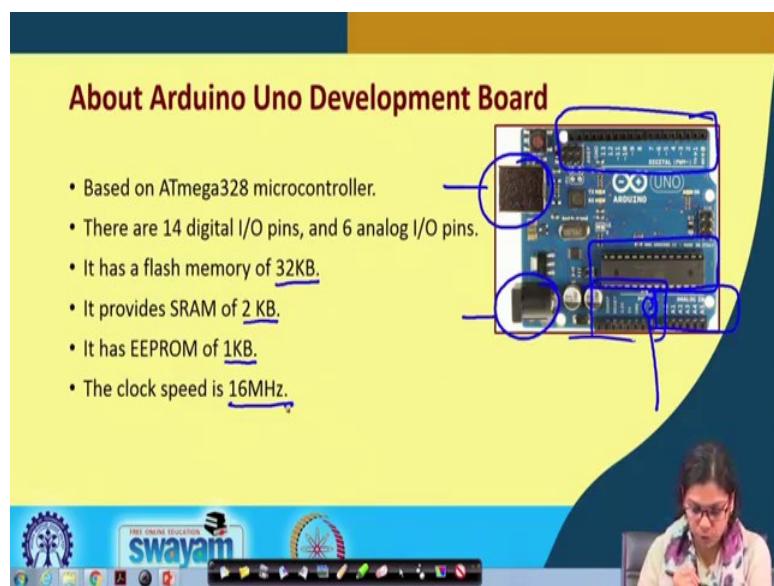
- Some points to note:
 - There are several digital I/O ports, PWM ports, analog input ports.
 - The digital I/O port lines can also be used as interrupt inputs.
 - Application development through embedded-C programs:
 - ✓ Connect the board to a desktop/laptop.
 - ✓ Write the program in C using necessary driver libraries.
 - ✓ Use an online compiler to compile the code.
 - ✓ Download the code to on-board flash memory.

As we know there are several digital IO ports, there are also PWM or Pulse Width Modulation enabled ports, and there are analog input ports. The digital I/O port lines can be used as interrupt inputs as well; this is already discussed.

I have already shown you the USB connector through which you have to connect to either a desktop or a laptop, then you have to open the id that is there and then you need to write the program.

Once you write the program in C using necessary driver libraries those are present, you can use the online compiler to compile the code. Once you compile the code this particular code gets downloaded, you can find this in your download folder. And you can then copy that particular code and put it in the device. I will take you through the tour of this particular thing that I have just told you. But these are the following steps that need to be followed when you are using this particular STM board.

(Refer Slide Time: 22:01)

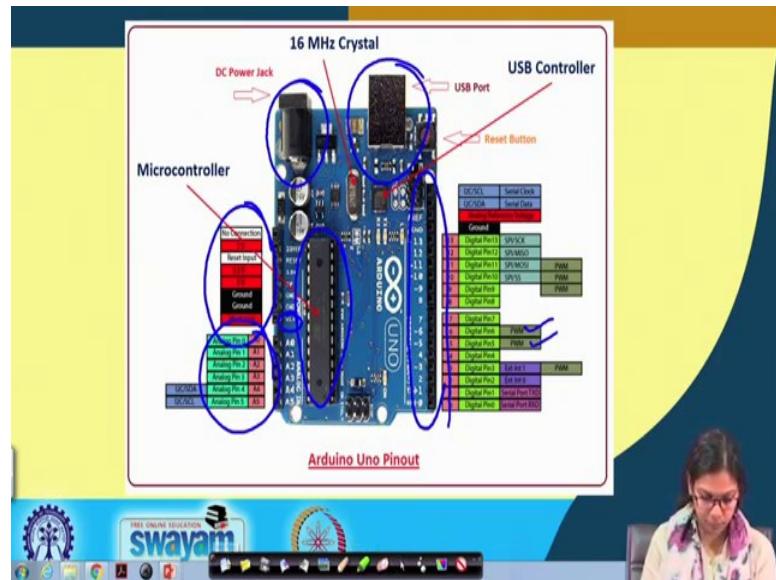


Let us now move on to Arduino UNO development board. This is the Arduino UNO development board; you can see these are the digital and PWM ports, this is the analog ports, this is for the power, this is the microcontroller chip. You can use the USB connector to power it, or you can also use a 9 volt battery to power it.

The Arduino UNO development board is based on ATmega 328 microcontroller. There are 14 digital input output pins, each of these pins can be programmed to use as an input

pin or it can be used for output pin. And there are 16 analog input pin, it has a flash memory of 32 KB, it provides SRAM of 2 KB, it has got an EEPROM of 1 KB, and the clock speed is 16 MHz. These are some typical features of this Arduino board.

(Refer Slide Time: 23:53)



As I have told you this is the USB port through which you can connect to the USB port of the PC. This is the microcontroller that we are using, these are the set of analog pins, these are for the power. And if you see these are digital pins, but some of the digital pins are also PWM, we will look into this specifically what is PWM in course of time.

You can see there is also a DC power jack through that you can power this entire board.

(Refer Slide Time: 25:02)

As I have already told you there are some other development boards as well, one of which is Raspberry Pi 3 model that is much more sophisticated. It has got two USB ports; this is for the network, these are audio and video.

It also has a HDMI port, there is a micro USB power point, this is a display port. You can also put a micro SD in this back side, it is also Bluetooth enabled, and this is the CPU, the memory, and these are the pins that can be used as ports.

(Refer Slide Time: 26:02)

PIC board typically consists of these pins and these are some typical features. This is port A; port A is from pin number 2 to pin number 7, port E is from pin number 8 to 10, and so on. Each of the pins has got certain functionalities and there are various features of this particular pin. This is a sample diagram showing PIC 16F877A microcontroller this is how it typically looks like.

These are the pins through which you can connect and you can do programming with. S

We have actually come to the end of this lecture and we have discussed about two important boards; one is the STM board another is Arduino UNO. But we have also given you an overview of how other board looks like. In the subsequent week I will be focusing on the STM board in more details.

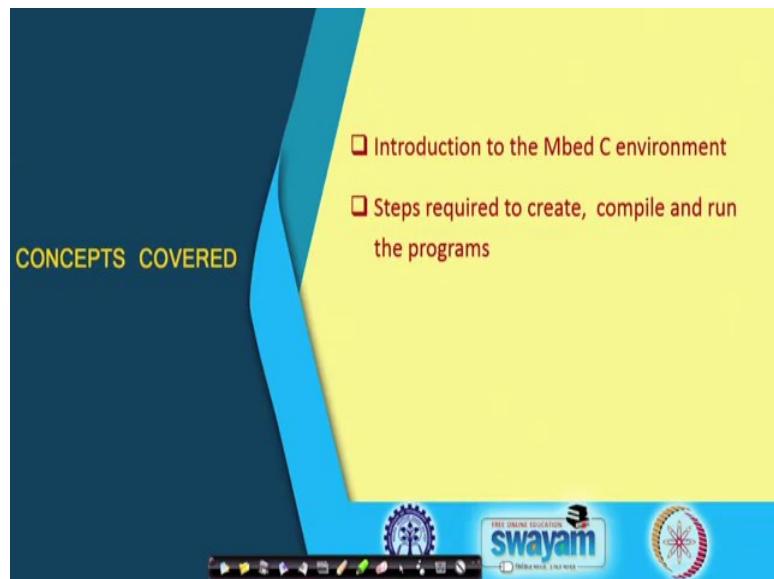
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 19
Mbed C Programming Environment

Welcome to the next lecture. In this lecture I will be talking about mbed C Programming Environment. I will introduce the mbed C programming environment and I will show you that how you can actually program using this environment.

(Refer Slide Time: 00:53)



As I said I will introduce the mbed C environment and I will show you the steps that are required to create, compile and run the programs.

(Refer Slide Time: 01:07)

What is Mbed C?

- It is an embedded software development platform.
 - Users can develop the software using C using microcontroller board specific library.
 - The program is compiled and downloaded onto the development board.
- We shall explain the Mbed-C platform using the example of STM32F401 Nucleo ARM Development Board.

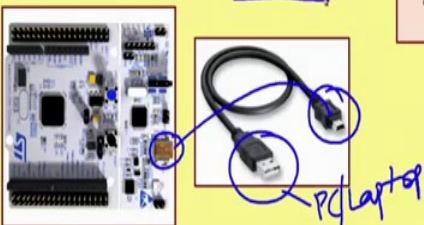


Firstly, what is Mbed C? It is an embedded software development platform using which users can develop software using C, with microcontroller board specific library. The program is compiled and downloaded onto the development boards. We shall explain the platform using the example of STM32F401 Nucleo ARM Development Board. For using this particular board you need to register, and you have to also check certain functionalities which are there, etc.

(Refer Slide Time: 02:19)

Requirements to Begin

- STM32F401 development board and driver
- USB mini to USB Type B connector
- Development environment (IDEs or mbed account).



The development environment runs on a desktop / laptop.



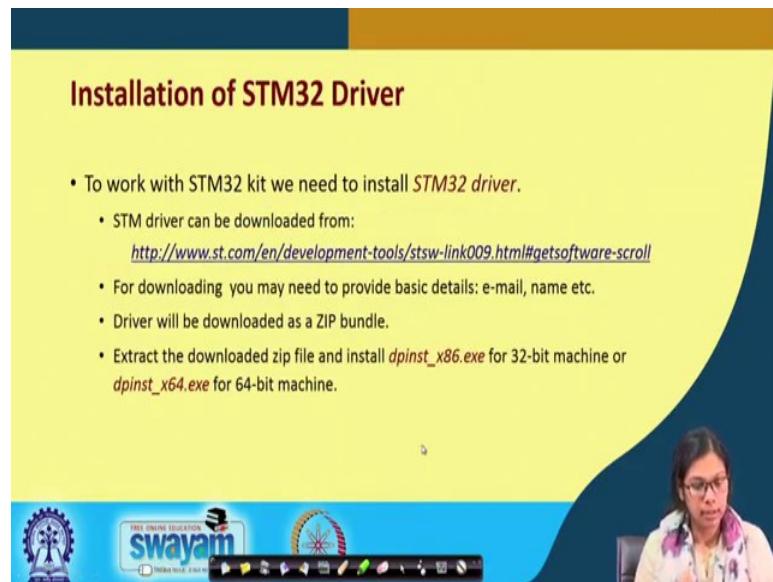
The first requirement is that you need to have this board in place and the driver installed. You also required the USB-mini to USB-typeB connector. You will be connecting this part here and this will be connected to your PC or laptop. This is the first thing you need to do. And as I had said for development environment we will be using an online complier that is mbed. So, you need to create an account in mbed.

(Refer Slide Time: 03:24)



Now, how to start? During the demonstration later in this course we will use the online complier that is present in <http://developer.mbed.org> to compile our projects. We can write our program in C or python, but most specifically we will be writing in C. The STM32 driver must first be installed on the desktop or laptop, then we will demonstrate the software development using this. Please remember that you need the USB cable, you need to go here where you will have your online complier and you have to also make sure that the STM driver is installed.

(Refer Slide Time: 04:26)



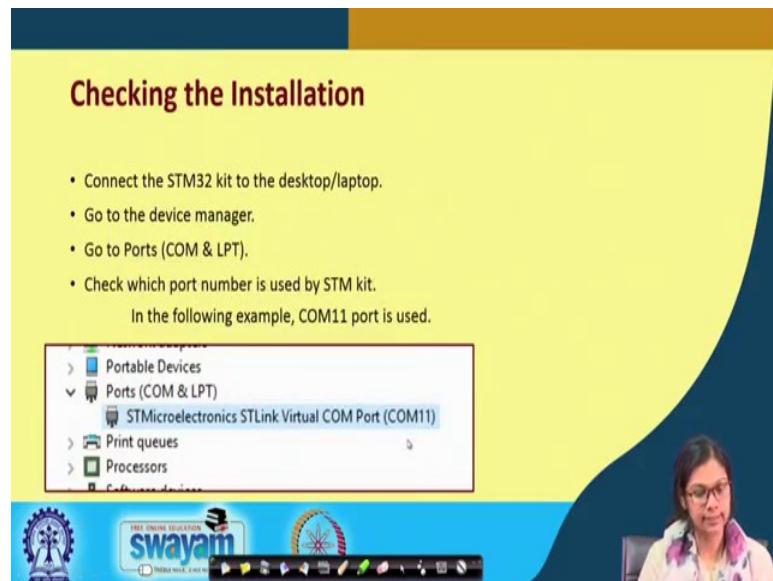
Installation of STM32 Driver

- To work with STM32 kit we need to install *STM32 driver*.
- STM driver can be downloaded from:
<http://www.st.com/en/development-tools/stsw-link009.html#getsoftware-scroll>
- For downloading you may need to provide basic details: e-mail, name etc.
- Driver will be downloaded as a ZIP bundle.
- Extract the downloaded zip file and install *dpinst_x86.exe* for 32-bit machine or *dpinst_x64.exe* for 64-bit machine.

Installation of the STM driver is fairly straightforward. All you need to do is that you need to go this particular URL, you need to download this, and then you have to provide some basic details. Then the driver will be downloaded as a zip bundle, extract the downloaded zip file and install this particular file for 32-bit machine, or if it is 64-bit machine then download this one.

This is an important step; prior to using this particular device that you need to install the drivers properly.

(Refer Slide Time: 05:37)



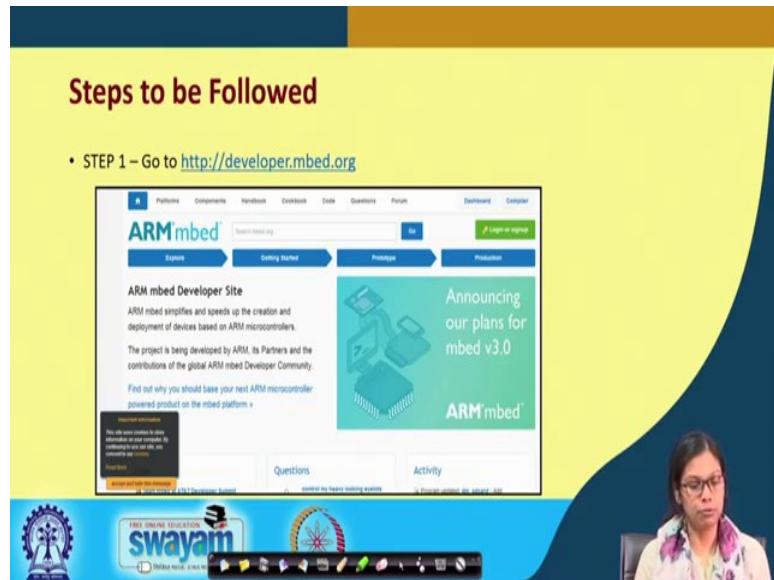
Checking the Installation

- Connect the STM32 kit to the desktop/laptop.
- Go to the device manager.
- Go to Ports (COM & LPT).
- Check which port number is used by STM kit.
In the following example, COM11 port is used.

Device Manager screenshot showing the 'Ports (COM & LPT)' section with 'STMicroelectronics STLink Virtual COM Port (COM11)' highlighted.

Next checking the installation; once it is already done connect this STM32 kit to the desktop or laptop. Go to device manager and check the port number. Once the diver is correctly installed this will be shown there.

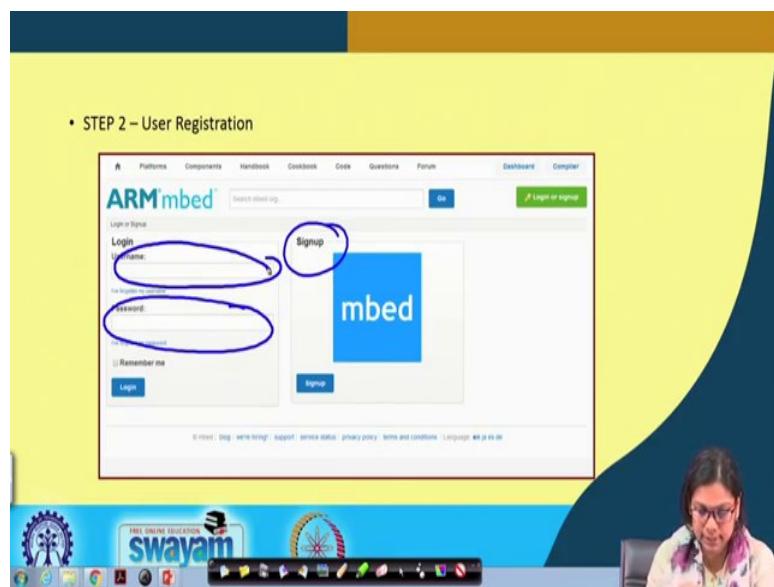
(Refer Slide Time: 06:40)



The slide shows a screenshot of the ARM mbed developer website. The main title is 'Steps to be Followed'. The first step is listed as 'STEP 1 – Go to <http://developer.mbed.org>'. The website interface includes a search bar, navigation menu (Platforms, Components, Handbook, Cookbook, Code, Questions, Forum, Dashboard, Compiler), and a banner for 'mbed v3.0'. Below the banner, there are sections for 'Questions' and 'Activity'. A woman is visible in a video overlay in the bottom right corner.

Now, what are the steps to be followed? First step is you have to go to developer.mbed.org

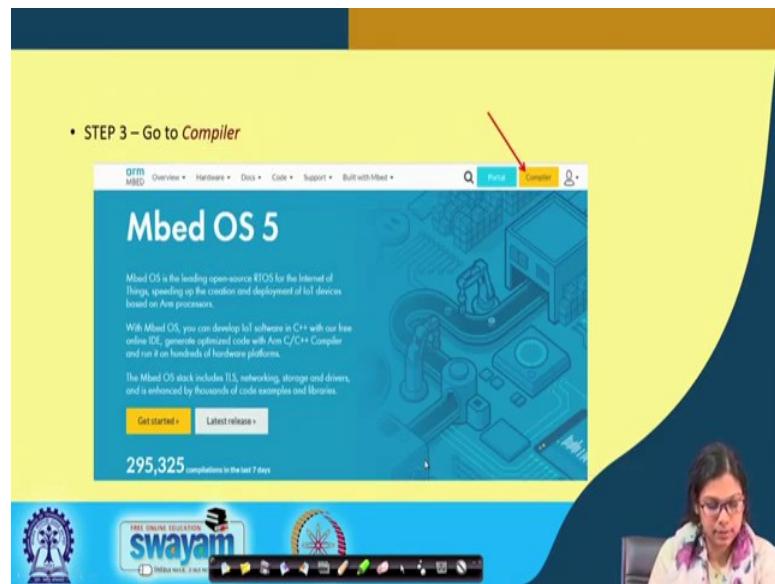
(Refer Slide Time: 06:54)



The slide shows a screenshot of the ARM mbed developer website. The main title is 'Steps to be Followed'. The second step is listed as 'STEP 2 – User Registration'. The website shows a 'Login' and 'Signup' form. The 'Username' and 'Password' fields are circled in blue. A woman is visible in a video overlay in the bottom right corner.

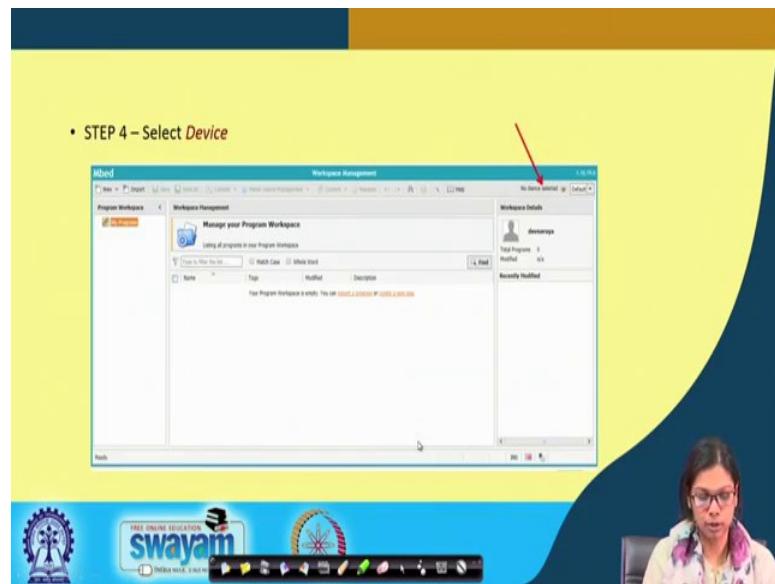
Then you see that there is a place for user name and password; of course, you have to first signup to do this. This is a simple process that you have to follow.

(Refer Slide Time: 07:20)



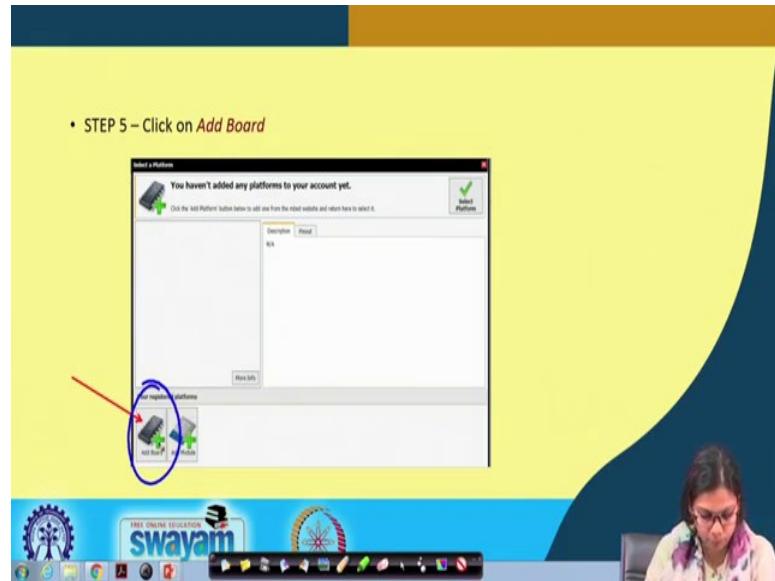
Once you login, put up your details. You will have a screen like this where you will find something which is written called compiler; click on that complier.

(Refer Slide Time: 07:46)



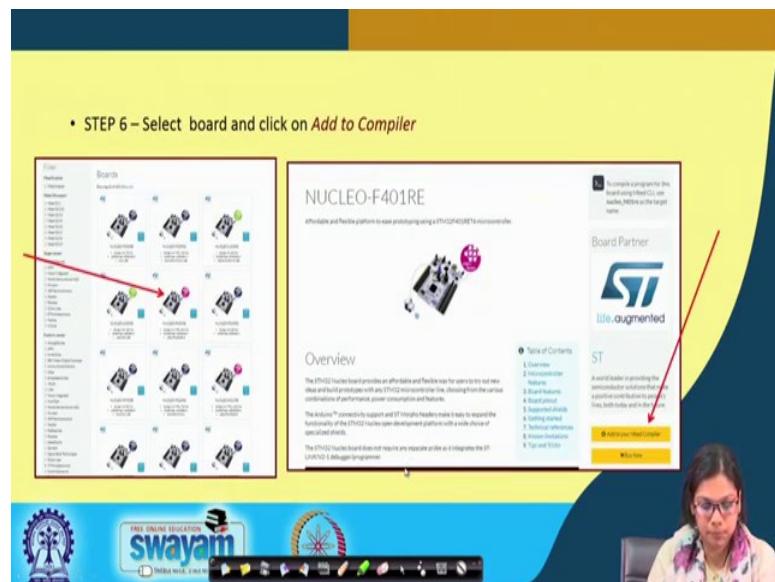
And then you move on. Once you click on Complier for the first time, those who are doing will come across this problem where you will see that it is written No Device Selected. Either it is STMF401RE or another device. Then you have to select that device.

(Refer Slide Time: 08:16)



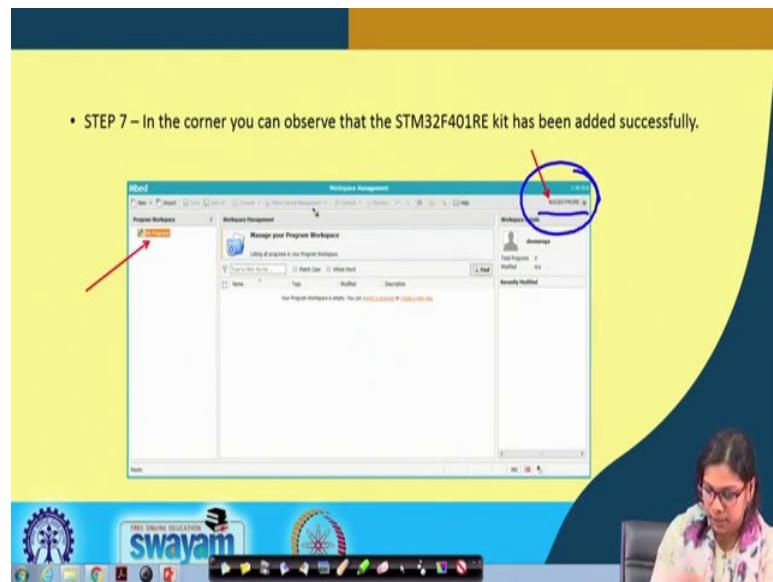
The next step you have to follow is: you click on Add Board.

(Refer Slide Time: 08:35)



And then you select NucleoF401RE; this is the board that we are using. So, whatever board you are using you have to select that particular board.

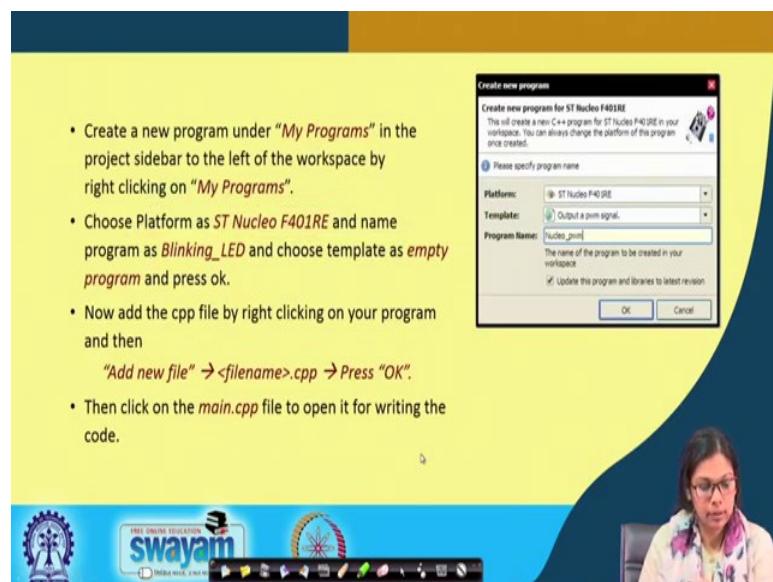
(Refer Slide Time: 09:00)



We select that board and then we move on. And once you do that you will see here that this particular board got selected, earlier it was showing no device selected. Now, once you do the following steps you will see that NucleoF401RE is selected.

Let us move on; once you see this particular screen you will also see my programs and you see that it is almost empty, but if you keep on writing the programs it will show up. Once I show you from my account you will see that there are many programs that are written here.

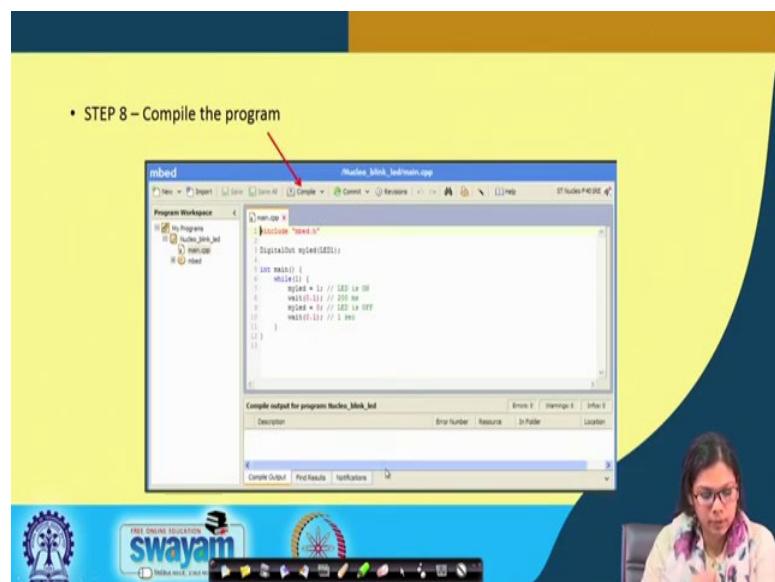
(Refer Slide Time: 09:46)



Next step is to create a new program under ‘my programs’ in the project slide bar to the left of the workspace by right clicking on MyPrograms, then you create a new one and name the program and choose an empty template and then you press ok.

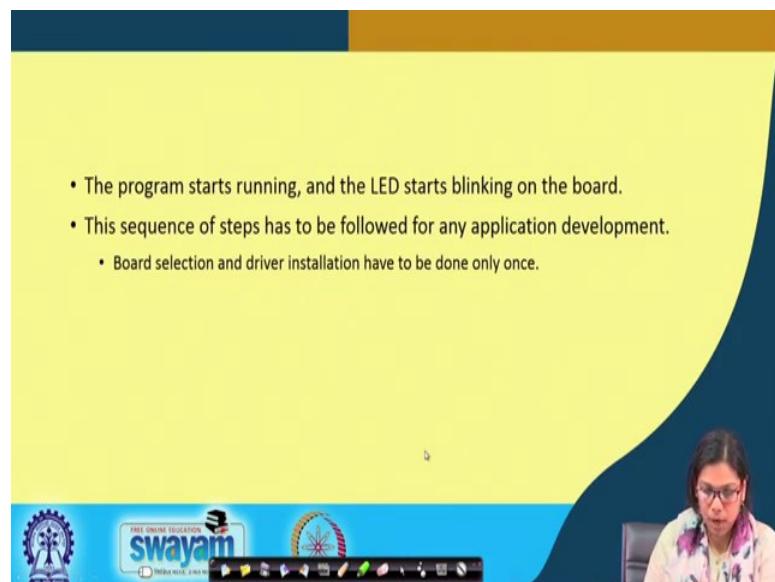
These are the steps that need to be followed up once you start writing the first program. Now what you have to do? You have to add the cpp file by right clicking on your program and then add a new filename. I will take it to this tool after just finishing few more slides.

(Refer Slide Time: 10:50)



So, this is where this is your main.cpp is. When you click here you see this is the main.cpp program.

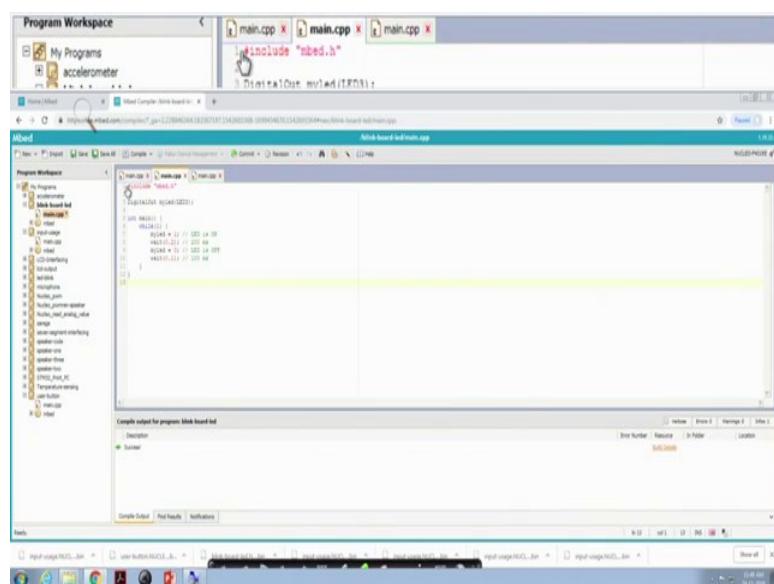
(Refer Slide Time: 11:2)



And then you have to compile the program. The compile button is shown here at the top; you click on the compile button and then it will compile. Once you compile then the code will get downloaded, save it in the Download folder.

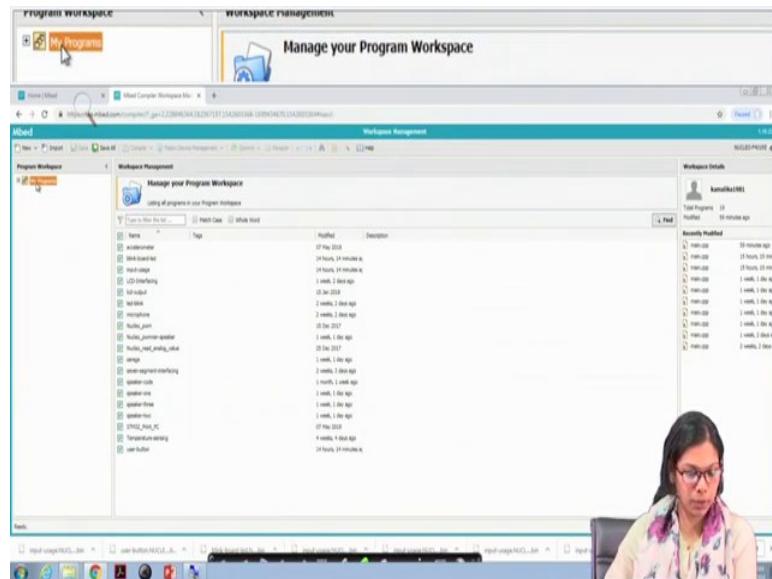
The program starts running and the LED starts blinking on the board. This sequence of steps has to be followed for any application development. The board selection and the driver selection have to be done only once.

(Refer Slide Time: 12:31)



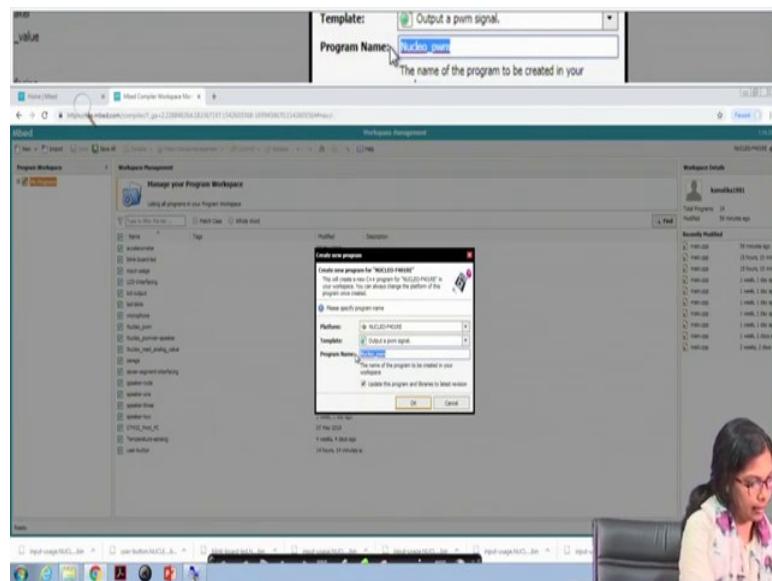
You see this is the environment; this is the simple program that I am showing. So, what we are doing the first step that is required is as I told you.

(Refer Slide Time: 13:18)



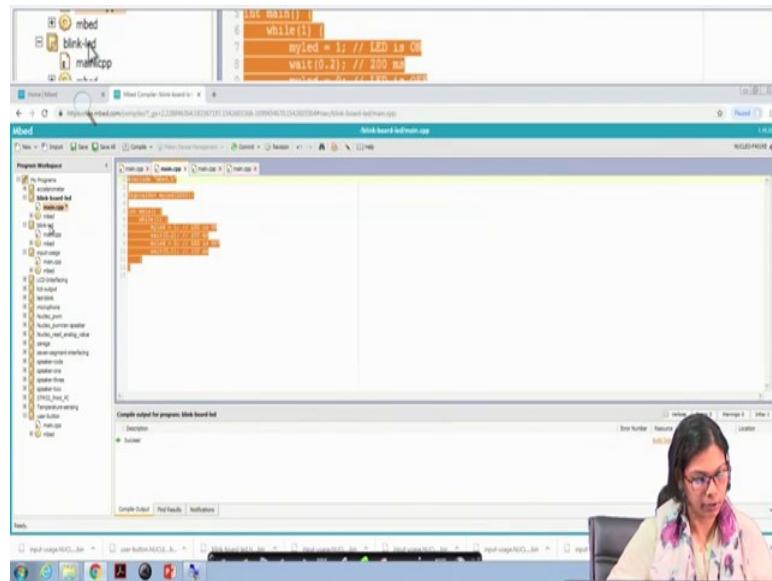
This is my program you have to right click here on main program.

(Refer Slide Time: 13:23)



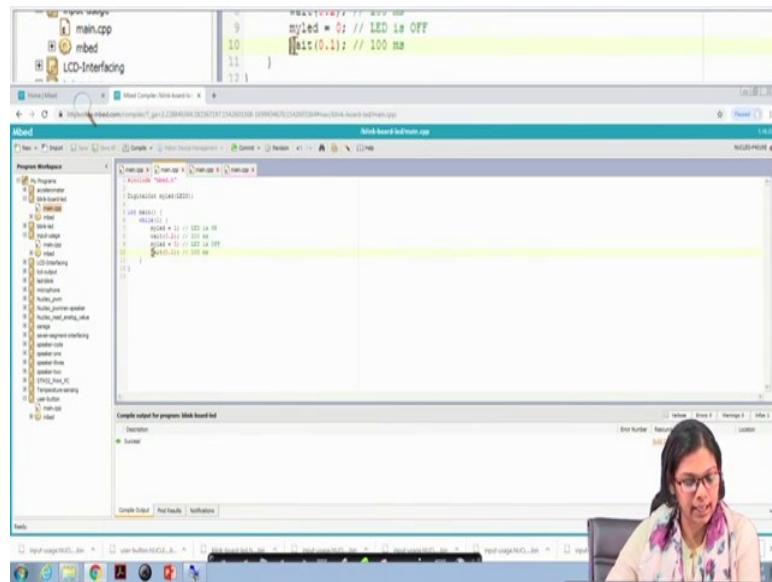
And then you see that something like this will come up, and you have to give a name to this program, let us say blinkLED and I click ok.

(Refer Slide Time: 13:47)



Once I do this you double click on this name, and you see some sample code. We will just cut out this sample code and we will be writing our code in here.

(Refer Slide Time: 14:17)

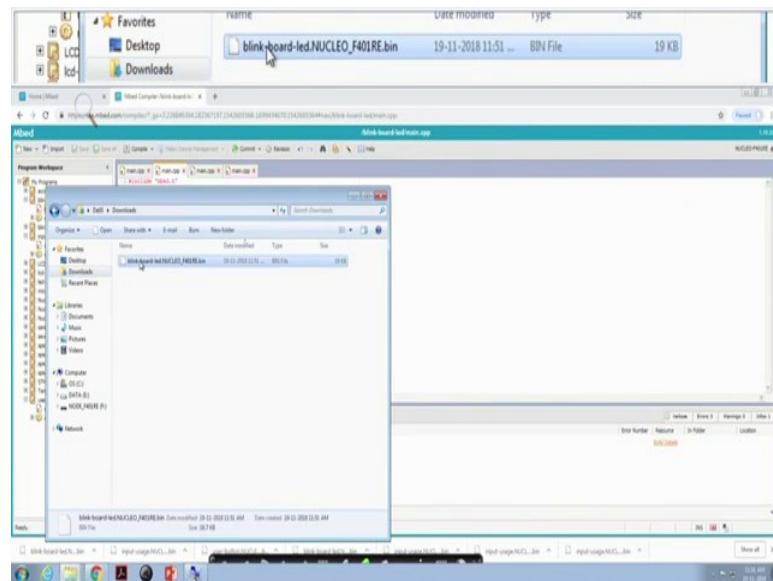


I click here; this is main.cpp as I told you. I will save this. This is #include mbed.h -- this is the header file that you need to include. Then the next thing that you are doing is DigitalOut myLED (LED3), this is one of the output pins. And the name of the LED in that STM board is LED3. The program is continuously running.

So, this is a simple code that I have written. We will be looking into more details in next slide.

Now, I will use this compile button to compile it, now the code is getting compiled you can see and I have told you that, it will get saved in Download folder.

(Refer Slide Time: 16:16)



You copy it or you can cut it from here, and you can see in the left side is the board which is mounted on the PC; you go here and you paste it.

If we do something like this the on-board led will start glowing.

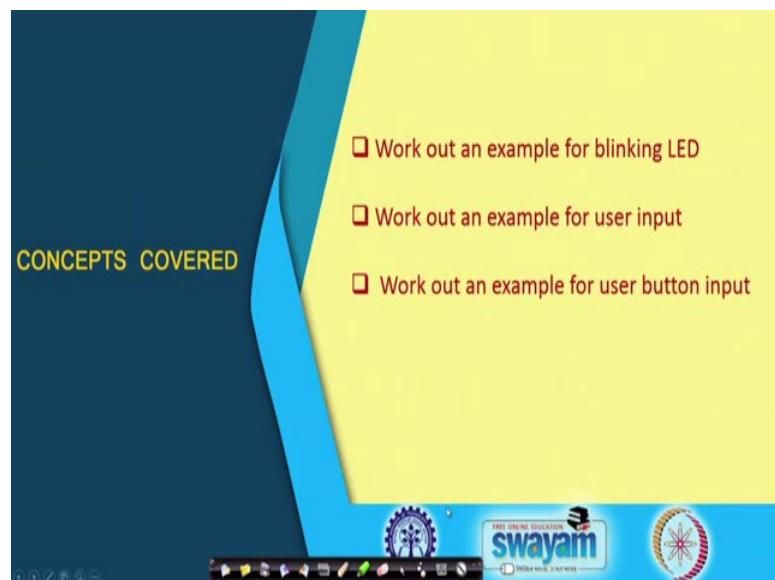
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 20
Interfacing with STM32F401 Board

In this lecture we will be showing you how we can Interface with STM32F401 board. I will be showing you three example programs in which I will be using the onboard LED, the onboard switch, and I will also connect an external switch to show the input/output usage of various pins.

(Refer Slide Time: 01:03)



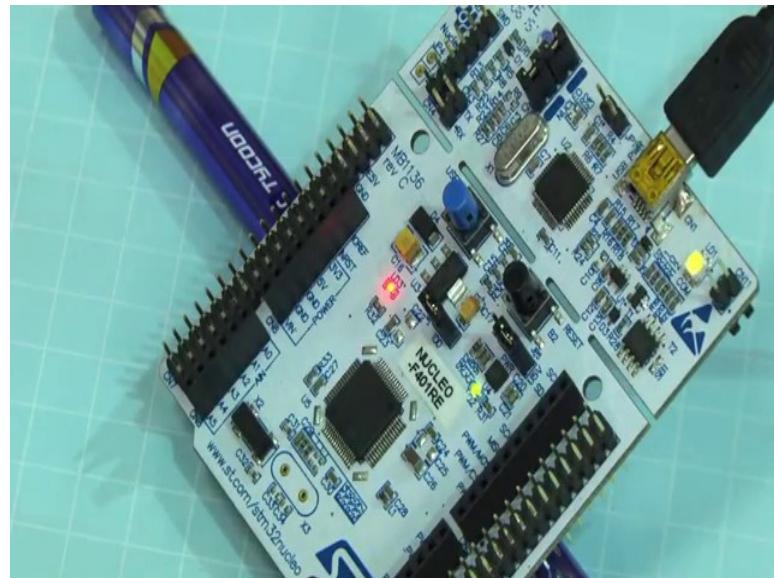
(Refer Slide Time: 01:35)

Introduction

- Three example programs shall be demonstrated on the STM32F401 development board.
- Here, we do not interface any external devices.
- We use the I/O facilities available on-board (like switches and LED), and simple external input.
- The examples:
 - Blink a LED on and off with specified on and off durations.
 - Depending on a user input on a port line, turn on or off a LED.
 - Use the on-board switch to turn on or off a LED.

The 3 example programs shall be demonstrated on this STM32F401 development board.

(Refer Slide Time: 01:48)



I wanted to show you the board. You can see this is the onboard LED. There is one more onboard LED, these are the user button and the reset button, and this is a red and green LED, when we dump the program this LED will be glowing. And, these are the input output digital pins that can be configured either as an input pin or as an output pin.

The examples go like this. The onboard LED will blink for certain duration, and it will be off for certain duration. This is the first one. The next one is, depending on the user

input on the port line the LED will glow. Let us say the LED will be glowing and when I press the switch, it will be off or the other way round. The LED will be off and when I press the switch, the LED will be on.

(Refer Slide Time: 04:00)

Example 1: Blinking_LED.c

```
#include "mbed.h"
DigitalOut myled(LED3);
int main() {
    while(1) {
        myled = 1;      // LED is ON
        wait(0.2);    // for 200 ms
        myled = 0;      // LED is OFF
        wait(0.1);    // for 100 ms
    }
    return 0;
}
```

Let us move on. This is the first program. Let us see how this program actually works. The first one is `#include mbed.h` that is mandatory for any program, this is the header file. The next one is `DigitalOut`.

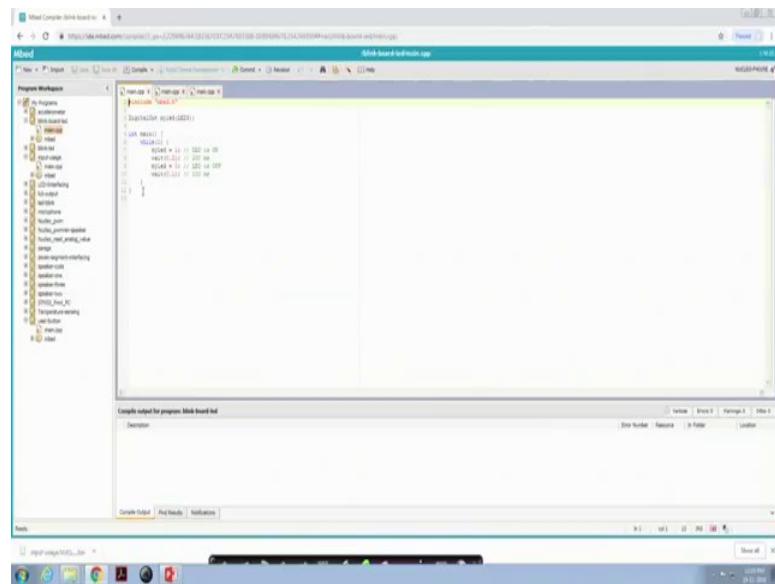
We are using the onboard LED `LED3`, we are making that LED as digital output and we are writing the name `myled`. There are many other ports as well, you can make them as `DigitalOut` also.

The main function is the one from where execution starts. Here in `main`, there is a `while` loop. You see in an embedded application what generally is done, it will read certain input and depending on that input, it will perform something and that will go on happening.

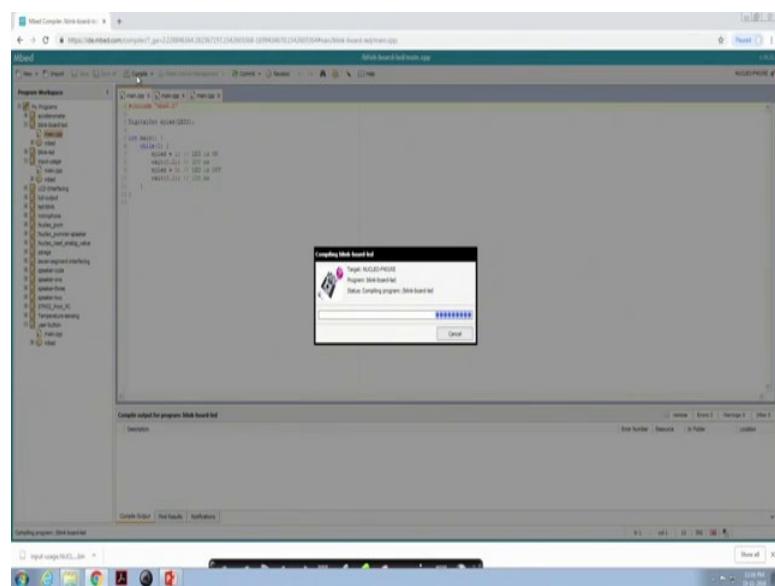
That is why I want this entire set of things to happen all the time, that is why it is `while(1)`. The process repeats in an infinite loop.

Now I will be showing you this particular code in the board. You can see I have already connected the USB port to my PC and now I will use the online mbed compiler, where I will be dumping the same code I have shown you.

(Refer Slide Time: 07:43)

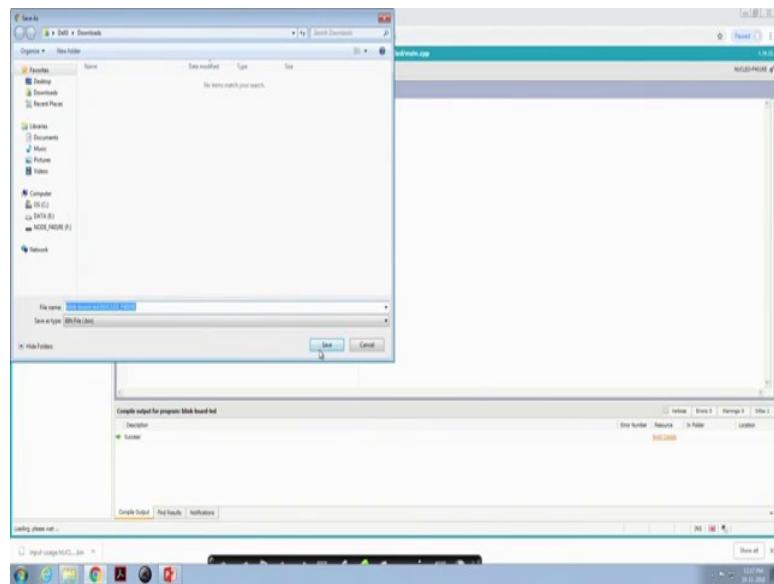


(Refer Slide Time: 07:51)

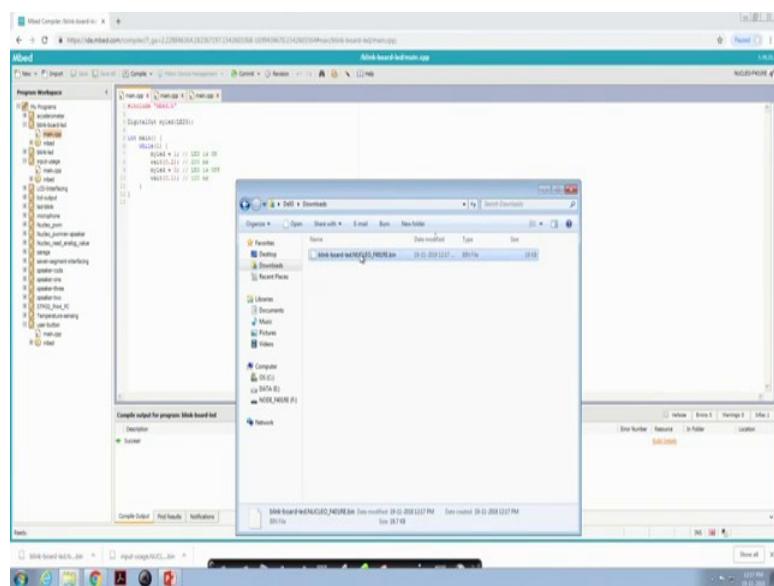


So, what I am doing here? I am compiling the code, the code I have already written, I am extracting it, I am saving it in the Download folder.

(Refer Slide Time: 08:00)

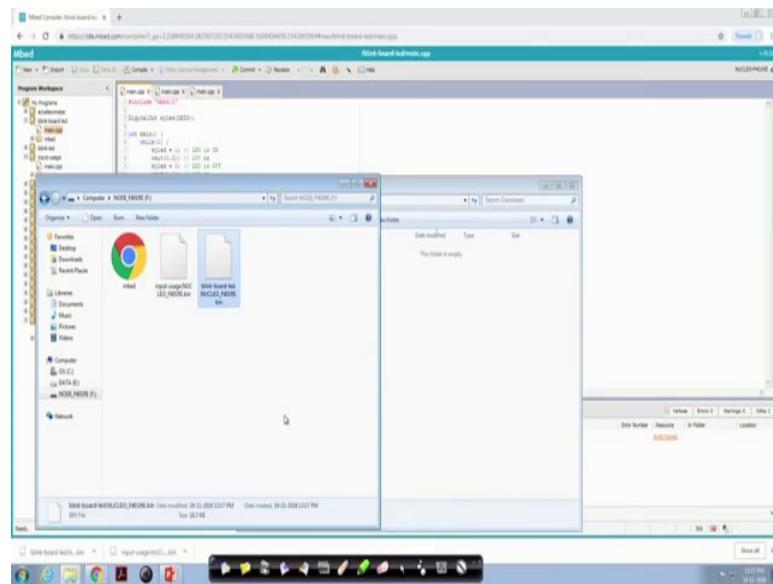


(Refer Slide Time: 08:06)



I open the Download folder, I take it, I copy it and paste it on this particular board.

(Refer Slide Time: 08:14)

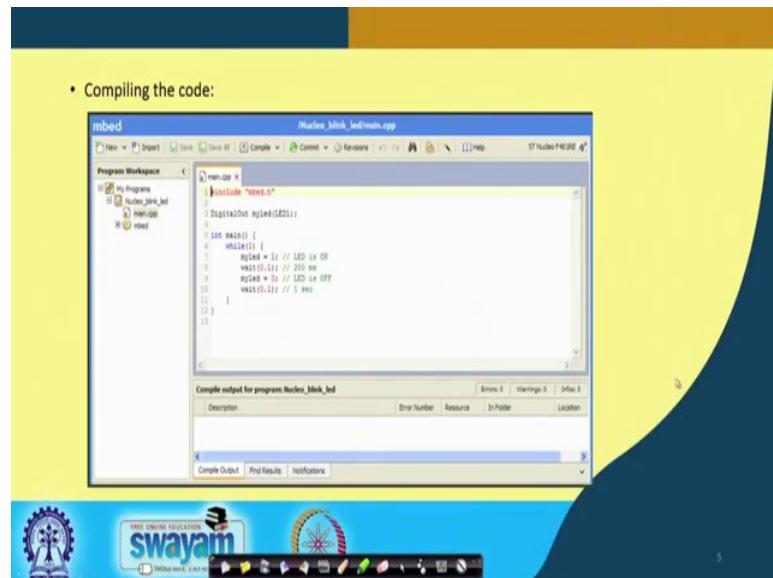


Now, you see after pasting what is happening? This is the LED3, which is actually blinking. I will just change the delay, you concentrate looking here and I will just change the delay; let us say the LED will glow for 2 seconds and will be off for 1 second.

I again recompile the code, I save it and then I copy it to the nucleo board, I dump the code into this board. Now you see the LED is glowing for 2 seconds and off for 1, it is glowing for 2 seconds again off for 1. So, that blinking speed has changed.

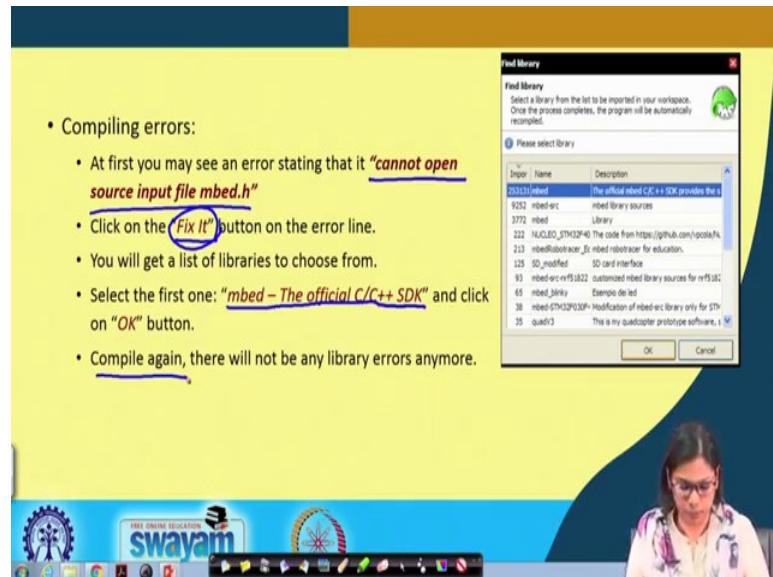
This is a sample example I have shown, how do you use the onboard LED.

(Refer Slide Time: 10:23)



So, coming back to the program, this was the program and you can change it accordingly.

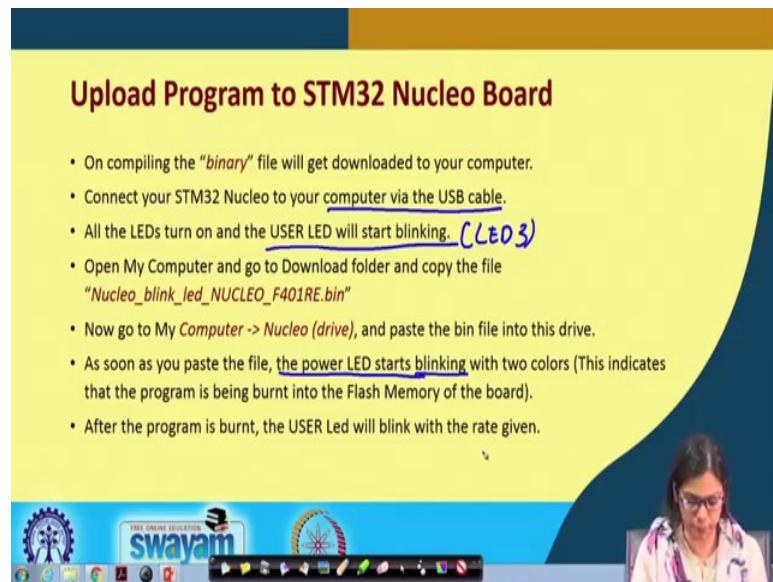
(Refer Slide Time: 10:32)



This is one of the errors that you might get when you do it for the first time. You can see this error, the error is like cannot open source input file embed.h. What you need to do is that, you need to click on FixIt. You will be seeing this button in the down corner, you click on that and you will get a list of libraries to choose from. And in that case you have to select the first one, which is mbed the official C C++ SDK.

You must remember that when you are you are doing it for the first time, you might get all these errors.

(Refer Slide Time: 12:00)



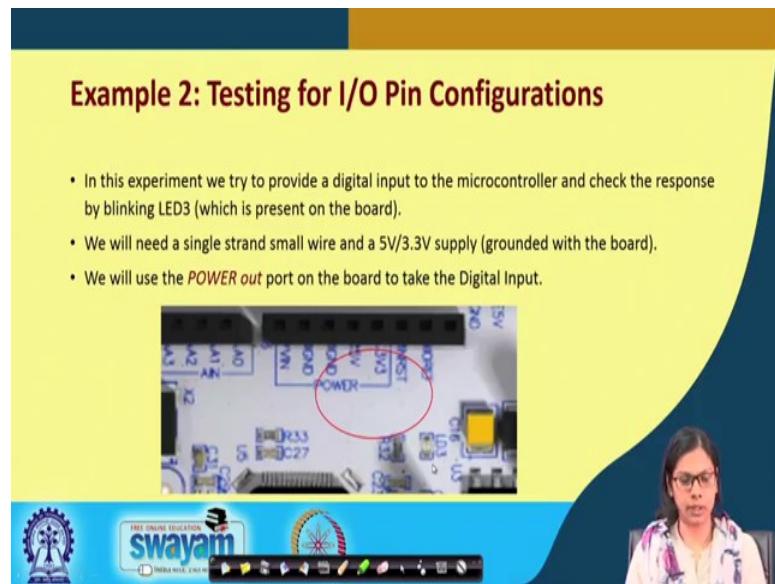
Upload Program to STM32 Nucleo Board

- On compiling the "binary" file will get downloaded to your computer.
- Connect your STM32 Nucleo to your computer via the USB cable.
- All the LEDs turn on and the USER LED will start blinking. (Lto3)
- Open My Computer and go to Download folder and copy the file "Nucleo_blink_led_NUCLEO_F401RE.bin"
- Now go to My Computer -> Nucleo (drive), and paste the bin file into this drive.
- As soon as you paste the file, the power LED starts blinking with two colors (This indicates that the program is being burnt into the Flash Memory of the board).
- After the program is burnt, the USER Led will blink with the rate given.

Now, how do you upload the program? On compiling the binary file this will get downloaded to the computer. Then we need to connect the STM Nucleo board to the computer or laptop and via this USB cable, and all the LEDs turn on, the user LED will start blinking.

This is the program and now you go to MyComputer Nucleo drive and paste the bin file into this drive, which I have already shown you. As soon as you paste that file, you will see that the power LED starts blinking. I will show you for the next program if you have not noticed it, then I will be showing you for the next program and with 2 colors. This indicates basically that the program is being burnt into the flash memory. And after the program is burnt the user LED will blink with the given rate.

(Refer Slide Time: 13:37)



Example 2: Testing for I/O Pin Configurations

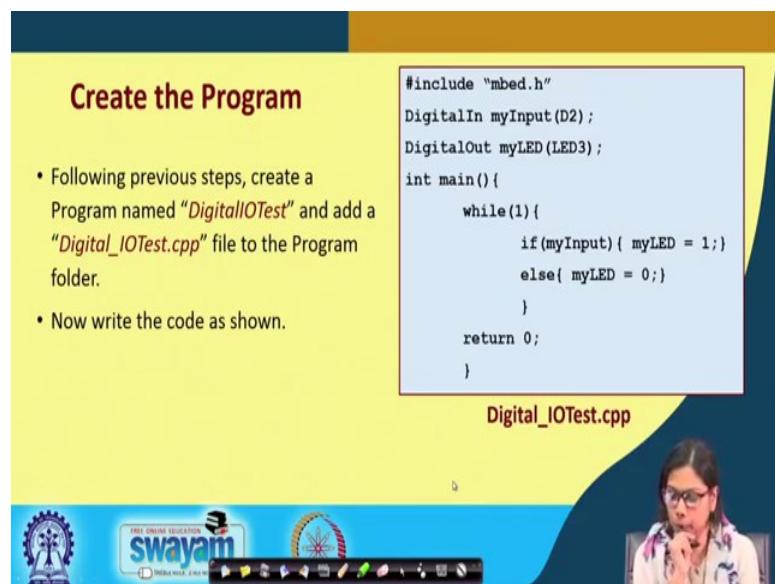
- In this experiment we try to provide a digital input to the microcontroller and check the response by blinking LED3 (which is present on the board).
- We will need a single strand small wire and a 5V/3.3V supply (grounded with the board).
- We will use the **POWER out** port on the board to take the Digital Input.

POWER

Now, I will move on with example 2. In this experiment we try to provide a digital input to the microcontroller. We will be using a switch, a matrix switch where we will be using one individual switch and check the response by blinking again the same LED3, which is present on board. We are not using an external LED for now.

We will need a single strand of small wire and you connect with 5V supply and ground with the board. From the matrix switch I will take one individual switch to make that input to one of the input ports of that board.

(Refer Slide Time: 14:50)



Create the Program

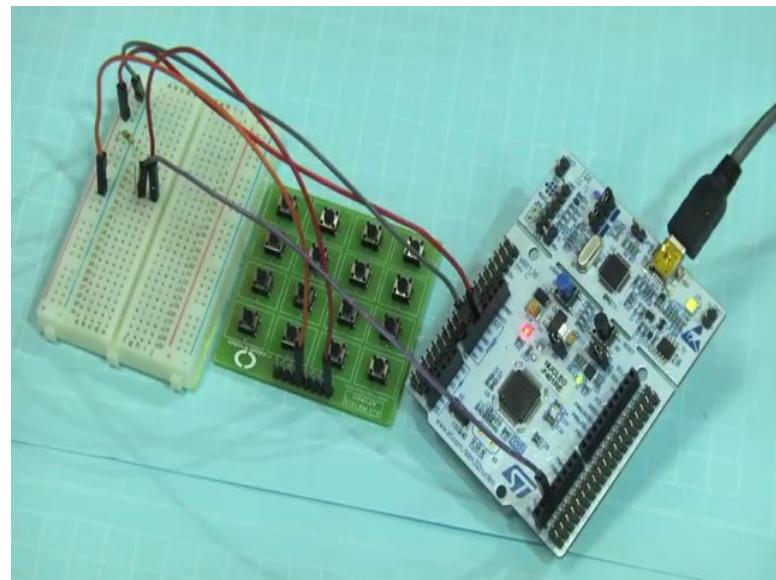
- Following previous steps, create a Program named "**DigitalIOTest**" and add a **"Digital_IOTest.cpp"** file to the Program folder.
- Now write the code as shown.

```
#include "mbed.h"
DigitalIn myInput(D2);
DigitalOut myLED(LED3);
int main() {
    while(1){
        if(myInput){ myLED = 1;}
        else{ myLED = 0;}
    }
    return 0;
}
```

Digital_IOTest.cpp

Here we have to do DigitalIn to specify the input pinut. D2 port is made the digital input port and digital output will be the same LED3, on board LED. The program is self explanatory.

(Refer Slide Time: 16:31)

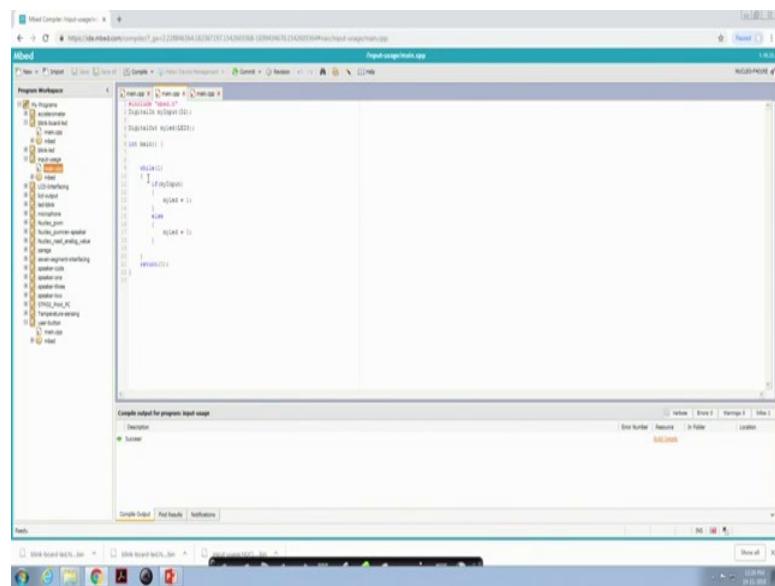


Now, we will see how the code works. This is my switch. Let me tell you what I have done. If you see this is row 1, this is column 1.

One end of the switch is connected through resistance to Vcc, another end of the port is connected to ground. And from this common point we are taking input to port D2.

So, what value it will receive initially? It will receive close to 5V.

(Refer Slide Time: 19:36)

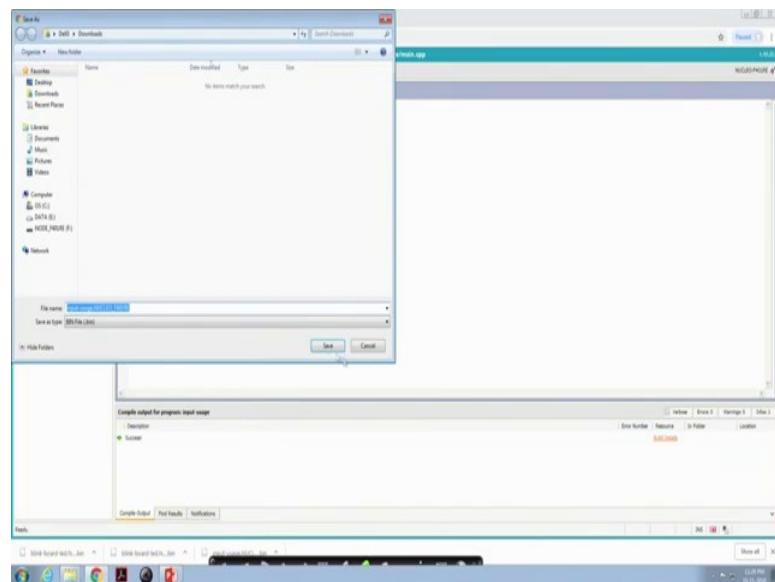


The screenshot shows a software interface with a code editor and a terminal window. The code editor displays C code for a program named 'input_usage'. The terminal window shows the output of a compilation command: 'Compiling for program input_usage' and 'Success'. The status bar at the bottom indicates the date and time as '10-12-2018 10:22 PM'.

```
Compiling for program input_usage
Success
10-12-2018 10:22 PM
```

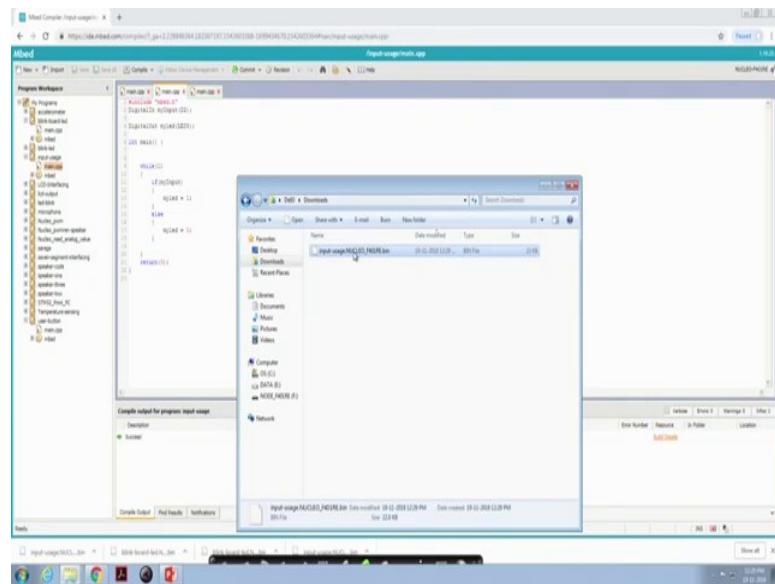
This is my code the code just now which I have discussed with you.

(Refer Slide Time: 19:53)



I will compile it, I will download it, and I will paste it.

(Refer Slide Time: 19:59)



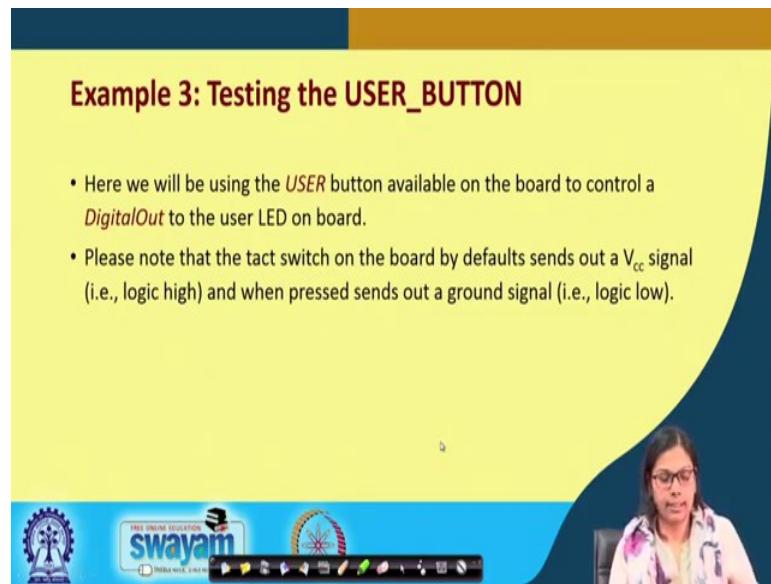
While pasting it, I will show you this board again, because then you can see that this particular LED will be blinking. The LED turns on and off depending on the press of the key.

(Refer Slide Time: 21:36)

A screenshot of a presentation slide. The title 'Typical Usage of Digital I/O' is at the top. The content is a bulleted list: 'Digital I/O can be used for interfacing external devices some of which we will see in the course of the lectures and demonstration.' followed by a list of applications: 'Switching based on activity (like interrupts, etc.).', 'Controlling electro-mechanical devices.', 'Taking sensor inputs.', 'Communicating other similar or dissimilar devices', and 'many more..'. At the bottom of the slide, there is a watermark for 'FREE ONLINE EDUCATION SWAYAM' with the Indian Space Research Organization (ISRO) logo. A woman is visible in the background, likely the speaker.

So, this digital IO can be used for interfacing external devices, some of which we will see in the course of the lectures and the demonstrations.

(Refer Slide Time: 22:23)



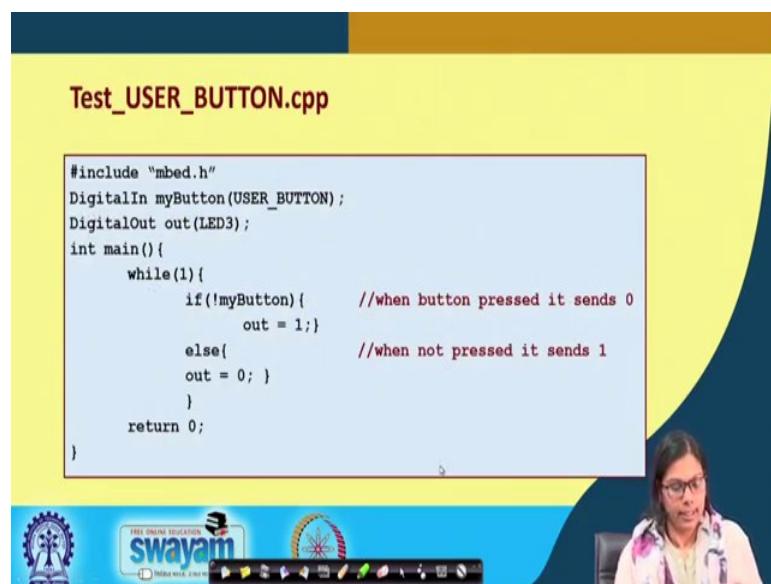
Example 3: Testing the USER_BUTTON

- Here we will be using the *USER* button available on the board to control a *DigitalOut* to the user LED on board.
- Please note that the tact switch on the board by default sends out a V_{cc} signal (i.e., logic high) and when pressed sends out a ground signal (i.e., logic low).

The last example is testing the user button. We have also seen that there is an onboard user button in this particular board. Here we will be using that user button to control a user LED on board.

You must note that this tact switch which is on board, by default sends out a V_{cc} signal. So, when it is not pressed, it will send out a 1 signal, and when pressed it sends out a ground signal; that is logic 0.

(Refer Slide Time: 23:35)



Test_USER_BUTTON.cpp

```
#include "mbed.h"
DigitalIn myButton(USER_BUTTON);
DigitalOut out(LED3);
int main(){
    while(1){
        if(!myButton){      //when button pressed it sends 0
            out = 1;
        }
        else{              //when not pressed it sends 1
            out = 0;
        }
    }
    return 0;
}
```

Now, let us first see the program, which is self explanatory.

Now, this is the user button, the LED is not glowing but when it is pressed it is glowing. Again I press, it is glowing, again I release, it is not glowing.

So, these are the few experiments I have shown you in this particular lecture.

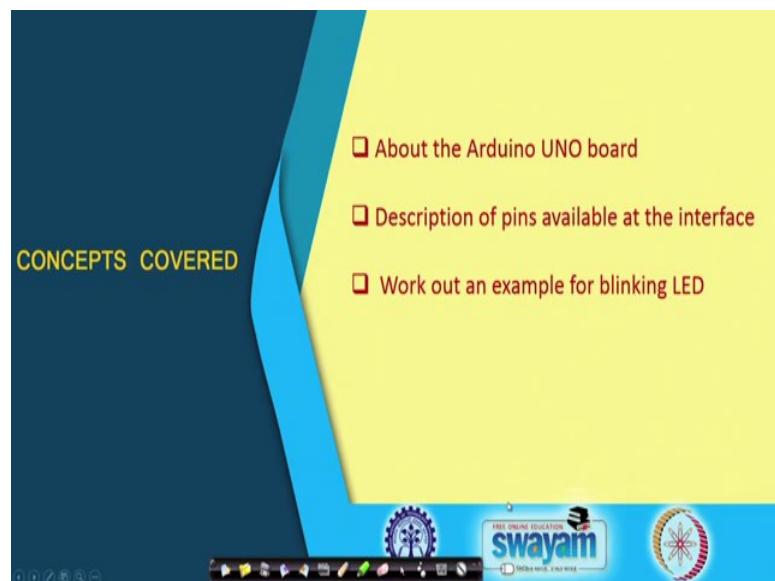
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 21
Interfacing with Arduino Uno

Welcome to lecture 21. In this lecture I will be showing you how we can Interface with Arduino UNO, one of the very popular boards. In the previous lectures I have specifically shown you how we will be using STM board and here I will take you through the steps that are required to use Arduino UNO board, which is again fairly very straightforward. I will just show you some of the features of this board first and then what are the steps that are required to actually interface with Arduino.

(Refer Slide Time: 01:08)



As I told you, I will be talking about Arduino UNO board description about the pins that are available and also work out an example for blinking LED. Although in this experiment I will not interface the LED, but in subsequent slides you will find how do we interface it with the board. I will be specifically showing you the steps that are required to use Arduino UNO board.

(Refer Slide Time: 01:41)

What is Arduino?

- Arduino is an open source computer hardware and software company.
- Arduino designs and manufactures embedded system boards and accessories.
- One such popular microcontroller board is the Arduino Uno.
- The Arduino IDE can be downloaded from:
<https://www.arduino.cc/en/Main/Software>

So, what is Arduino? It is an open source computer hardware and software company. It designs and manufactures various embedded system boards and accessories. And, one of the very popular boards is Arduino UNO, which is used by the student community across the world to design various small embedded system projects. This Arduino IDE can be downloaded from this particular site, you can directly go there and download from that particular site.

(Refer Slide Time: 02:26)

Requirements to Begin

- Arduino development board (Arduino UNO)
- USB connector
- Development environment (Arduino IDE).

R/L

So, what are the requirements? For STM32 board we have already seen what are the requirements that are there. Now, for Arduino board it is very similar. It depends on how do we interface, but the easiest way is like you connect through this USB connector along with your PC. This jack is connected here and this jack is will be connected to your PC or laptop. The development environment used is Arduino IDE, which looks like this, which can be downloaded from the website I have already discussed.

(Refer Slide Time: 03:24)

Arduino UNO

- The Arduino UNO is a widely used open-source micro-controller board based on the ATmega328P microcontroller and developed by Arduino.
- It consists of:
 - 14 Digital pins and 6 Analog pins.
 - It is programmable with the Arduino IDE via a type B USB cable.
 - It can be powered by a USB cable or by an external 9 volt battery, though it accepts voltages between 7 and 20 volts.

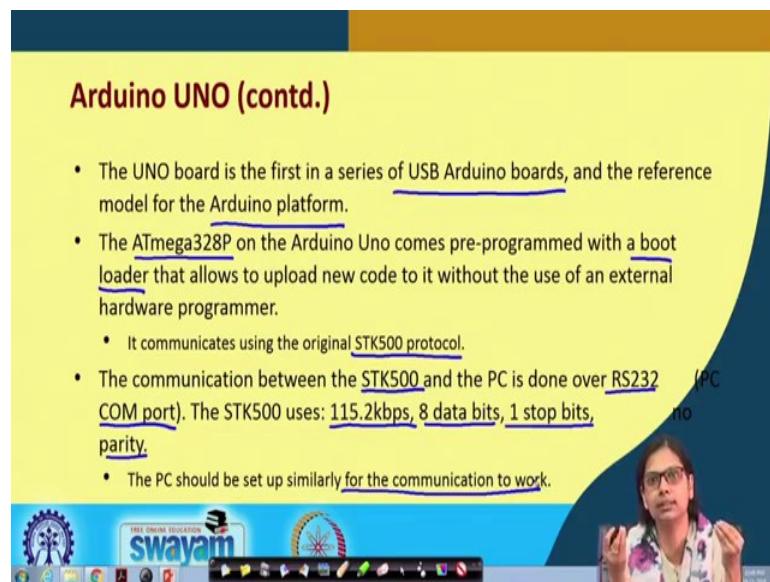
SWAYAM

A photograph of an Arduino Uno microcontroller board is shown on the right side of the slide. The board is blue with various electronic components and pins. The Arduino logo is visible on the board. In the bottom right corner, a person wearing glasses and a white shirt is partially visible, looking towards the camera.

This is the Arduino UNO, which is widely used open source microcontroller board, based on ATmega328P microcontroller and is developed by Arduino. 14 digital input output pins are available, which are all Arduino compatible pins.

So, there are 14 digital pins and 6 analog pins, and they are programmable with Arduino IDE via this typeB USB cable which I have already shown. It can be powered by USB cable or by an external 9 volt battery. Let us say you have already dumped the program and you want it to run without the support of any PC anywhere. So, how do you power it? You can do that using this external battery.

(Refer Slide Time: 04:40)



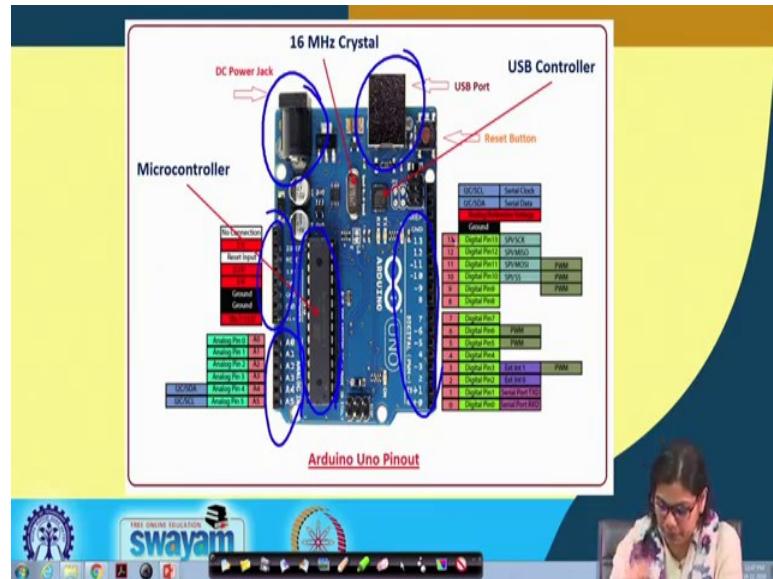
Arduino UNO (contd.)

- The UNO board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform.
- The ATmega328P on the Arduino Uno comes pre-programmed with a boot loader that allows to upload new code to it without the use of an external hardware programmer.
 - It communicates using the original STK500 protocol.
- The communication between the STK500 and the PC is done over RS232 (PC COM port). The STK500 uses: 115.2kbps, 8 data bits, 1 stop bits, no parity.
 - The PC should be set up similarly for the communication to work.

This board is the first in the series of USB Arduino boards, in earlier boards you need to dump it through some other means not through this USB's. But, when USB compatible boards came, Arduino UNO is the first of those kinds of board and the reference model for the Arduino platform. The ATmega328P on the Arduino UNO comes pre-programmed with a boot loader that allows to upload new code to it without the use of any external hardware programmer. So, we do not need to have any external hardware programmer; rather if when we connect this inbuilt boot loader is already there.

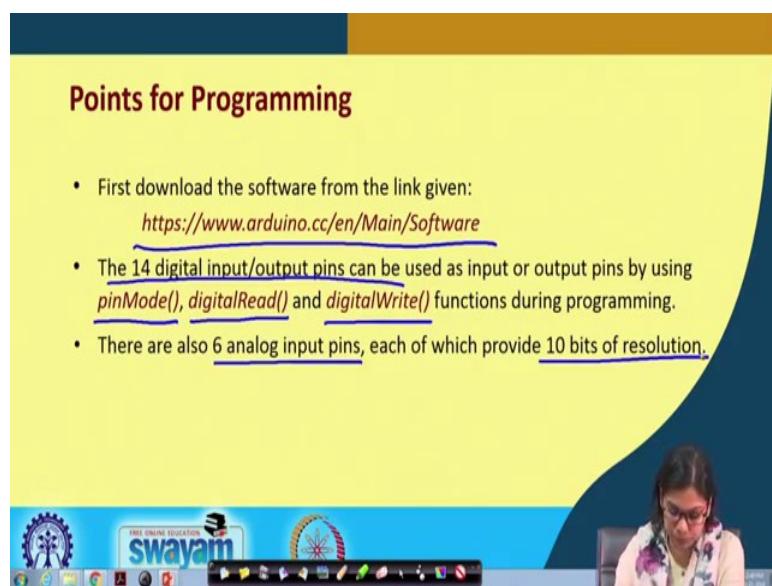
It communicates using the original STK500 protocol, this is a protocol that is used for this communication. The communication between this and the PC is done over RS232, and this STK500 uses 115.2 kbps, 8 bit of data, 1 stop bit and parity. The PC should be set up similar for the communication to work. When 2 devices communicate with each other, if one device runs at a higher rate other device does not run at that rate, this device must know at what rate the other device is communicating. So, for that purpose this is required.

(Refer Slide Time: 00:34)



This is the overall board you can see. I have already discussed about this power jack, this is the USB controller, and USB port through which it can be connected. This is the ATmega microcontroller, this is the digital power supply pins, these are the analog pins and these are the digital input output pins. We will be looking into these when we perform experiment with this particular board. We will be specifically looking into the various aspects of these particular pins.

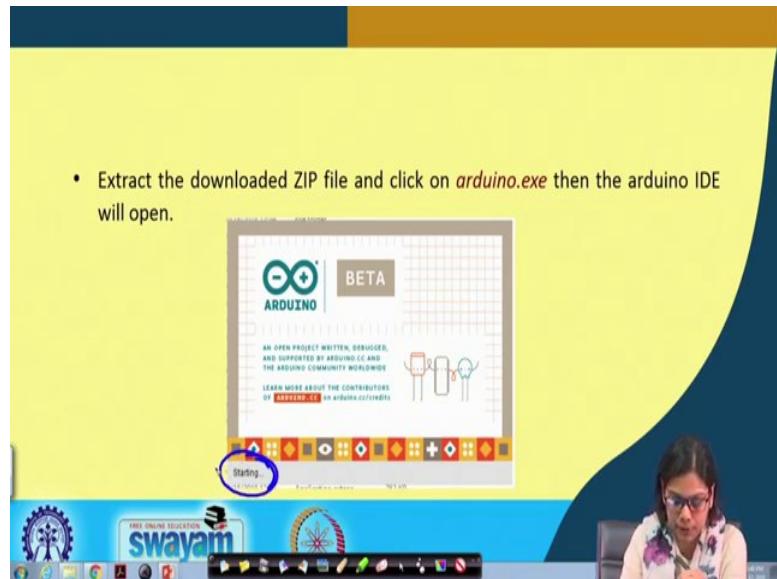
(Refer Slide Time: 07:24)



What are the points that you need to remember for programming using Arduino? The first and foremost thing is that you need to download the software, the IDE. And for using the 14 digital input output pins, we need to know the following functions: one is spin mode, another is digital read for reading the input, digital write to write the value into the port.

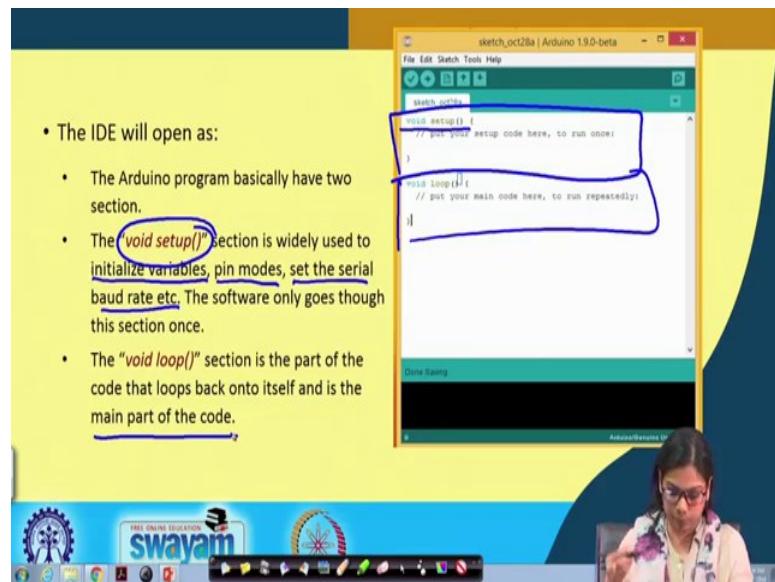
There are also 6 analog input pins, each of which provide 10 bits of resolution. We have already discussed about the resolution and other aspects of analog to digital converter. So, this has 10-bit resolution.

(Refer Slide Time: 08:22)



Once you have downloaded, extract the downloaded zip and click on arduino.exe, then the arduino IDE will open that looks similar to this.

(Refer Slide Time: 08: 48)



When you have already done with installing the IDE which is fairly very straightforward, then when you open it the IDE will open as like this. This is setup phase and this is the loop. In the setup phase, the setup code here that we write is only run once. The setup section is widely used to initialize the variables, mention about the pin modes and set the serial baud rate etc. When we can use other communicating devices like the GSM or Bluetooth, you need to specify that rate here.

The software only goes through this once. The void loop section is the part of the code that loops back onto itself and it is the main part of the code. The code that you write, let us say that, LED will glow for some time. That part of the code will be put in this void loop phase.

(Refer Slide Time: 10:26)

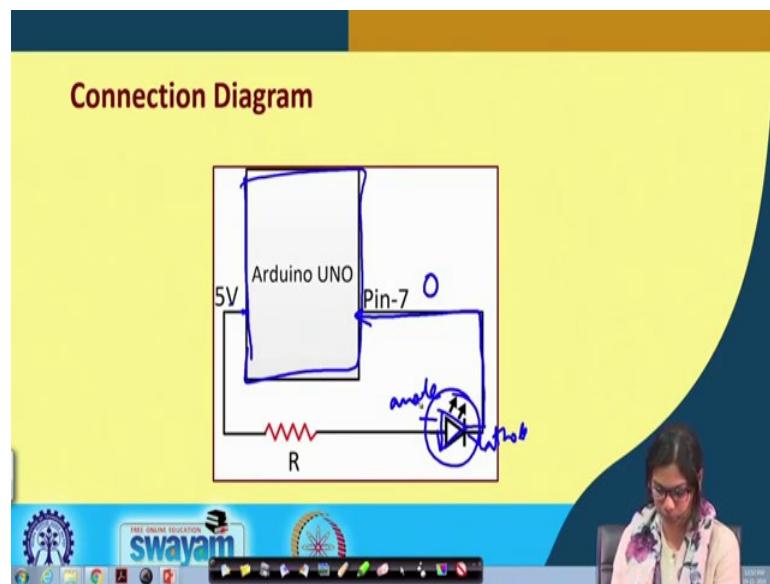
```
void setup() {
    pinMode(7, OUTPUT); // set the digital pin 7 as output
}

void loop() {
    digitalWrite(7, HIGH); // sets the digital pin 7 on
    delay(1000); // wait for 1 second
    digitalWrite(7, LOW); // sets the digital pin 7 off
    delay(1000); // wait for 1 second
}
```

Now, I will take a small example to blink an LED. You have seen in mbed compiler we need to include a header file, we will see what is required here. The main phases that are required here are two, one is the setup phase. Here I am basically doing the initialization, we are setting pin mode, pin 7 of Arduino as output. This is what we are doing using this particular statement.

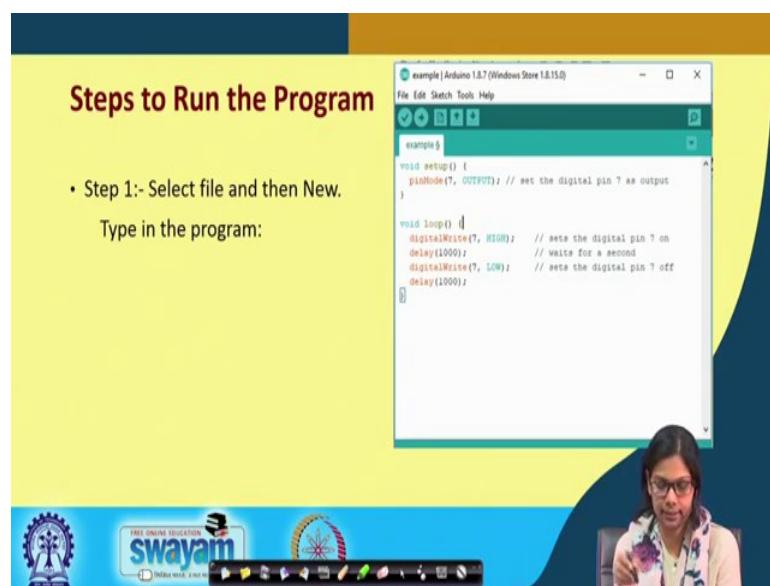
Then comes the void loop. We are using a function called digitalWrite, where the output port 7 we are making high. Then we are giving a delay of one second, here it is specified in millisecond. 1000 milliseconds is 1 second. Then we do a digitalWrite to the same pin 7 to low, and then we delay it for again the same 1000 milliseconds or 1 second, and this goes on in a loop.

(Refer Slide Time: 12:27)



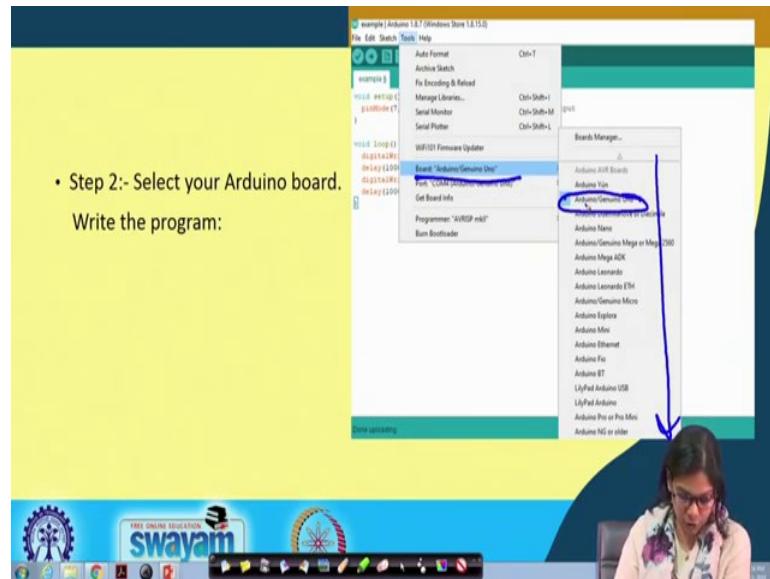
This is a typical pin diagram. We are not using the onboard LED here; rather, we are using an external LED. This is the Arduino UNO board, and this is the 5V supply, and this is the LED. This is the anode and this is the cathode. So, the anode end through a resistance is connected to 5V, and the cathode end is directly connected to pin 7. This is the circuit diagram, when the LED will glow? As this is connected to 5V, when this pin will have a 0 input, then only this LED will glow.

(Refer Slide Time: 13:27)



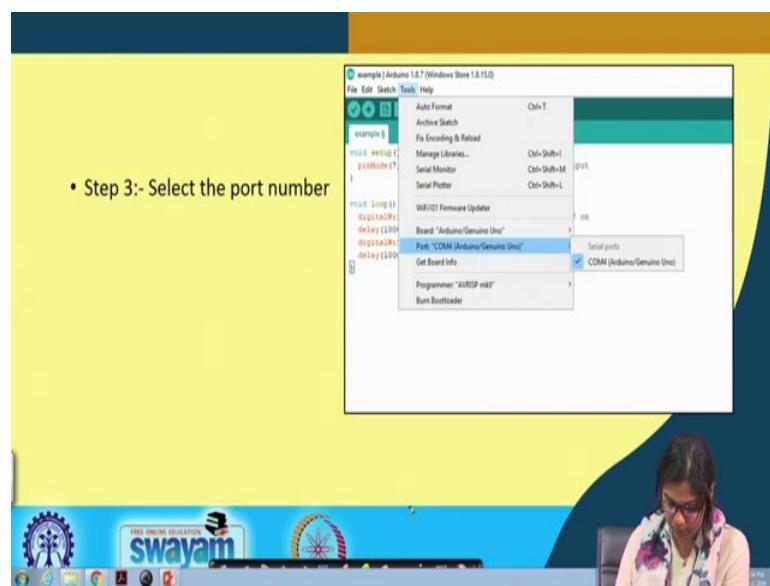
What are the steps that are required here to do the needful? Here we select this pin mode 7 as output, we write the program, and then we select the Arduino board.

(Refer Slide Time: 13:47)



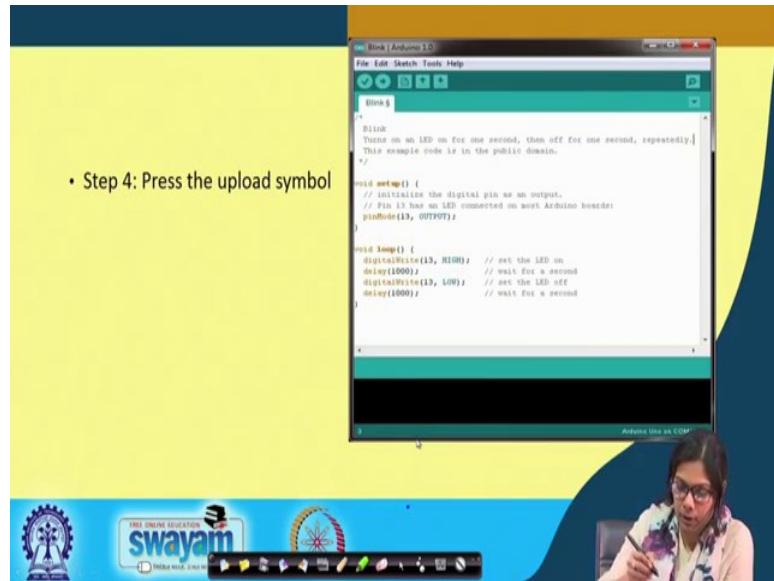
So, specifically you have to use the Arduino UNO board from this particular list.

(Refer Slide Time: 14:09)



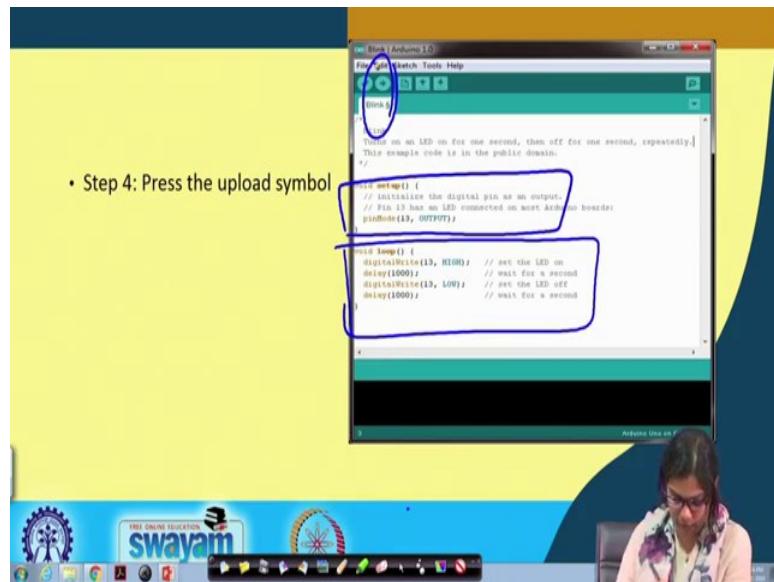
Next select the port number; we can select this COMM for this particular UNO.

(Refer Slide Time: 14:25)



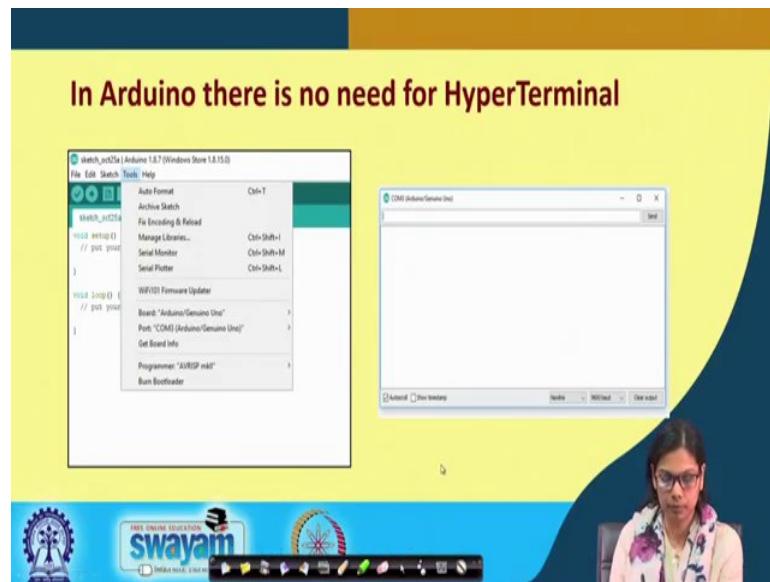
And then we press this upload button to upload it.

(Refer Slide Time: 14:34)



So, this is the setup phase, this is the loop phase and we press on the upload button to upload the program to it.

(Refer Slide Time: 14:54)



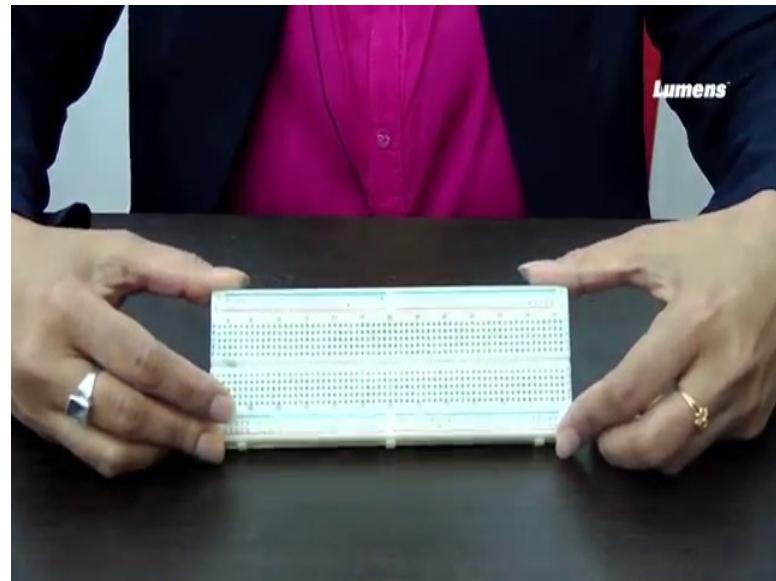
One important thing we have not talked about hyper terminal that we will be discussing in the next week. Arduino has a good feature that it does not require any external hyper terminal, it is in-built there. You go to tools and then you can click on this port and you will get this. Whatever data communication you want you can see it in this port.

We will look into that later, this is one of the feature of Arduino. So, this is all about this Arduino board, how will you interface with Arduino board, you need that ID, you need to write the code, the code goes in 2 phases and finally, you have to upload the code, it gets uploaded into the device and then it is done.

Today I will be showing you now how do we connect an LED with the two kinds of boards that I have already discussed.

Both the boards has got some features, which we have already seen. Now we will be going to the interfacing experiments. I have already discussed how the onboard LED will glow. Now I will be connecting another LED using breadboard with the STM board, as well as with the Arduino board.

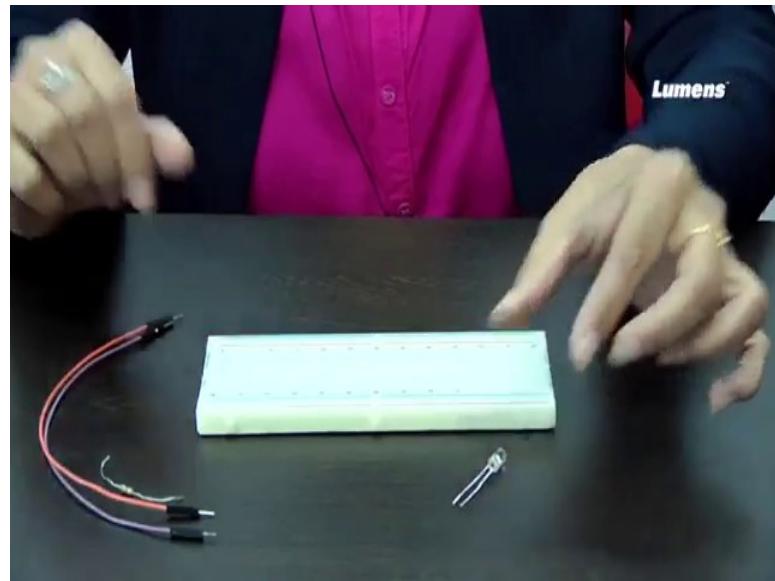
(Refer Slide Time: 17:25)



I will first talk about the breadboard. This is a breadboard through which we can connect various small interfacing devices and we can do small experiments. These lines are horizontally connected. There are 2 lines we can see here, these are vertically connected together.

So, from any point you can put either Vcc or ground, and then you can use and connect with other pins as required. Again one thing you have to remember is that there is a cut here. So, this is connected, but this is not connected with this one. So, this is a separate connection, this is a separate connection. If you want to connect the whole thing you have to connect a wire from here till here. We will be showing the first experiment with LED, using this breadboard.

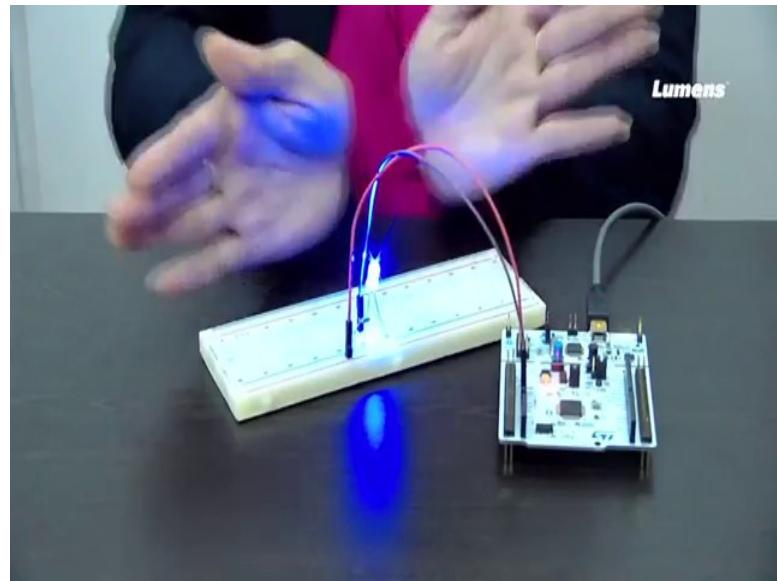
(Refer Slide Time: 19:05)



We also require some jumper wires, and a resistor. Firstly, this is the LED. If you see this LED, you can make out that one of its legs is longer, and another is shorter. I already discussed this in the lecture, but what does it mean, this is the anode and this is the cathode. So, the long leg is anode and the short leg is cathode, and current always flows from anode to cathode.

The first thing that I will be do is I will connect this LED with STM board. And the first thing again I will do is first I will check whether this LED is working fine or not. How will you check that? What you can do, the anode with a resistance you can connect to Vcc and the cathode you can directly connect to ground, and we see whether the LED glows or not.

(Refer Slide Time: 20:34)



So, let us do this experiment with STM board first. This is my anode, this is cathode. So, I put it up in the breadboard like this. Then through a resistance, I connect to this end of the breadboard and from this end that is vertically connected, I will put it to Vcc. So, the Vcc pin here is the fourth pin, so I put it there. And this from this point, that is the anode, I will connect it to ground and I see that the LED is glowing. If I take it out, it is not glowing. Now again I will put it up here, it is again glowing.

So, if you can understand that now you can make out that yes this LED is a working LED. Now what I will do is that, I will connect the anode with Vcc, but now I will connect the ground with digital pin D2. I will be dumping a code into this board. For dumping the code, we have already seen what all are the steps that need to be followed.

Now can you see what is happening? The LED is glowing for 1 second and it is off for 2 second, you see it is glowing for 1 second and it is off for 2 second and this is repeating.

This is how you can do programming with this STM board using a single LED. You can also make different various kinds of experiments with LED, with few more LED's if you connect, you can make a counter that way. So, you can put another LED here and what will be the counter? The counter will be first 00 then 01 then 10 and then 11. This can also repeat forever.

Now I will be doing the same experiment with Arduino board. The code I have already discussed. Now I will be doing the connection first with the Arduino board, this is Arduino UNO board. The same way this connection is straightforward. Now I will connect this part with Vcc and this one with pin D2. Now I will again use the Arduino editor, which we have used for dumping the code.

Now the code is dumped and it is also doing the same thing, it is on for 1 second and it is off for 2 seconds.

So in this experiment we have connected a single LED to both the boards that is STM and with Arduino UNO. You can try out other examples connecting a few more number of LED's and then you can make a LED counter in some way or the other.

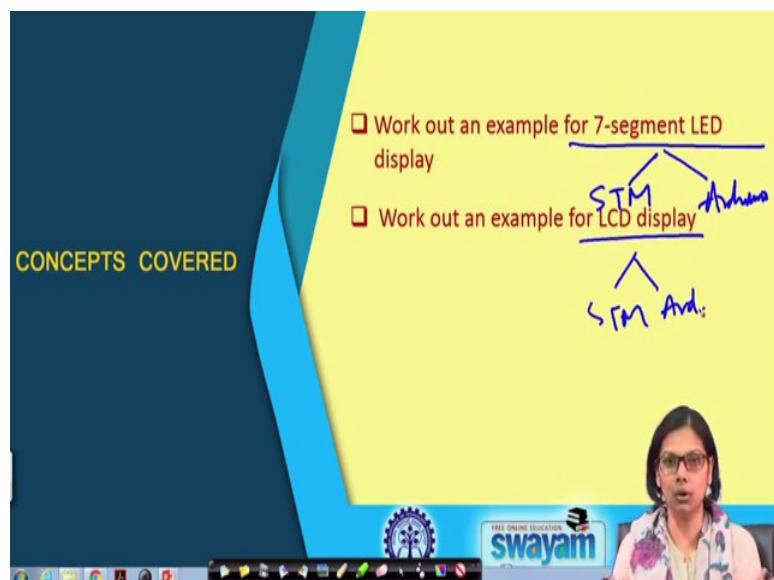
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 22
Interfacing with 7-Segment LED and LCD Displays (Part I)

Welcome to lecture 22, in this lecture we will be interfacing 7-segment LED and LCD displays. We have already discussed about 7 segment display, LCD, how they work. And, we have already discussed about the two boards, Arduino board and STM board. In this particular lecture, I will be showing you the circuit diagram and the code that is required for interfacing the 7 segment display and the LCD.

(Refer Slide Time: 01:11)



I will be showing you a worked out example of a 7 segment display and also I will be showing you a worked out example of a LCD display. And I will be showing for both STM and Arduino.

(Refer Slide Time: 01:50)

Example 1

- Interface a 7-segment display unit to the STM32 board, and display the characters 0 to 9 cyclically with time delay of 1.5 seconds.
 - Connect the segments A to G to the data output pins D2 to D8 on the Arduino connector.



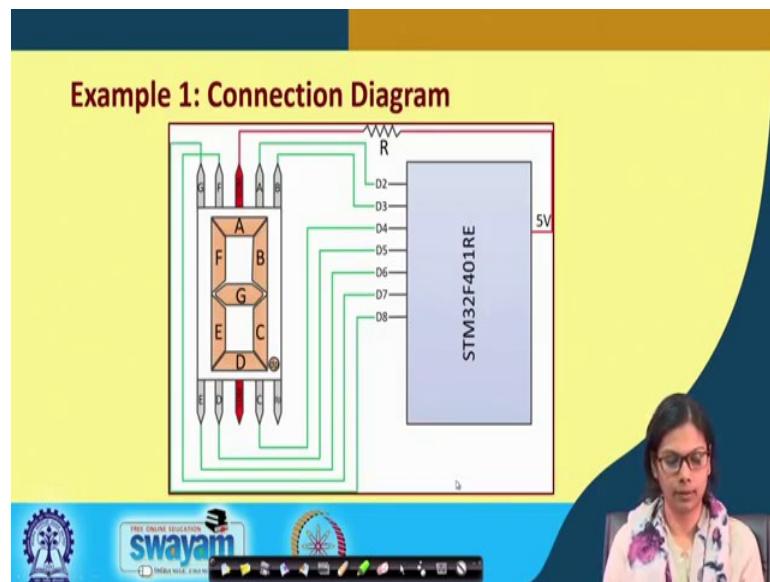
The diagram shows the segments of a 7-segment display (A, B, C, D, E, F, G) and a time delay of 1.5 seconds between segments. The segments are labeled A, B, C, D, E, F, G, and a common cathode terminal. A blue arrow points to the segments, and the text '1.5' is written between the segments to indicate the time delay.



A screenshot of a video conference showing a woman in a pink shirt and glasses. The background is a blue wall with a circular emblem and the word 'swayam'.

The first example that I will be showing is for interfacing 7-segment display unit to the STM32 board, and display the characters from 0 to 9 in a cyclic fashion with a time delay of 1.5 seconds. And I will be connecting this from segment A to G of the data output pins D2 to D8 on the Arduino connector using this STM board.

(Refer Slide Time: 03:10)

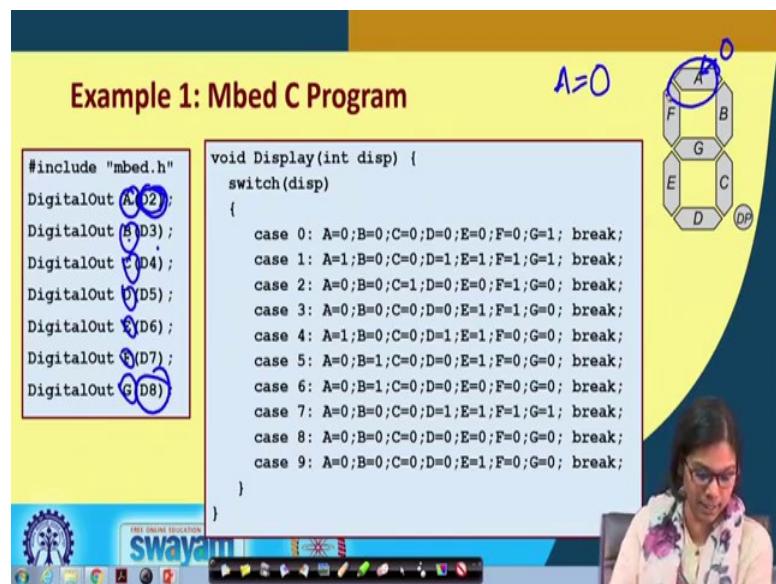


This is the typical circuit diagram. You can see this is the common port. This is a common anode 7 segment display, where the common port is connected through a

resistance to 5V and all other ports, that is A, B, C, D, E, F, and G, are connected to pins D2 to D8 of this STM board.

You can see the connection: A is connected to D2, B is connected to D3, C is connected to D4, D is connected to D5, E is connected to D6, F is connected to D7, and G is connected to D8. This is how we have made the connection, there is no hard and fast rule that you have to make the same connection, but this is how we have connected it.

(Refer Slide Time: 05:06)



Now, let us see the mbed C code. As I have told you, we have used the available pins D2 to D8 starting from the segments that are connected A, B, C, D, E, F and G, and in the same way, we have given these names also for the digital output port.

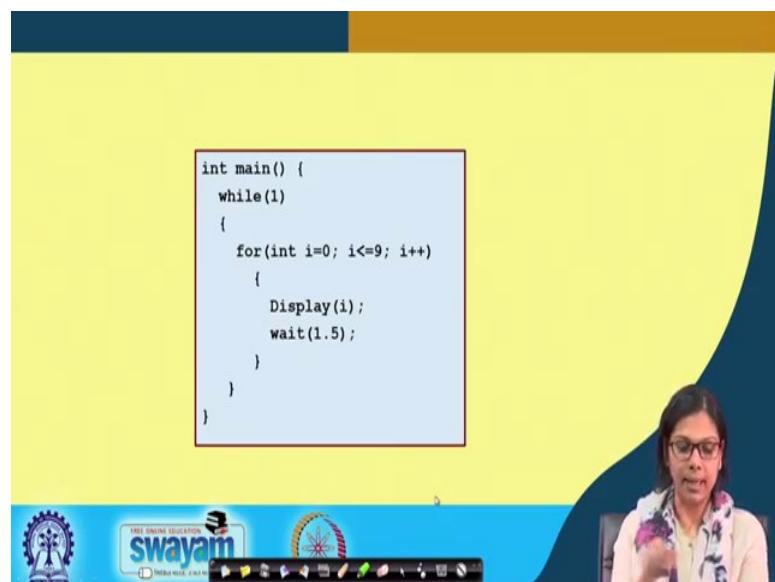
Now, it is a common anode display; when this segment will glow? The common point is connected to Vcc, this segment will glow when we pass a 0 value to this line. So, you have to pass in D2 through this A, A is the signal you have to pass a 0 then only this particular segment will glow. So, to glow it you need to pass a 0, because it is a common anode; if it is common cathode, you have to do the opposite one.

Now, there are 9 cases because as I have told you, I will be displaying from 0 till 9 with a delay. Let us first consider case 0, what is 0? You see how many segments will glow, all the segments except G. So, only the G segments will have value 1 and all other

segments, you want to glow, you have to pass a value 0. Similarly that is what I have done A0, B0, C0, D0, E0, F0, but G is 1.

Similarly, for 1, you only need B and C to glow. So, B and C should have a 0 value all else will have 1 value. So, you can do that for all others. So, this is the code that you have to write.

(Refer Slide Time: 07:47)



In while(1) what we are doing is that in the for loop we vary the value of i from 0 to 9. We are calling Display(i) to display the digit. Then there is a 1.5 second delay.

It goes in a continuous loop.

(Refer Slide Time: 09:33)

The slide has a yellow header with the title 'Example 2' in dark red. The main content area is white. Below the slide, a blue footer bar is visible, featuring the 'swayam' logo and various icons.

- Interface a 7-segment display unit to the Arduino UNO board, and display the characters 0 to 9 cyclically with time delay of 1.5 seconds.
 - Connect the segments A to F to the digital pins 2 to 8 on the Arduino board.

The next example is very similar, but with Arduino board. The previous example that we did is with STM board, this is with Arduino board. You have to again connect from segment here from A till F till G, and digital pins D2 to D8 on the Arduino board.

(Refer Slide Time: 10:18)

The slide has a yellow header with the title 'Example 2: Connection Diagram' in dark red. The main content area is white. Below the slide, a blue footer bar is visible, featuring the 'swayam' logo and various icons.

The diagram shows a 7-segment display connected to an Arduino Uno. The segments are labeled A through G. The connections are as follows:

- Segment A is connected to digital pin 2.
- Segment B is connected to digital pin 3.
- Segment C is connected to digital pin 4.
- Segment D is connected to digital pin 5.
- Segment E is connected to digital pin 6.
- Segment F is connected to digital pin 7.
- Segment G is connected to digital pin 8.
- The common cathode of the display is connected to ground (GND).
- The 5V power pin of the Arduino Uno is connected to the 5V pin of the display.
- The GND pin of the Arduino Uno is connected to the GND pin of the display.
- A resistor (labeled 'R') is connected between the 5V pin of the Arduino Uno and the 5V pin of the display.

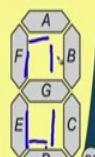
Let us see the circuit diagram again, which is very similar.

(Refer Slide Time: 10:56)

Example 2: Arduino Program

```
void setup() {  
    pinMode(2, OUTPUT);  
    pinMode(3, OUTPUT);  
    pinMode(4, OUTPUT);  
    pinMode(5, OUTPUT);  
    pinMode(6, OUTPUT);  
    pinMode(7, OUTPUT);  
    pinMode(8, OUTPUT);  
}  
  
void loop() {  
    while(1) {  
        for (int i=0; i<=9; i++) {  
            Display(i);  
            delay(1500);  
        }  
    }  
}
```

```
void Display(int disp) {  
    switch(disp)  
    {  
        case 0: digitalWrite(2, LOW);  
        digitalWrite(3, LOW);  
        digitalWrite(4, LOW);  
        digitalWrite(5, LOW);  
        digitalWrite(6, LOW);  
        digitalWrite(7, LOW);  
        digitalWrite(8, HIGH);  
        break;  
        ... similarly for case 1 to 9  
    }  
}
```



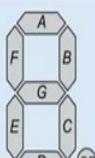
Now, see the code here. In the setup phase, we have to specify the pin mode for each of the pins, we are using pin 2 to pin 8. So, we have to specify the pin mode of each one of these and all are output. And in the loop phase, what we are doing? We are doing something very similar as in the previous code.

(Refer Slide Time: 13:05)

Example 2: Alternate Program

```
void setup() {  
    for(int i=2; i<=8; i++)  
    {  
        pinMode(i, OUTPUT);  
    }  
    pinMode(2, OUTPUT);  
    pinMode(3, OUTPUT);  
}  
  
void on() {  
    for(int i=2; i<=8; i++)  
    {  
        digitalWrite(i, LOW);  
    }  
}
```

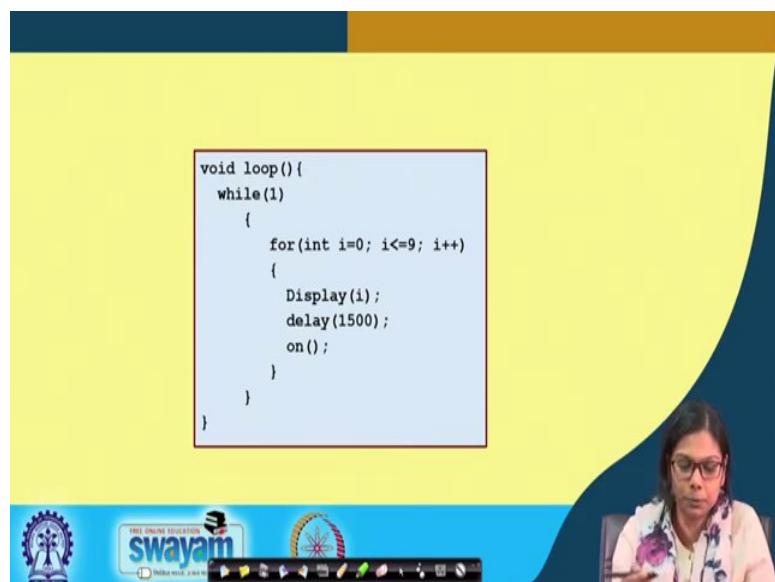
```
void Display(int disp) {  
    if (disp==1 || disp==4) {  
        digitalWrite(2, HIGH);  
    }  
    if (disp==2) {  
        digitalWrite(4, HIGH);  
    }  
    if (disp==5 || disp==6) {  
        digitalWrite(3, HIGH);  
    }  
    if (disp==1 || disp==4 || disp==7) {  
        digitalWrite(5, HIGH);  
    }  
    if (disp==1 || disp==3 || disp==4 || disp==5 ||  
        disp==7 || disp==9) {  
        digitalWrite(6, HIGH);  
    }  
    if (disp==1 || disp==2 || disp==3 || disp==7) {  
        digitalWrite(7, HIGH);  
    }  
    if (disp==0 || disp==1 || disp==7) {  
        digitalWrite(8, HIGH);  
    }  
}
```



An efficient and alternate program could be something like this. for $i=2$; $i \leq 8$; $i++$, we are setting the pin modes. Similarly in the next for loop, we are initializing all the 8 digital output pins to LOW.

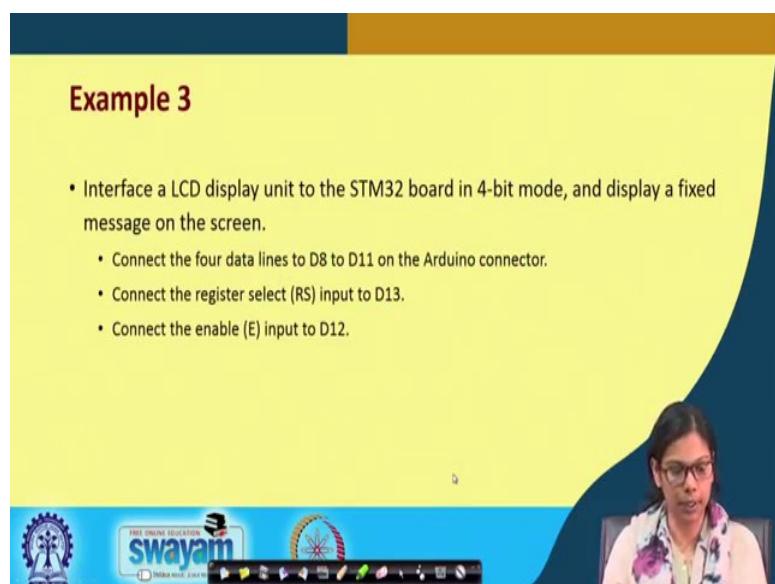
In Display function, depending on the digit, we are writing HIGH to some of the port pins.

(Refer Slide Time: 16:48)



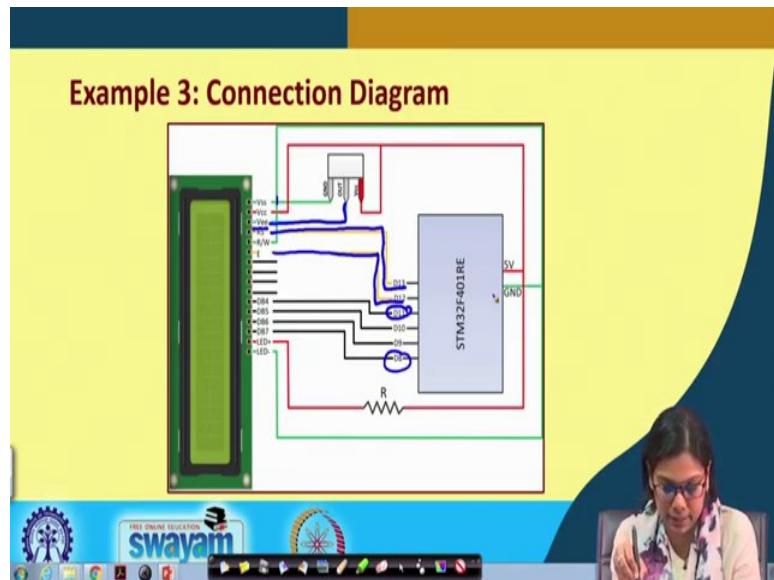
So here, in the loop section we do something very similar to the previous code. The code is self explanatory.

(Refer Slide Time: 17:48)



Now I will interface an LCD unit to the STM32 board in 4-bit mode, and display a fixed message on the LCD. We connect the 4 lines, D8 to D11 on the Arduino connector and we connect this register select to input pin D13, and we connect this enable input to D12, these are the connections that are required for STM32. This is how the connection goes.

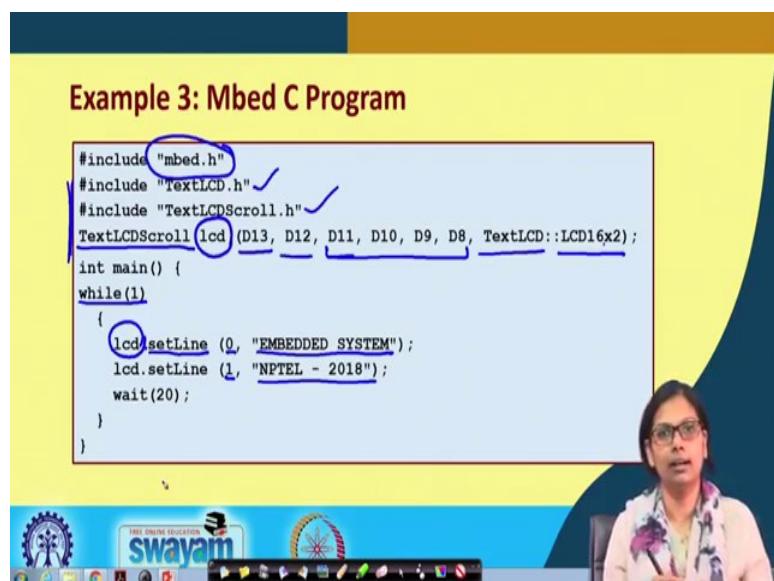
(Refer Slide Time: 18:30)



If you see the LCD, it has got these pins. The connections are also shown.

Read/write is permanently connected to ground, because we are only writing here.

(Refer Slide Time: 20:37)



Now, let us see the mbed program here. Instead of only including mbed.c, we also need to include few other header files. The two header files that are required to be included for LCD interfacing is TextLCD.h and TextLCDScroll.h, and we have to use a function TextLCDScroll. We create an object with the name lcd and we specify the data lines for connection. These are for the data bit line and what kind of LCD we are using? LCD16x2 can display 16 characters in each of two lines.

Next is the main program in the while(1). We are displaying two character strings on two lines of the LCD, 0 means first line and 1 means second line. Then there is a delay of 20 seconds.

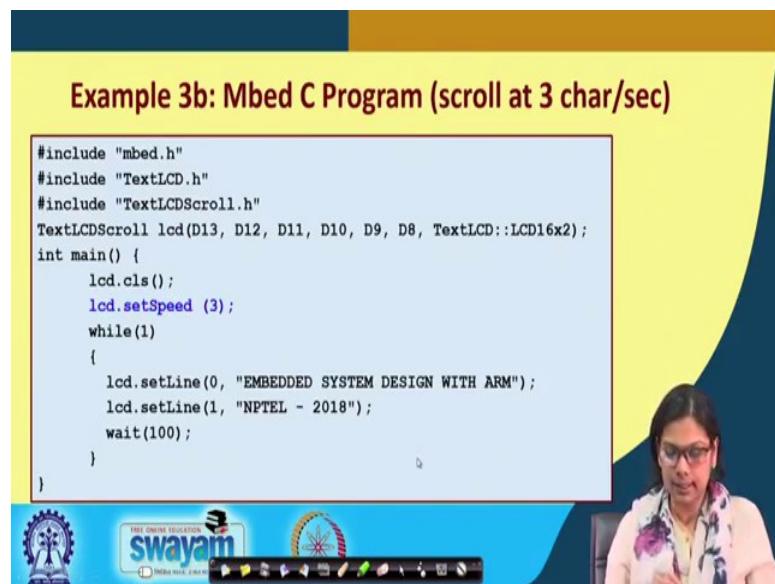
(Refer Slide Time: 22:33)

```
#include "mbed.h"
#include "TextLCD.h"
#include "TextLCDScroll.h"
TextLCDScroll lcd(D13, D12, D11, D10, D9, D8, TextLCD::LCD16x2);
int main() {
    lcd.cls();
    lcd.setSpeed (2);
    while(1)
    {
        lcd.setLine(0, "EMBEDDED SYSTEM DESIGN WITH ARM")
        lcd.setLine(1, "NPTEL - 2018");
        wait(100);
    }
}
```

This is the next program, which is fairly similar, but the only difference being here is that the scrolling speed we are specifying. We are specifying that 2 characters will be scrolled per second if the string does not fit in 16 characters.

If the text is less than 16 character or equal to 16 character, it will be displayed without scrolling. But if it is more than that, the particular text will be scrolling.

(Refer Slide Time: 23:44)



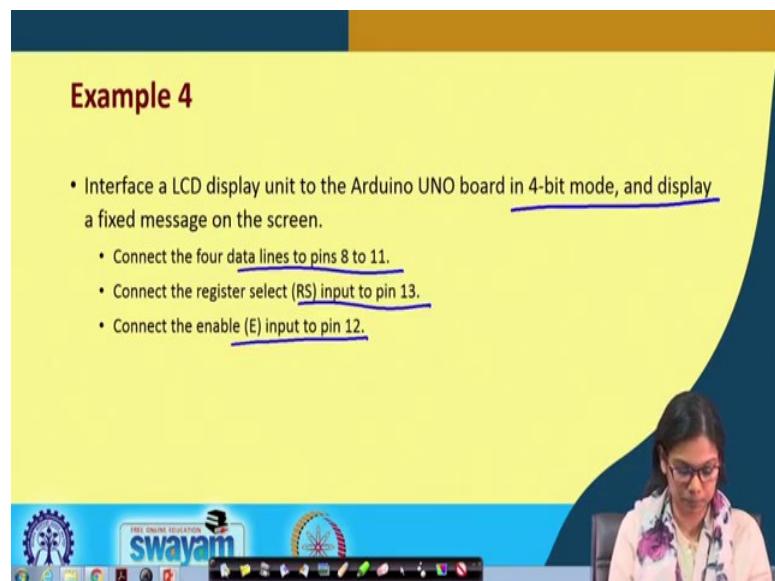
Example 3b: Mbed C Program (scroll at 3 char/sec)

```
#include "mbed.h"
#include "TextLCD.h"
#include "TextLCD8scroll.h"
TextLCD8scroll lcd(D13, D12, D11, D10, D9, D8, TextLCD::LCD16x2);
int main() {
    lcd.cls();
    lcd.setSpeed (3);
    while(1)
    {
        lcd.setLine(0, "EMBEDDED SYSTEM DESIGN WITH ARM");
        lcd.setLine(1, "NPTEL - 2018");
        wait(100);
    }
}
```

The presentation is running on the Swayam platform, as indicated by the logo and interface elements at the bottom of the slide.

This is another program similar program, just that we have made the scroll speed to 3 characters per second.

(Refer Slide Time: 24:07)



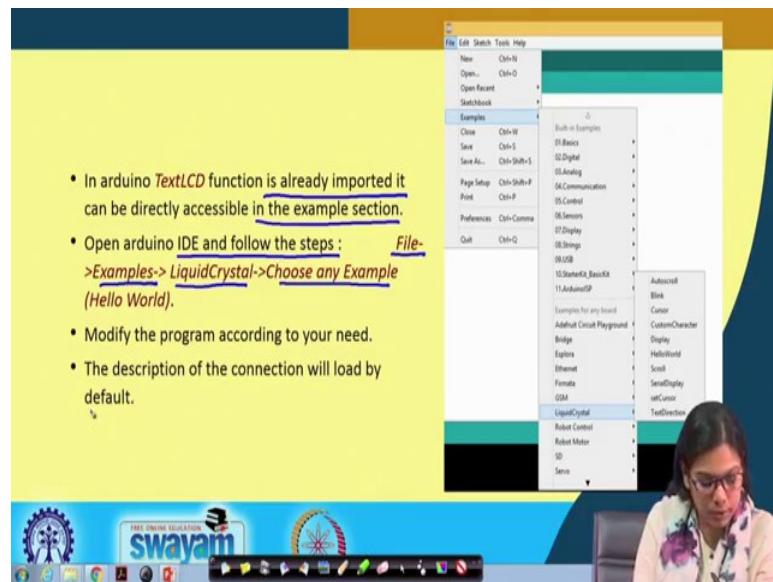
Example 4

- Interface a LCD display unit to the Arduino UNO board in 4-bit mode, and display a fixed message on the screen.
 - Connect the four data lines to pins 8 to 11.
 - Connect the register select (RS) input to pin 13.
 - Connect the enable (E) input to pin 12.

The presentation is running on the Swayam platform, as indicated by the logo and interface elements at the bottom of the slide.

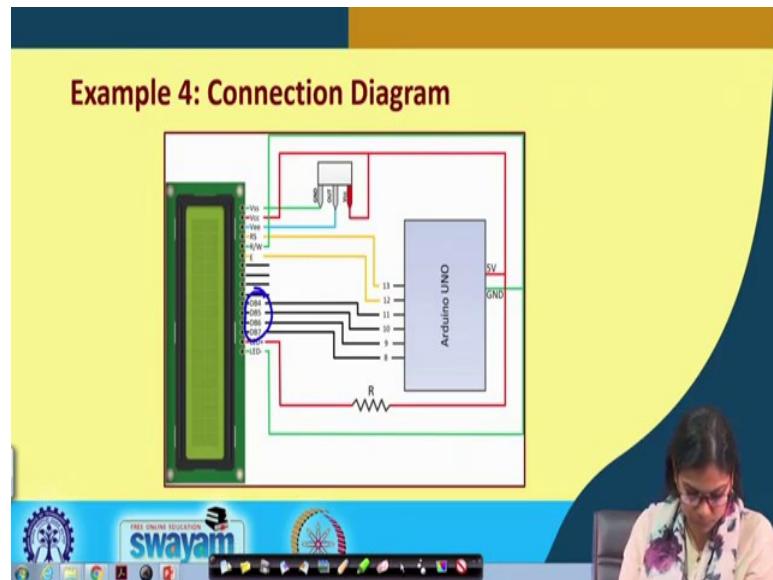
Next I will be showing, how do we interface the same LCD with Arduino UNO board in a 4-bit mode and display a fixed message on the screen. So, here we are not playing around with too many things. The connections are shown.

(Refer Slide Time: 24:49)



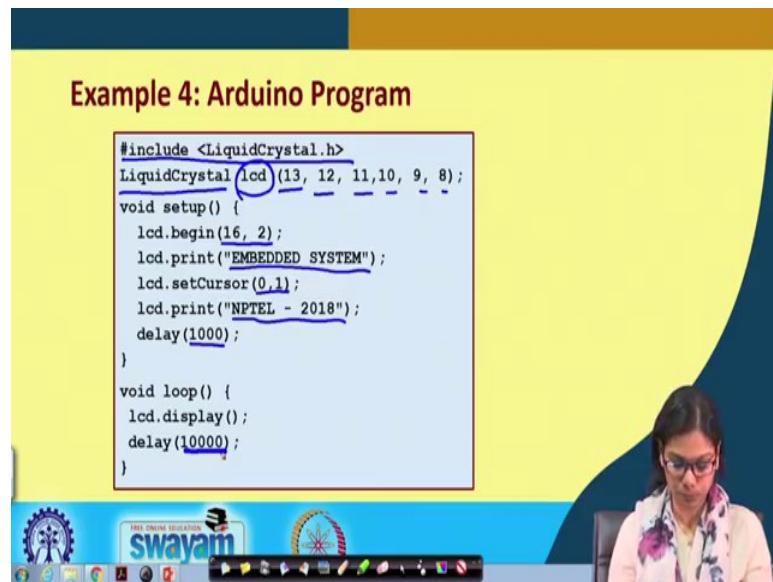
Similarly, as you have already seen that for the previous code for STM, we were using some header files. We were including some header files in the same way for Arduino you have to do the following in order, we know this textLCD function is already imported, it can be directly accessed in the example section.

(Refer Slide Time: 25:56)



This is the connection diagram, which is very similar to the previous one.

(Refer Slide Time: 26:29)



Example 4: Arduino Program

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);
void setup() {
    lcd.begin(16, 2);
    lcd.print("EMBEDDED SYSTEM");
    lcd.setCursor(0,1);
    lcd.print("NPTEL - 2018");
    delay(1000);
}
void loop() {
    lcd.display();
    delay(10000);
}
```

This is the code. So, here also you need to include this particular header file that is LiquidCrystal.h. For class LiquidCrystal, you make an object lcd and you just specify the pin numbers for connection. In the setup phase, you basically we do the following. lcd.begin specifies that this is a 16 x 2 display. lcd.print specifies the string EMBEDDED SYSTEM, with lcd.setCursor(0,1). The cursor will now move to this position and it will print NPTEL - 2018. And, we give a delay of 1 second and the same thing repeats.

So, we have come to the end of this lecture. In this lecture I have discussed about two output devices, one is 7 segment, another is LCD. And, I have also shown you the circuit diagrams both using Arduino and using STM, and how you can display any text using these 2 devices. We will move on and we will be showing some more interfacing experiment in the next lectures.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 23
Interfacing with 7-Segment LED and LCD Displays (Part II)

In this lecture we will be demonstrating you interfacing with LCD, with 7 segment, and LED.

We will be demonstrating you the experiments with the two set of boards. We first show you the experiment on LED interfacing, basically 7-segment LED. I have already discussed what is a 7-segment, how it works. There are two types of 7-segment display units, one is common anode another is common cathode.

In this experiment we will be taking common anode 7-segment unit. In a common anode display we will be connecting the common point through a resistor to Vcc, and to glow any of the LEDs we have to pass a 0. What we will be doing in this experiment is that we will be glowing the LED in some fashion, we will start from 0 and it will be a going to 1, 2 up to 9, and then again back to 0.

So, it will be a kind of counter that will count from 0 to 9, and then back to 0.

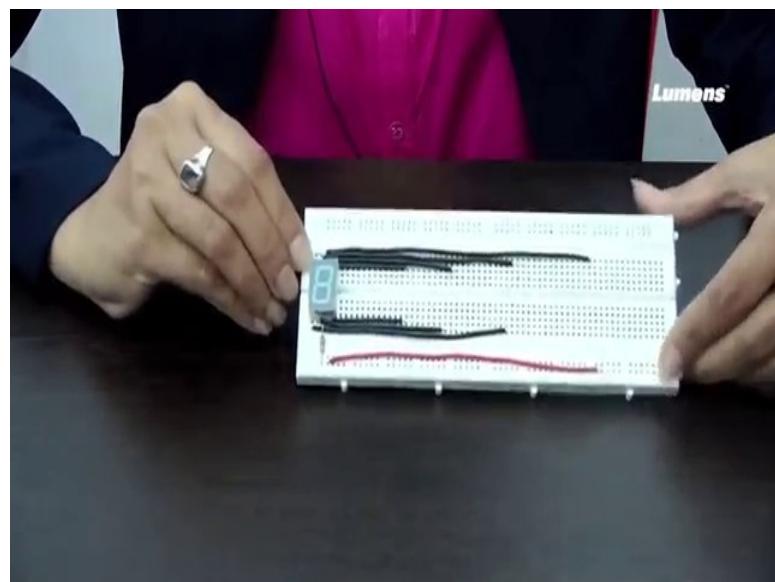
(Refer Slide Time: 02:27)



So, see this is a 7-segment display, you must have seen this in many places. Let us see what is the functionality or what are these segments. So, these are the segments that we have in this 7 segment display.

You see these are the pins. This is a common anode 7 segment, this from this particular common point through a resistor we will be connecting to Vcc.

(Refer Slide Time: 03:57)



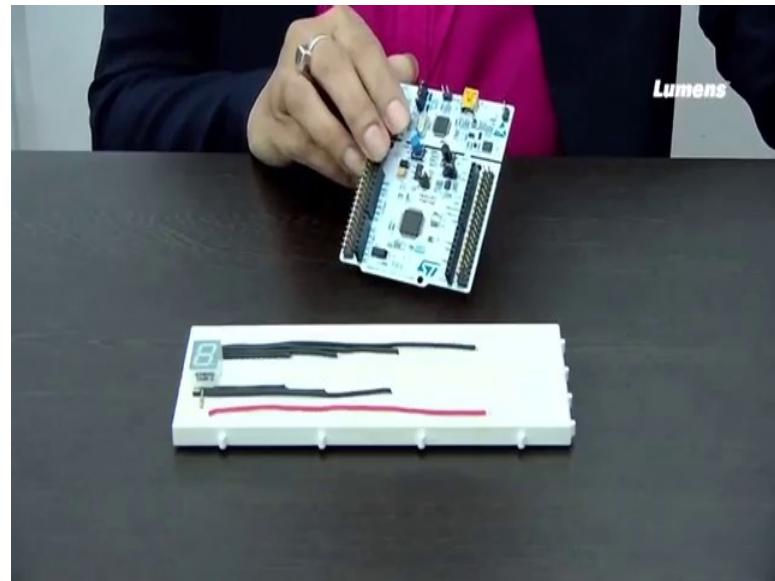
So, now I will be showing you the connection first with STM board.

And from the common point through a resistor I have connected this to Vcc. Also I will be connecting the segments to some of the digital output pins of both the boards.

As I have already told you I will be using pins from D2 to D8, basically and the common point will be connected to Vcc and then I will be writing the code where each segment will glow for little bit more than 1 second, and then it will move to the next one, and so on.

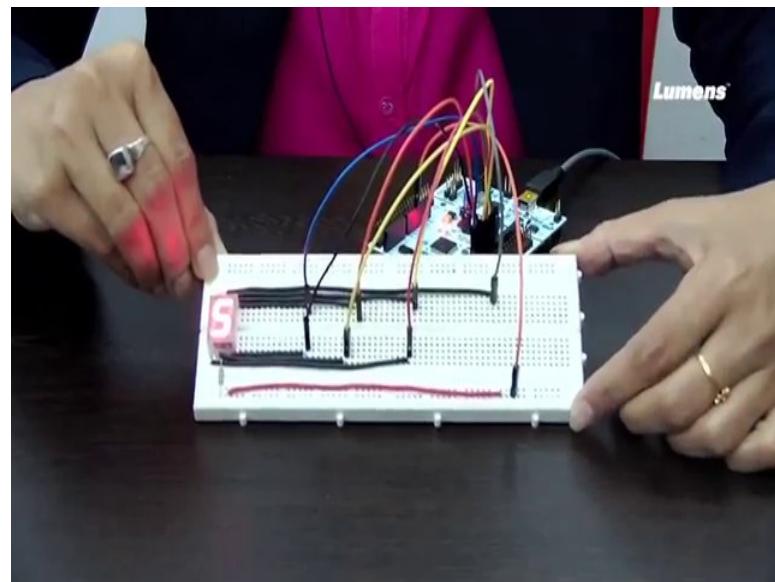
So, you have to make the connection in this fashion. Let me connect this to STM board first.

(Refer Slide Time: 05:18)



This is my STM board; as I have told you I will be connecting it from D2 to D8.

(Refer Slide Time: 05:31)

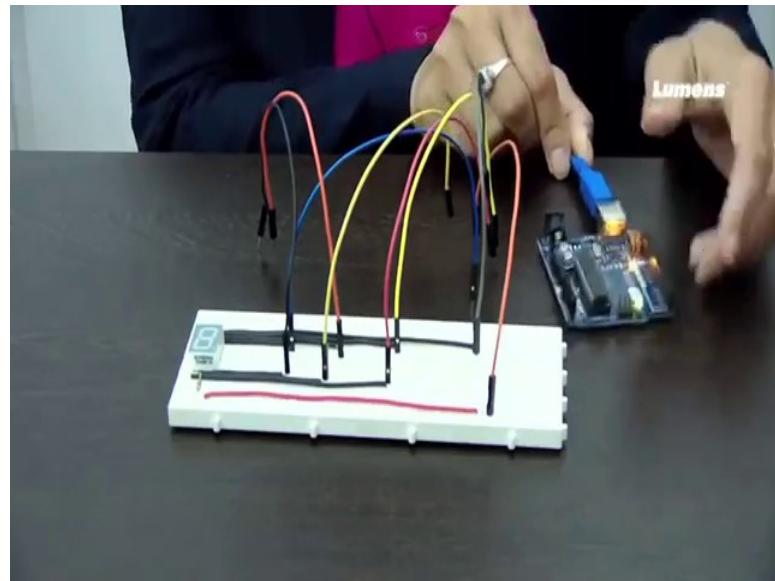


Connecting this to D3, next I will be connecting A to D2, B to D3, C to D4, D to D5, E to D6, F to D7, G to D8. From the common point through a resistor, this will be connected to Vcc. Now I will dump the code into the STM board.

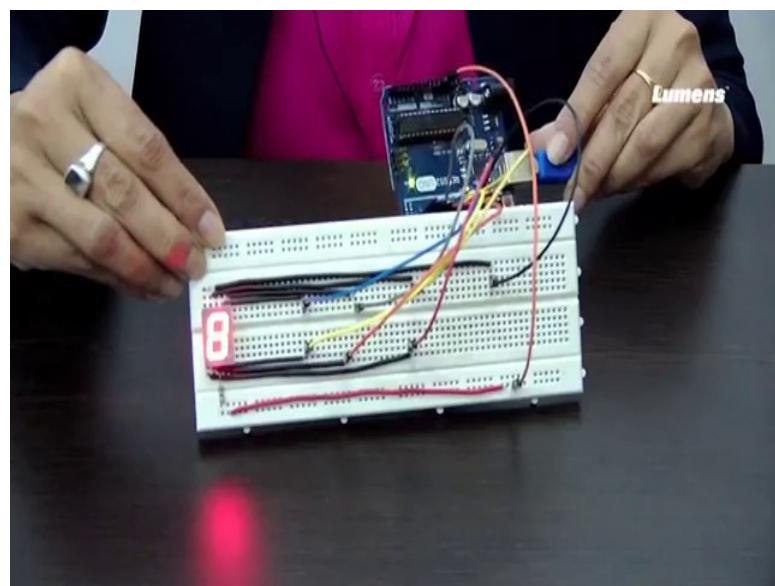
Now, you see what is happening. The digits are displayed one by one.

Now I will be doing the same thing with Arduino board.

(Refer Slide Time: 09:33)



(Refer Slide Time: 09:40)



Let me make the connection first, this is Vcc, and then from d2 to d8 I will be connecting.

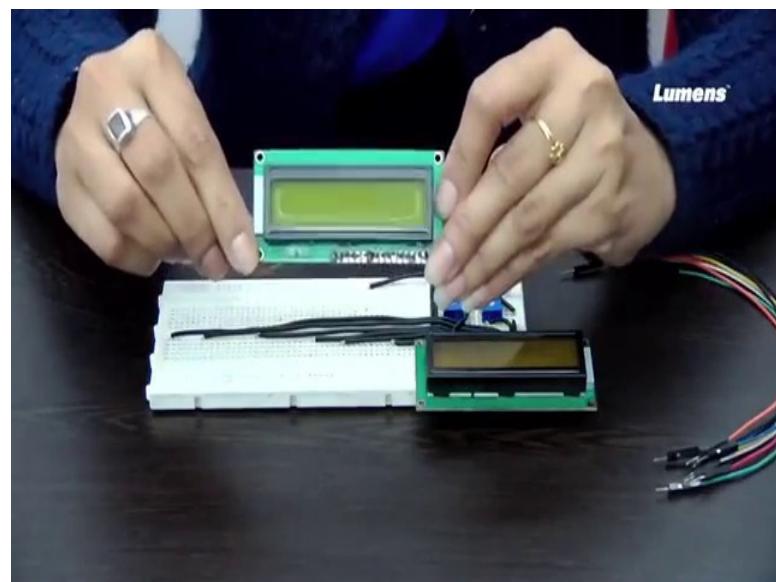
Now I will be connecting this connector and we will be putting up the code. Now see I have put up the code into this Arduino board, the same code that I have written for STM board and it is glowing.

Next I will be showing you how we can interface LCD, the same LCD that we have already discussed earlier. Now this particular LCD can display 2 lines of text, maximum

of 16 characters in each line, and if the text goes beyond 16 characters it scrolls. I have also discussed how we will be doing with two representative boards, one is with STM board and another is Arduino board.

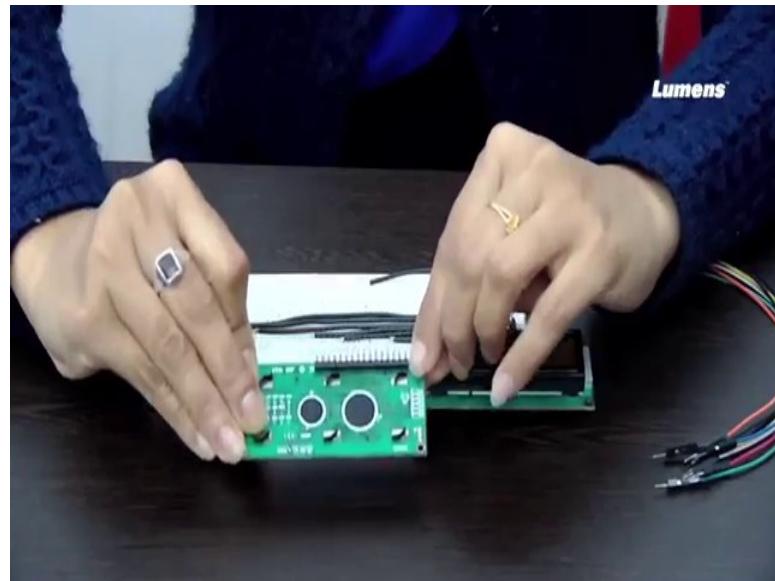
So, for both the boards we have to remember a few things. There are certain library functions that I have already discussed, those functions you have to include. When you write the code you have to make sure that you have already included those functions. The two functions LCDdisplay and LCDscroll we have already discussed.

(Refer Slide Time: 15:52)



This is the LCD we will now be interfacing; it has got 16 pins.

(Refer Slide Time: 116:07)

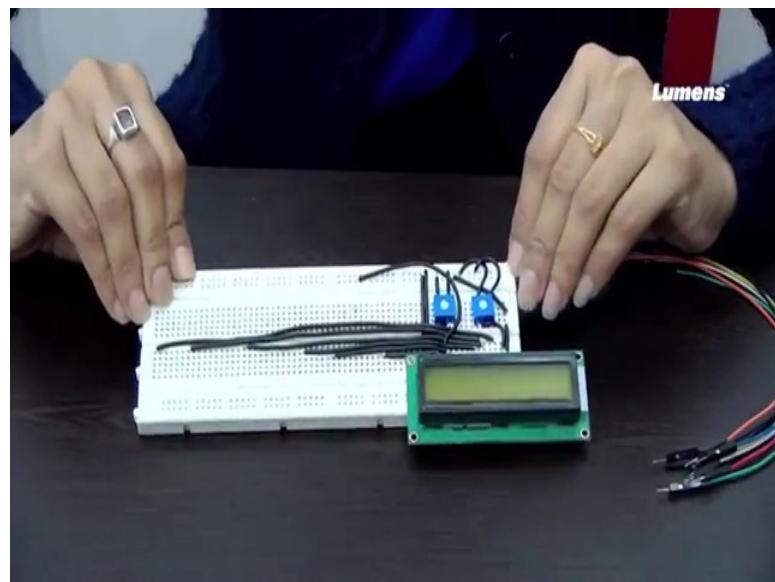


These are the pins basically. So, this particular pin is number 1, then 2 then 3 and so on till 16.

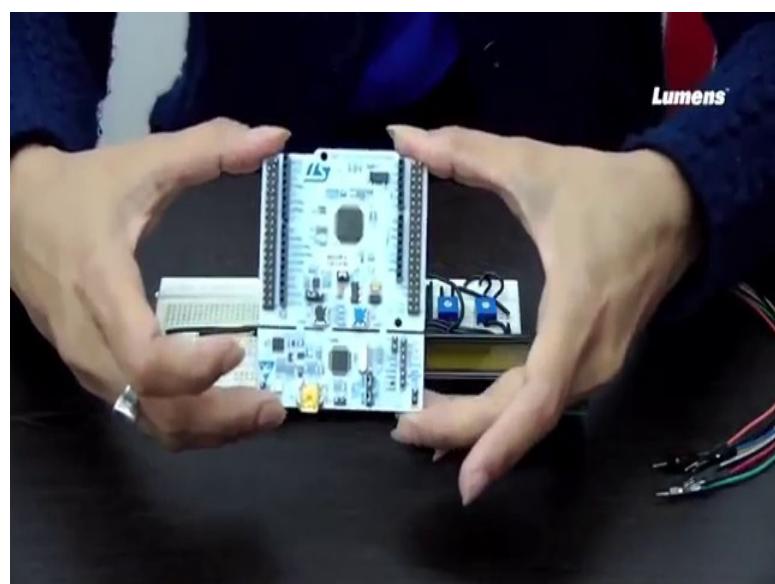
I have already discussed about all the pins, but let me very briefly tell that first pin will be connected with ground, the next pin will be connected with Vcc that is 5V, the next pin is the contrast control that we will be connecting using a potentiometer. For that potentiometer, one end will be connected to Vcc, another end will be connected to ground, and from the third pin we make the connection.

The fourth pin is register select that we will be connecting with the pin number d13. Now this is how we have done the connection, let us say you want to change the connection and you want to connect it to any other pin you have the freedom of doing that as well. The next is read/write, in this case we are only writing and so we have connected that to ground. The next pin is enable which is connected to d12 and then the consecutive 4 bits are connected to d11, d10, d9 and d8 of the pins of STM board.

(Refer Slide Time: 18:13)

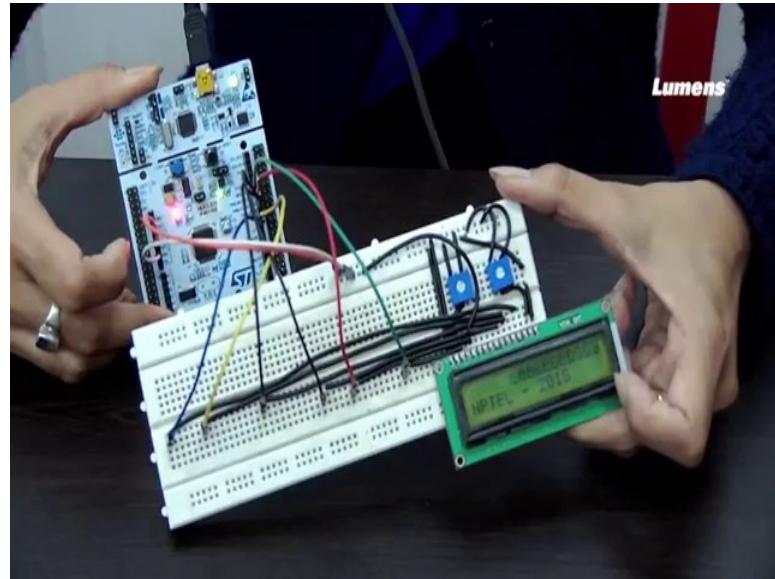


(Refer Slide Time: 18:37)



As I said I will be first showing you with the STM board; with this board we will be doing the connection and you already know about the pins of these boards. So, there is Vcc, ground, and then there are other digital pins available in this particular board. I will be connecting them one by one.

(Refer Slide Time: 19:08)



Let me first connect the first pin which is here to ground and the other one to Vcc, so these are the two common points.

As I said the first pin of the LCD is connected with ground, the next pin is connected to Vcc, the next pin is connected through this potentiometer.

Now, next one is register select that will be connected to d13. d12 will be connected to read/write; basically read/write is directly connected to ground. The enable pin is connected to d12.

d3 is register select, and this is the enable pin, and these 4 pins are basically the data pins.

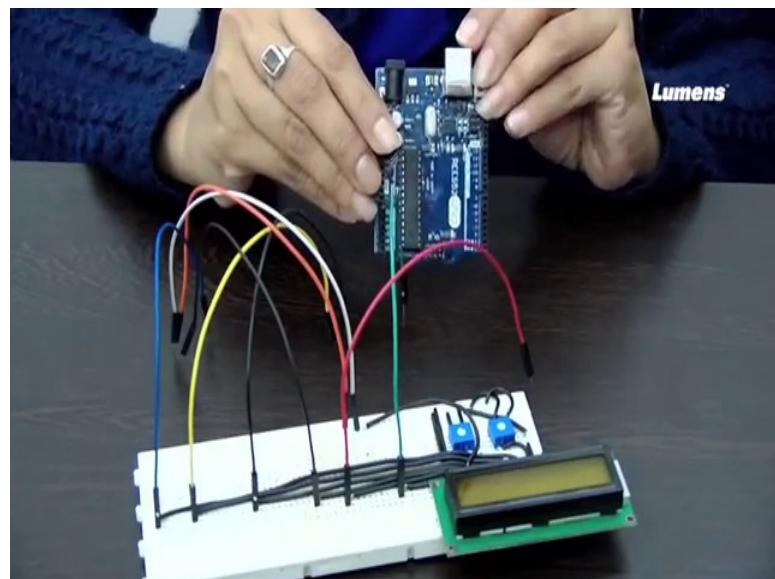
We have connected all the pins of this LCD starting from pin 1 till pin 16 to various pins into this STM board and internally to ground, Vcc. Follow the circuit diagram that is already provided to you in the slide where we have given exactly which pin will be connected to which pin of this particular board.

And cathode is directly connected to ground, this is all about the connection. If you want to make this at your end you require the following things; you require the following jumper wires, you require potentiometer, you require LCD and of course this board. Now what I will do the code that I have already written, I will be dumping it into this board. The text is getting displayed "embedded system NPTEL 2018".

Now, I will simply change this text to some other text, and let me see what happens. I have entered a text which is “embedded system design with ARM”, the text is scrolling as the number of characters is more than 16. There is also one important thing, you see the speed of this one. Here we have made in one second two characters will get scrolled. Now we can also make that in one second 3 characters will get scrolled. Now we will see that, I will not be dumping the other code where you will see that suddenly it will change its speed, can you see this?

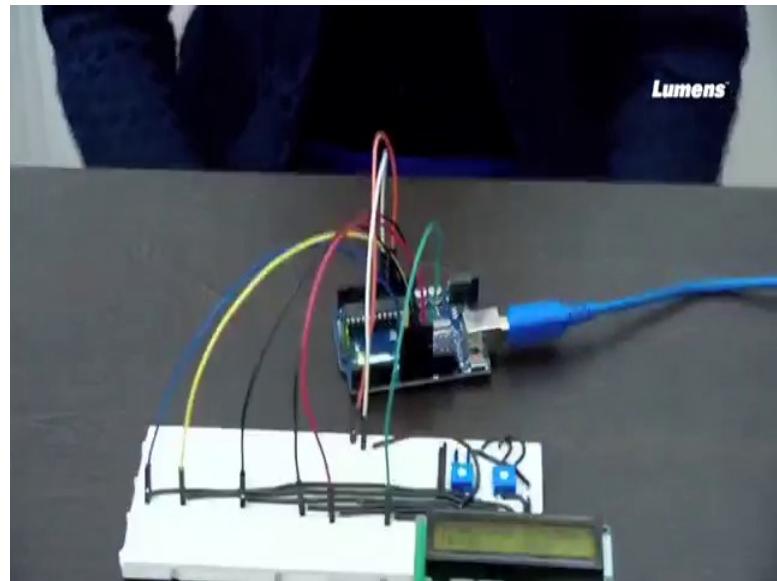
Now, you can see that it is going little faster as compared to the previous one. So, this is how you can make numerous variety of changes in this particular LCD, I would also like to show you now the connection with Arduino board.

(Refer Slide Time: 27:49)



So, this is the Arduino board which I have already discussed in detail. So, now, I will be doing the same kind of connection.

(Refer Slide Time: 28:01)



Now I will be showing you that how it works with Arduino, the code we have already seen how to make the code for Arduino. Now you see the code we have already dumped and now it is displaying “embedded system NPTEL 2018”.

Now we will be moving on with how do we interface various kinds of sensors.

Once we interface the sensors basically what we are doing is, we will be getting some parameters from the environment and those parameters we need to have the digital value displayed. So, you can either use 7segments for your display or you can do it using LCD. For all the experiments that we will be doing we would require something to be displayed; we will be using these two output devices.

Thank you.

Embedded System Design with Arm
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 24
Serial Port Terminal Application (Coolterm)

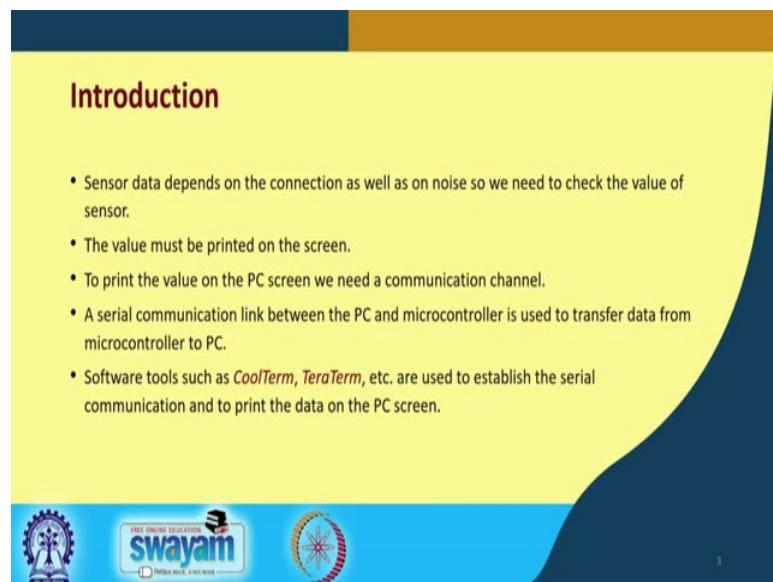
Welcome to lecture 24. In this lecture prior going towards working with sensors, we must know that what my sensor value is actually generating. In this regard, we will be discussing about a Serial Port Terminal Application called CoolTerm. There are other applications as well, but we have considered CoolTerm in this particular lecture.

(Refer Slide Time: 01:01)



In this lecture, I will talk about CoolTerm and of course, the demonstration.

(Refer Slide Time: 01:07)



Introduction

- Sensor data depends on the connection as well as on noise so we need to check the value of sensor.
- The value must be printed on the screen.
- To print the value on the PC screen we need a communication channel.
- A serial communication link between the PC and microcontroller is used to transfer data from microcontroller to PC.
- Software tools such as *CoolTerm*, *TeraTerm*, etc. are used to establish the serial communication and to print the data on the PC screen.

FREE ONLINE EDUCATION
swayam
India's first e-university

When we are using Arduino board there is a serial monitor. We just need to provide that connection; serial connection we have to create that object and through that object we can directly print it, but in STM board this is not available. So, for that purpose you need to print the value in some hyper terminal. This sensor data depends on connection as well as the noise as we need to check the value of the sensor. So, at every point in time, we need to check what value we are receiving from that sensor. The value gets printed on the screen.

To print the value on the PC screen, we need a communication channel. The communication channel must be built from the device, that is the STM board, to the PC. A serial communication link between the PC and the microcontroller is used to transfer data from microcontroller to PC, which is what we will be doing. We will be connecting the microcontroller using the USB cable and then we will be establishing a connection with the PC and then in the PC screen we will print the value.

The software tool such as CoolTerm or Teraterm can be used to establish the serial communication and to print the data on the screen. Let us see that.

(Refer Slide Time: 02:53)

HyperTerminal

- Software Tools that enables serial communication between the desktop/laptop and the microcontroller are termed as HyperTerminal Tool.
 - Examples: CoolTerm, TeraTerm, etc.
- For our experiments, it is required to ensure that there is some HyperTerminal installed on the computer.
 - Will enable serial data communication with the microcontroller.
 - HyperTerminal connections in the present context shall be made using USB port.

FREE ONLINE EDUCATION
swayam
India's first
one-stop
education
portal

What is a hyper terminal? It is a software tool that enables serial communication between a desktop or a laptop and the microcontroller. These are some example, one is CoolTerm, one is TeraTerm, we will be working with CoolTerm. So for our experiment, it is required to ensure that there is a hyper terminal installed in the computer. This will enable the serial data communication with the microcontroller and this hyper terminal connection in the present context shall be made using this USB port.

What we are essentially doing here is that we will connect first the device, the microcontroller, using the USB to our PC. Then we will use one hyper terminal, which we will set with the same baud rate with the port through which the microcontroller is connected with the PC. We need to check the port as there can be many ports.

(Refer Slide Time: 04:21)

CoolTerm

- In our experiments we shall be using *CoolTerm*.
 - Available on a wide variety of platforms.
- It can be downloaded from several sites:
 - http://download.cnet.com/CoolTerm/3000-2383_4-10915882.html
 - <http://coolterm.en.lo4d.com/>
- After extracting the file we shall get the *CoolTerm.exe* file.
- This will be used for serial connection between the PC/laptop and the STM32 kit.

FREE ONLINE EDUCATION
swayam
India Me Jana Me

A woman in a pink jacket is speaking on the right side of the slide.

CoolTerm can be downloaded from several sites. I have given these 2 representative site here, these are available from other places also.

After extracting the file we shall get this CoolTerm.exe file and this will be used for the serial communication between the PC or laptop and the STM32 kit.

(Refer Slide Time: 05:03)

Settings on CoolTerm

- For proper data communication, proper settings have to be made on the HyperTerminal software.
 - We shall explain the steps for CoolTerm.
- The default serial configuration in CoolTerm is 9600 bauds, 8-bit data.
- If you have not set any bauds rate in the program, then no need to change on this. Else it can be set to match with the program.

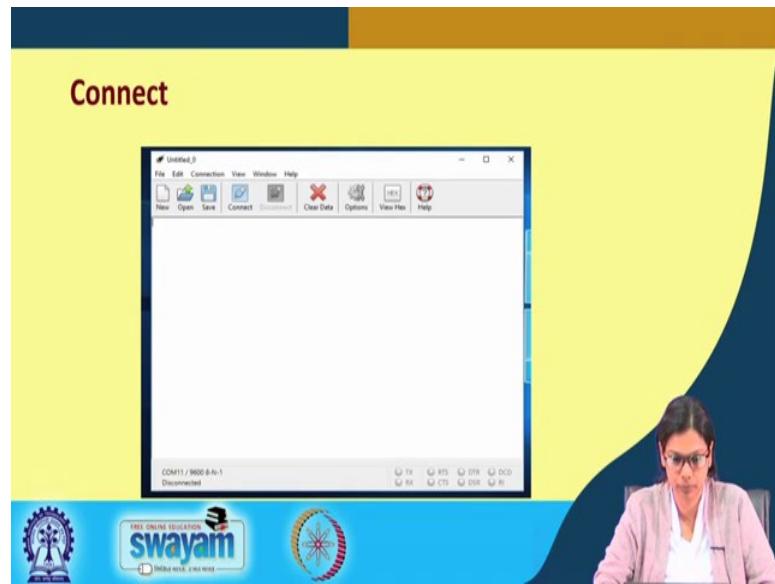
FREE ONLINE EDUCATION
swayam
India Me Jana Me

A woman in a pink jacket is speaking on the right side of the slide.

Now, for setting the CoolTerm, proper settings have to be made on the hyper terminal software. The default serial configuration is 9600 baud rate an 8 bits. If you want to change that you can do it.

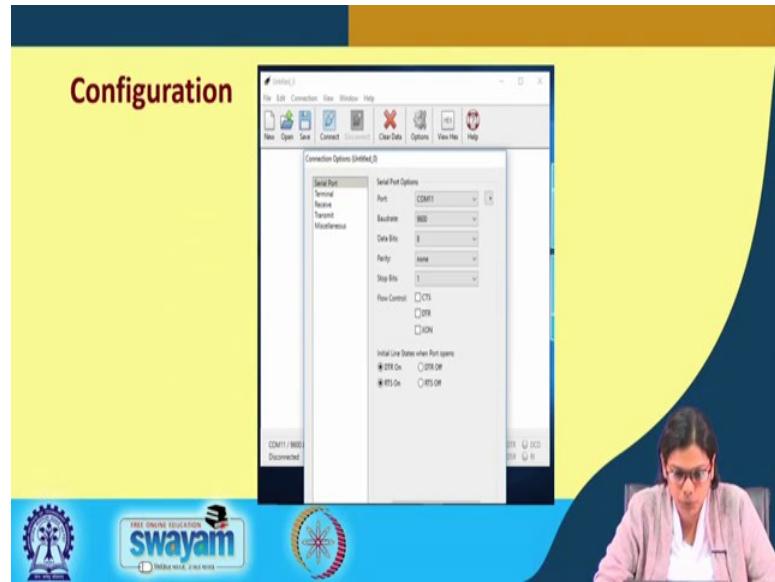
This setting must be done prior to using this CoolTerm.

(Refer Slide Time: 05:45)



Now once you open the CoolTerm you will get a screen like this and then you have to go to this option.

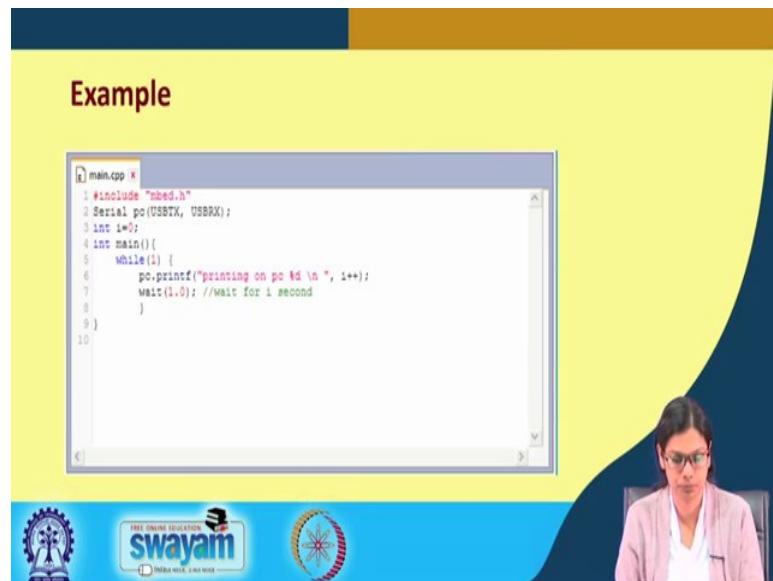
(Refer Slide Time: 05:57)



Once you go to this option you will find this port com11 here. So, it depends it is 11 for this example, but it depends on to which port it is connected in your machine. So, please make sure you check that and accordingly connect to that particular com port, and this is

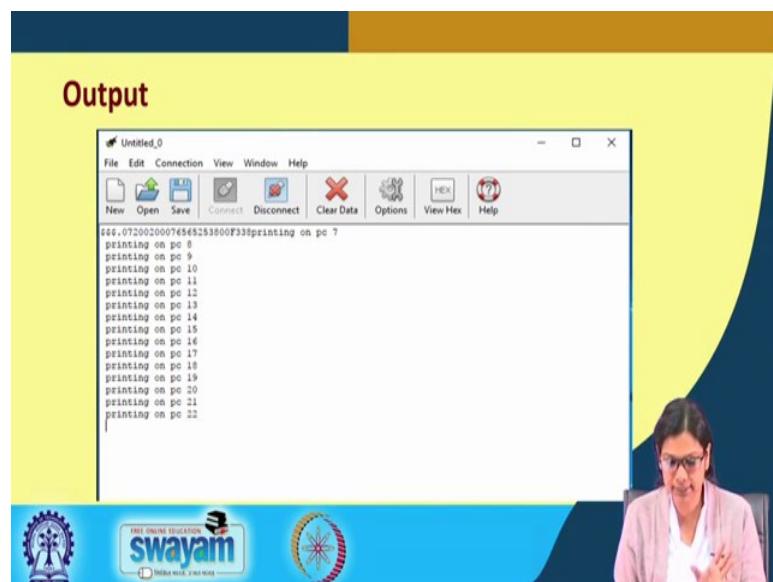
9600 and press for the configuration. Make sure to do this configuration prior to using CoolTerm.

(Refer Slide Time: 06:27)



Now, this is an example that we will be printing. I will be also showing you by connecting it with one of the sensors that we will be using in this week in a subsequent lecture.

(Refer Slide Time: 06:49)

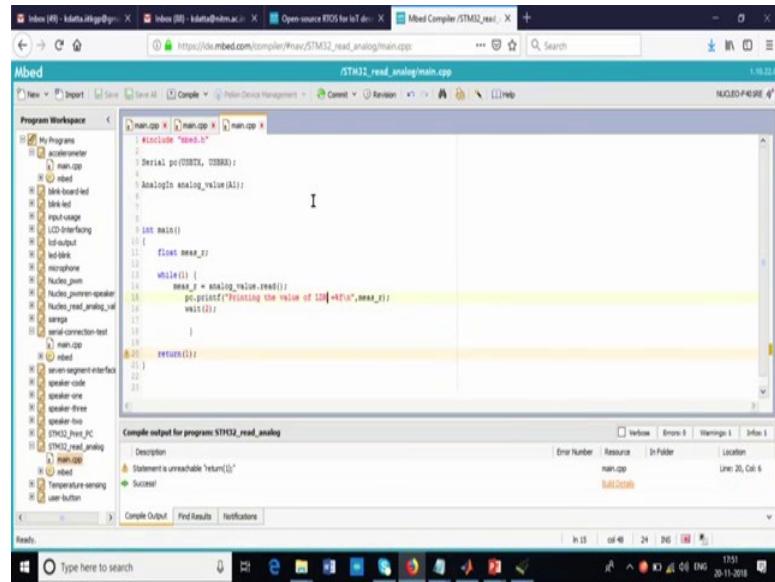


Once you write a code and make the serial communication, the serial communication will be made using this object that is serial PC USBTX and USBRX and here in this

case, we are just printing some value from the microcontroller. But in general case, what we will be doing? We will be printing the sensor value. In this case, the output is just getting incremented.

Now I will be showing you the configuration. I have already installed CoolTerm and this is how it goes. Basically you can disconnect it and then again you can connect it. So, just a second I will just make the connection first.

(Refer Slide Time: 08:35)

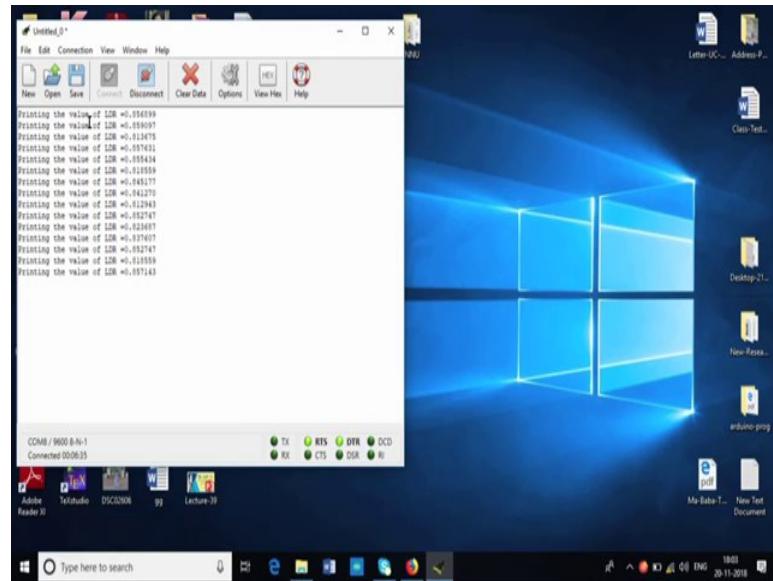


```
#include "seed.h"
#include "stm32f10x.h"
#include "uart.h"
#include "analog.h"

int main()
{
    float max_x;
    while(1)
    {
        max_x = analog_value.read();
        printf("Printing the value of %f\n",max_x);
        wait(2);
    }
    return(0);
}
```

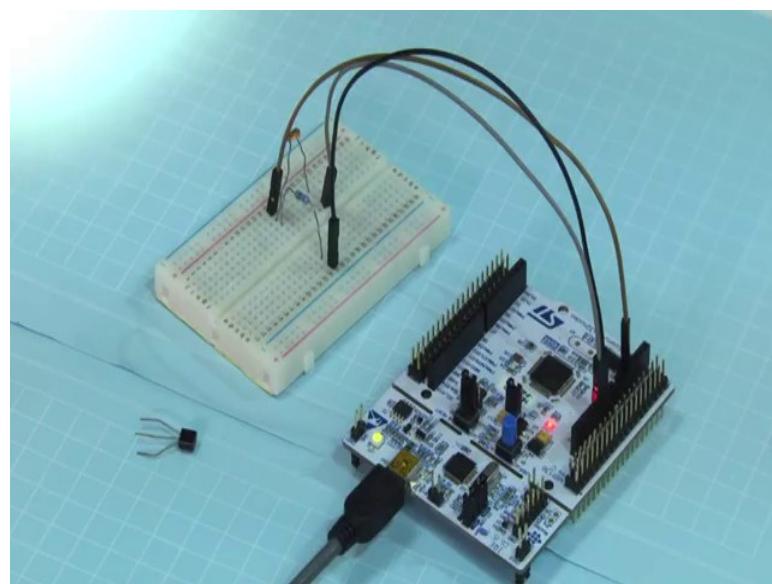
I will just compile it and then I have already told you, how you will dump the code. I am going to the download folder, where the code is available, which I will dump it in this.

(Refer Slide Time: 09:29)



I will disconnect and then again connect. This is where you can see that we are printing the value of LDR, I will show you the connection that I have made. This is printing some value from LDR; it is getting some value from the analog pin, and after analog to digital conversion it is showing the digital value. Now, I will just show you the connection diagram here.

(Refer Slide Time: 10:09)

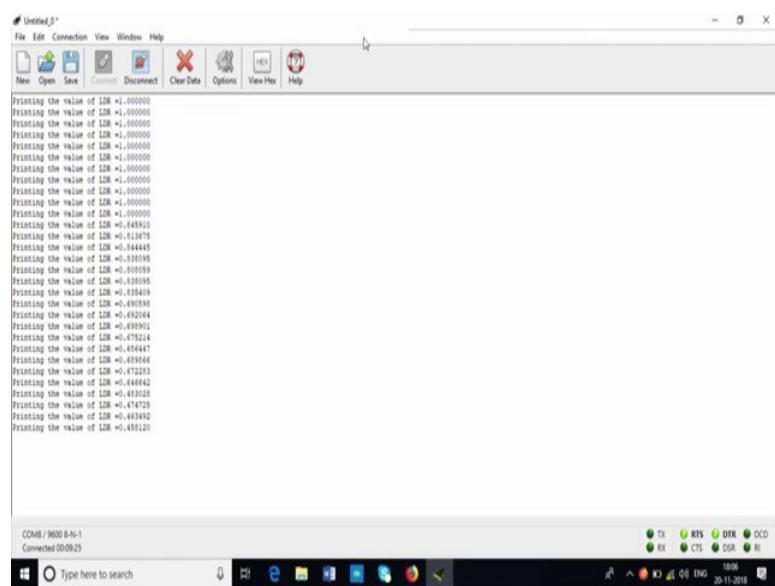


This is the LDR; I have made the connection. Later when I discussed about LDR in detail. One point of the LDR is connected to Vcc, another point through a resistance is

connected to ground. This is what we have done and we have seen that what value we are receiving.

You see that from this point, we have got the analog input into this A1 pin. So, we have got the analog input from here to A1 pin, what I will do basically, now is that I will put this bright light in this LDR slowly. I will take away the light and I will show you the screen, where the value changes now you see.

(Refer Slide Time: 11:23)



I will show you the screen, I have cleared out the data. Now see, it is printing 1. The maximum value the range of the digital output is 0 to 1. So, it is printing 1 now slowly, I will take away the light. Now see what value it is printing? Now it is printing 0.8. So, the value from 1 has been reduced to 0.8. Now I will put my hand there once, you see what value it is getting printed. Now it is getting printing 0.6. So, slowly the value is getting decreased, now I am putting little more hand. Now, it is coming to 0.4 and finally, I have put my hand here and now you see that it has been reduced to 0.2.

So, we have used CoolTerm to connect with the microcontroller and to display some value that we are receiving from the microcontroller into this PC. Prior to that I will just show you the code that is there for this one, you already come across with the codes, how you have to download the code, how you have to put the code into the STM board.

Now here, first you see we have to make this connection serial PC USBTX and USBRX this is the first thing, you have to do and here we are reading an analog value.

We will see in the subsequent weeks that we will be using or reading many analog values with various sensors. We may be requiring Coolterm to see the sensor outputs.

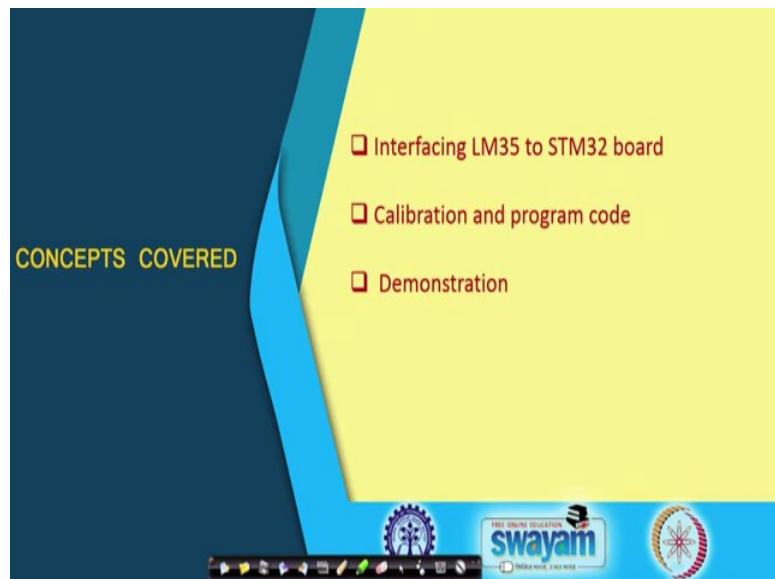
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institutes of Technology, Meghalaya

Lecture – 25
Experiment with Temperature Sensor

Welcome to lecture 25. In this week we will be interfacing various sensors with STM board. In this particular lecture we will be interfacing LM35 temperature sensor; LM stand for Linear Monolithic. We look into the properties of this LM35 temperature sensor and then we will look into the circuit diagram followed by the code that is used to interface this particular device with STM board, and then will show you the demonstration of this experiment.

(Refer Slide Time: 01:23)



We will interface LM35 to STM32 board will also show you how we can calibrate, because for any sensor to work some kind of calibration is required, the program code and the demonstration.

(Refer Slide Time: 01:49)

Temperature Sensing

- Sensing temperature has various applications:
 - Measure temperature of body, sophisticated industrial appliances, chemical plants, etc.
 - Weather forecast.
- Thermometer is the most widely used instrument for temperature sensing.
 - A mercury thermometer does not produce signals on temperature change, instead changes its property (viz. expansion or contraction).
 - A digital thermometer displays temperature on a tiny LCD display.

So, what is temperature sensing? Temperature is a physical parameter; sensing temperature has various applications. It can measure the temperature of a body, it is also used in sophisticated industrial appliances, it is used in chemical plants, where it is needed to monitor the temperature and depending on that temperature certain devices may be required to be made on or off, weather forecast, etc. If you think of how we measure temperature, thermometer is the most widely used instrument for temperature sensing.

A mercury thermometer does not produce signals on temperature change, instead it changes its property like it can expand or contract -- this is how a mercury thermometer works. On the other hand, a digital thermometer displays the temperature on a tiny LCD; we have also seen what an LCD is. In a digital thermometer, the temperature is displayed on a tiny LCD.

(Refer Slide Time: 03:29)

LM35 Temperature Sensor

- Precision integrated-circuit temperature device with an output voltage linearly proportional to the temperature in degrees Celsius (or Fahrenheit).
- Linear characteristic: $+10 \text{ mV}^{\circ} \text{ C}$ change in output.
- Rated for full -55° C to 150° C range.
- How to use the LM35?
 - Connect power supply (say, $+5\text{V}$ and GND) to pins 1 and 3.
 - Measure the analog voltage output on pin2.
 - We can connect pin 2 to one of the analog input pins of the microcontroller.

LM35 is a linear monolithic temperature sensor, this is an integrated circuit developed by National Semiconductor. This is a precision integrated circuit temperature device with an output voltage linearly proportional to the temperature in degree Celsius or Fahrenheit.

It means that the output voltage is linearly proportional to the temperature in degree Celsius. If you think of these linear characteristics, it is like there will be with 10 millivolt change for every degree Celsius increase in temperature. The full scale range of LM35 is -55 to $+150$ degree centigrade.

How do we use this LM 35? In LM35 there is a flat end and then there is a half round end, this way this pin number 1 is the voltage pin, this one should be connected to Vcc , pin number 3 must be connected to ground. The output voltage will be available from pin number 2.

Pin 1 should be connected to Vcc , pin 3 must be connected to ground, and from this middle point, we connect to the analog input port.

(Refer Slide Time: 07:17)

Reading Analog Value from LM35

- Connect the two extreme terminals to power (say, 5V) and GND.
 - The middle terminal will produce a voltage proportional to the temperature.

LM35

1 4-20V
2 OUT
3 GND

```
#include "mbed.h"
AnalogIn sensor(A1);
...
float p;
p = sensor.read();
...
pc.printf ("\n Value read: %f", p);
```

A0, A2, A3, A4
0 - 1K

How do we read the analog value? For reading the analog value, firstly we need to connect LM35 to ground and Vcc, and the middle terminal will produce a voltage proportional to the temperature.

The mbed C code that is used, you have to include this header file then we have to initialize an analog pin as an input analog pin. The A1 pin of the board is considered as the analog pin here, we take a float variable p because it will get a value ranging from 0 to 1. sensor.read will read the value, which is through this port A1 and you can print it using pc.printf. So, these are the few lines of code that are required to read the sensor value from the analog port A1. There are other analog ports as well like A0, A2, A3, A4 and A5.

(Refer Slide Time: 09:37)

Calibration

- LM35 output voltage increases linearly with increase in temperature.
 - 10 mV increase for every $^{\circ}\text{C}$ rise in temperature.
 - For $k^{\circ}\text{C}$, the output voltage will be $10k\text{ mV}$.
 - If maximum temperature is 50°C , maximum output voltage will be 0.5 V ($10 \times 50 = 500$).
- How to compute the temperature in $^{\circ}\text{C}$?
 - The `read()` function in `AnalogIn` class returns a fraction between 0 and 1.
 - Apply a voltage 0.5V directly to the analog input pin, and measure the value printed by the program (suppose, the value printed is P_{max}).
 - For an unknown temperature, if the value printed is P , the temperature value can be calculated as

$$T = 50 / P_{\text{max}} * P$$

50% P_{max}

1% P

FREE ONLINE EDUCATION **swayam**

Now, how do we do calibration? Suppose the current temperature of the room is x . For that current temperature, what value you are getting? If we know that now the temperature is, let us say 20 degree centigrade, I am getting certain value in my analog output, then if I am getting x value, what will be the temperature?

This is one way of doing the calibration. Let us see how we have done in this particular code. LM35 output voltage as I have already mentioned increases linearly with temperature. There is a 10 mV increase for every degree centigrade rise in temperature.

So, for k degree centigrade rise, the output voltage will be $10k\text{ mV}$. If maximum temperature is, let us say 50 degree centigrade, and the maximum output voltage is 0.5 volt. So, $10 \times 50 \text{ mV} = 500 \text{ mV}$. Now how do we compute the temperature in degree centigrade? Let us see this, the `read` function in the `AnalogIn` class returns a fraction between 0 and 1. I have already told that we can apply a voltage 0.5 volt directly to the analog input pin, and then we measure the value printed by the program. Suppose, the value which is printed is P_{max} . This means, for 0.5 volt the value we are getting is P_{max} . Then for an unknown temperature, if the value printed is P , then the temperature value can be calculated as $50 / P_{\text{max}} * P$.

(Refer Slide Time: 12:55)

Calibration

- LM35 output voltage increases linearly with increase in temperature.
- 10 mV increase for every $^{\circ}\text{C}$ rise in temperature.
- For $k^{\circ}\text{C}$, the output voltage will be $10k$ mV.
- If maximum temperature is 50°C , maximum output voltage will be 0.5 V ($10 \times 50 = 500$).
- How to compute the temperature in $^{\circ}\text{C}$?
 - The `read()` function in `AnalogIn` class returns a fraction between 0 and 1.
 - Apply a voltage 0.5V directly to the analog input pin, and measure the value printed by the program (suppose, the value printed is P_{max}).
 - For an unknown temperature, if the value printed is P , the temperature value can be calculated as

$$T = 50 / P_{\text{max}} * P$$

$P_{\text{max}} = 50$
 $1 = \frac{50}{50}$
 $P = \frac{P_{\text{max}} \times P}{50}$

(Refer Slide Time: 13:39)

Alternate way of computing:

- Suppose the room temperature is 20°C
- Using Coolterm, we run the program and suppose we find the value printed as 0.06
- For any unknown temperature, if the value printed is P , the temperature in $^{\circ}\text{C}$ will be $20P / 0.06$

$$0.06 = 20^{\circ}\text{C}$$
$$1 = \frac{20^{\circ}\text{C}}{0.06}$$
$$P = \frac{20^{\circ}\text{C} \times P}{0.06}$$

Let me show that the alternate way of computing that we have used. Suppose, the room temperature is 20 degree centigrade. We will see that what value we are getting from that voltage output. Suppose, the room temperature is 20 degree centigrade, we have already discussed about CoolTerm; suppose we find the value as 0.06.

So basically, if it is 0.06 then the temperature is 20 degree centigrade. If we receive the value P , then the temperature will be $20 / 0.06 * P$. So, we have earlier calculated this using the current temperature of the room that we were getting at that particular time.

(Refer Slide Time: 15:15)

An Experiment using LM35

- Design a *Temperature Sensing Unit* using STM32 Nucleo-F401RE kit and a LM35 temperature sensor to sense the environmental temperature, and display it in Centigrade scale on a LCD display unit.
 - The analog output voltage from LM35 is connected to pin A1 of the Arduino connector of the STM32 board.

FREE ONLINE EDUCATION **swayam**

A woman is speaking in the video feed.

Let us look into the experiment. I have discussed about LM35, how do we connect LM35 with STM and how do we read an analog value from certain port and also how do we calibrate. Now we will design a temperature sensing unit using STM Nucleo-F401RE kit and LM35 temperature sensor to sense the environmental temperature and display it in the centigrade scale on a LCD display unit. The analog output voltage from LM35 is connected to pin A1 of the STM board.

Let us now look into the circuit diagram.

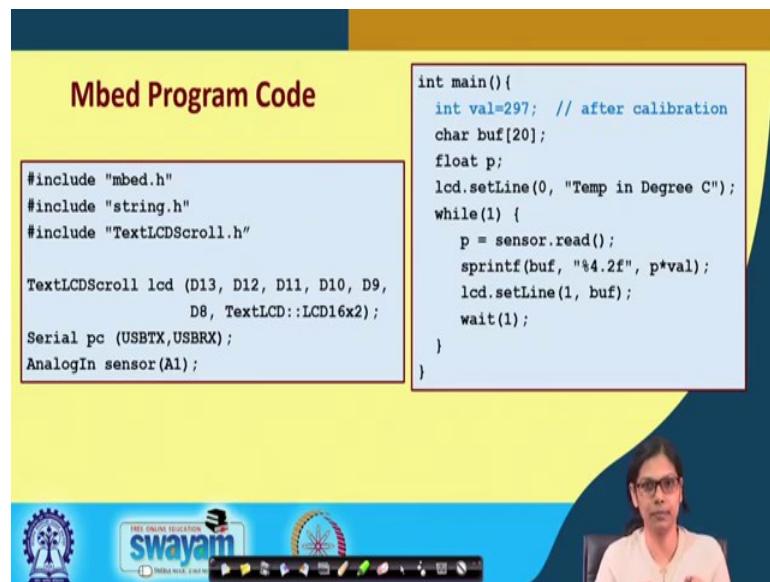
(Refer Slide Time: 16:13)

Connection Diagram

The diagram illustrates the connection between an STM32F401RE microcontroller and an LM35 temperature sensor. The STM32 board is shown with various pins labeled: V_{DD}, V_{REF}, V_{SS}, V_{DDA}, V_{SSA}, V_{DD3}, V_{SS3}, V_{DD4}, V_{SS4}, V_{DD5}, V_{SS5}, V_{DD6}, V_{SS6}, V_{DD7}, V_{SS7}, V_{DD8}, V_{SS8}, V_{DD9}, V_{SS9}, V_{DD10}, V_{SS10}, V_{DD11}, V_{SS11}, V_{DD12}, V_{SS12}, V_{DD13}, V_{SS13}, V_{DD14}, V_{SS14}, V_{DD15}, V_{SS15}, V_{DD16}, V_{SS16}, V_{DD17}, V_{SS17}, V_{DD18}, V_{SS18}, V_{DD19}, V_{SS19}, V_{DD20}, V_{SS20}, V_{DD21}, V_{SS21}, V_{DD22}, V_{SS22}, V_{DD23}, V_{SS23}, V_{DD24}, V_{SS24}, V_{DD25}, V_{SS25}, V_{DD26}, V_{SS26}, V_{DD27}, V_{SS27}, V_{DD28}, V_{SS28}, V_{DD29}, V_{SS29}, V_{DD30}, V_{SS30}, V_{DD31}, V_{SS31}, V_{DD32}, V_{SS32}, V_{DD33}, V_{SS33}, V_{DD34}, V_{SS34}, V_{DD35}, V_{SS35}, V_{DD36}, V_{SS36}, V_{DD37}, V_{SS37}, V_{DD38}, V_{SS38}, V_{DD39}, V_{SS39}, V_{DD40}, V_{SS40}, V_{DD41}, V_{SS41}, V_{DD42}, V_{SS42}, V_{DD43}, V_{SS43}, V_{DD44}, V_{SS44}, V_{DD45}, V_{SS45}, V_{DD46}, V_{SS46}, V_{DD47}, V_{SS47}, V_{DD48}, V_{SS48}, V_{DD49}, V_{SS49}, V_{DD50}, V_{SS50}, V_{DD51}, V_{SS51}, V_{DD52}, V_{SS52}, V_{DD53}, V_{SS53}, V_{DD54}, V_{SS54}, V_{DD55}, V_{SS55}, V_{DD56}, V_{SS56}, V_{DD57}, V_{SS57}, V_{DD58}, V_{SS58}, V_{DD59}, V_{SS59}, V_{DD60}, V_{SS60}, V_{DD61}, V_{SS61}, V_{DD62}, V_{SS62}, V_{DD63}, V_{SS63}, V_{DD64}, V_{SS64}, V_{DD65}, V_{SS65}, V_{DD66}, V_{SS66}, V_{DD67}, V_{SS67}, V_{DD68}, V_{SS68}, V_{DD69}, V_{SS69}, V_{DD70}, V_{SS70}, V_{DD71}, V_{SS71}, V_{DD72}, V_{SS72}, V_{DD73}, V_{SS73}, V_{DD74}, V_{SS74}, V_{DD75}, V_{SS75}, V_{DD76}, V_{SS76}, V_{DD77}, V_{SS77}, V_{DD78}, V_{SS78}, V_{DD79}, V_{SS79}, V_{DD80}, V_{SS80}, V_{DD81}, V_{SS81}, V_{DD82}, V_{SS82}, V_{DD83}, V_{SS83}, V_{DD84}, V_{SS84}, V_{DD85}, V_{SS85}, V_{DD86}, V_{SS86}, V_{DD87}, V_{SS87}, V_{DD88}, V_{SS88}, V_{DD89}, V_{SS89}, V_{DD90}, V_{SS90}, V_{DD91}, V_{SS91}, V_{DD92}, V_{SS92}, V_{DD93}, V_{SS93}, V_{DD94}, V_{SS94}, V_{DD95}, V_{SS95}, V_{DD96}, V_{SS96}, V_{DD97}, V_{SS97}, V_{DD98}, V_{SS98}, V_{DD99}, V_{SS99}, V_{DD100}, V_{SS100}, V_{DD101}, V_{SS101}, V_{DD102}, V_{SS102}, V_{DD103}, V_{SS103}, V_{DD104}, V_{SS104}, V_{DD105}, V_{SS105}, V_{DD106}, V_{SS106}, V_{DD107}, V_{SS107}, V_{DD108}, V_{SS108}, V_{DD109}, V_{SS109}, V_{DD110}, V_{SS110}, V_{DD111}, V_{SS111}, V_{DD112}, V_{SS112}, V_{DD113}, V_{SS113}, V_{DD114}, V_{SS114}, V_{DD115}, V_{SS115}, V_{DD116}, V_{SS116}, V_{DD117}, V_{SS117}, V_{DD118}, V_{SS118}, V_{DD119}, V_{SS119}, V_{DD120}, V_{SS120}, V_{DD121}, V_{SS121}, V_{DD122}, V_{SS122}, V_{DD123}, V_{SS123}, V_{DD124}, V_{SS124}, V_{DD125}, V_{SS125}, V_{DD126}, V_{SS126}, V_{DD127}, V_{SS127}, V_{DD128}, V_{SS128}, V_{DD129}, V_{SS129}, V_{DD130}, V_{SS130}, V_{DD131}, V_{SS131}, V_{DD132}, V_{SS132}, V_{DD133}, V_{SS133}, V_{DD134}, V_{SS134}, V_{DD135}, V_{SS135}, V_{DD136}, V_{SS136}, V_{DD137}, V_{SS137}, V_{DD138}, V_{SS138}, V_{DD139}, V_{SS139}, V_{DD140}, V_{SS140}, V_{DD141}, V_{SS141}, V_{DD142}, V_{SS142}, V_{DD143}, V_{SS143}, V_{DD144}, V_{SS144}, V_{DD145}, V_{SS145}, V_{DD146}, V_{SS146}, V_{DD147}, V_{SS147}, V_{DD148}, V_{SS148}, V_{DD149}, V_{SS149}, V_{DD150}, V_{SS150}, V_{DD151}, V_{SS151}, V_{DD152}, V_{SS152}, V_{DD153}, V_{SS153}, V_{DD154}, V_{SS154}, V_{DD155}, V_{SS155}, V_{DD156}, V_{SS156}, V_{DD157}, V_{SS157}, V_{DD158}, V_{SS158}, V_{DD159}, V_{SS159}, V_{DD160}, V_{SS160}, V_{DD161}, V_{SS161}, V_{DD162}, V_{SS162}, V_{DD163}, V_{SS163}, V_{DD164}, V_{SS164}, V_{DD165}, V_{SS165}, V_{DD166}, V_{SS166}, V_{DD167}, V_{SS167}, V_{DD168}, V_{SS168}, V_{DD169}, V_{SS169}, V_{DD170}, V_{SS170}, V_{DD171}, V_{SS171}, V_{DD172}, V_{SS172}, V_{DD173}, V_{SS173}, V_{DD174}, V_{SS174}, V_{DD175}, V_{SS175}, V_{DD176}, V_{SS176}, V_{DD177}, V_{SS177}, V_{DD178}, V_{SS178}, V_{DD179}, V_{SS179}, V_{DD180}, V_{SS180}, V_{DD181}, V_{SS181}, V_{DD182}, V_{SS182}, V_{DD183}, V_{SS183}, V_{DD184}, V_{SS184}, V_{DD185}, V_{SS185}, V_{DD186}, V_{SS186}, V_{DD187}, V_{SS187}, V_{DD188}, V_{SS188}, V_{DD189}, V_{SS189}, V_{DD190}, V_{SS190}, V_{DD191}, V_{SS191}, V_{DD192}, V_{SS192}, V_{DD193}, V_{SS193}, V_{DD194}, V_{SS194}, V_{DD195}, V_{SS195}, V_{DD196}, V_{SS196}, V_{DD197}, V_{SS197}, V_{DD198}, V_{SS198}, V_{DD199}, V_{SS199}, V_{DD200}, V_{SS200}, V_{DD201}, V_{SS201}, V_{DD202}, V_{SS202}, V_{DD203}, V_{SS203}, V_{DD204}, V_{SS204}, V_{DD205}, V_{SS205}, V_{DD206}, V_{SS206}, V_{DD207}, V_{SS207}, V_{DD208}, V_{SS208}, V_{DD209}, V_{SS209}, V_{DD210}, V_{SS210}, V_{DD211}, V_{SS211}, V_{DD212}, V_{SS212}, V_{DD213}, V_{SS213}, V_{DD214}, V_{SS214}, V_{DD215}, V_{SS215}, V_{DD216}, V_{SS216}, V_{DD217}, V_{SS217}, V_{DD218}, V_{SS218}, V_{DD219}, V_{SS219}, V_{DD220}, V_{SS220}, V_{DD221}, V_{SS221}, V_{DD222}, V_{SS222}, V_{DD223}, V_{SS223}, V_{DD224}, V_{SS224}, V_{DD225}, V_{SS225}, V_{DD226}, V_{SS226}, V_{DD227}, V_{SS227}, V_{DD228}, V_{SS228}, V_{DD229}, V_{SS229}, V_{DD230}, V_{SS230}, V_{DD231}, V_{SS231}, V_{DD232}, V_{SS232}, V_{DD233}, V_{SS233}, V_{DD234}, V_{SS234}, V_{DD235}, V_{SS235}, V_{DD236}, V_{SS236}, V_{DD237}, V_{SS237}, V_{DD238}, V_{SS238}, V_{DD239}, V_{SS239}, V_{DD240}, V_{SS240}, V_{DD241}, V_{SS241}, V_{DD242}, V_{SS242}, V_{DD243}, V_{SS243}, V_{DD244}, V_{SS244}, V_{DD245}, V_{SS245}, V_{DD246}, V_{SS246}, V_{DD247}, V_{SS247}, V_{DD248}, V_{SS248}, V_{DD249}, V_{SS249}, V_{DD250}, V_{SS250}, V_{DD251}, V_{SS251}, V_{DD252}, V_{SS252}, V_{DD253}, V_{SS253}, V_{DD254}, V_{SS254}, V_{DD255}, V_{SS255}, V_{DD256}, V_{SS256}, V_{DD257}, V_{SS257}, V_{DD258}, V_{SS258}, V_{DD259}, V_{SS259}, V_{DD260}, V_{SS260}, V_{DD261}, V_{SS261}, V_{DD262}, V_{SS262}, V_{DD263}, V_{SS263}, V_{DD264}, V_{SS264}, V_{DD265}, V_{SS265}, V_{DD266}, V_{SS266}, V_{DD267}, V_{SS267}, V_{DD268}, V_{SS268}, V_{DD269}, V_{SS269}, V_{DD270}, V_{SS270}, V_{DD271}, V_{SS271}, V_{DD272}, V_{SS272}, V_{DD273}, V_{SS273}, V_{DD274}, V_{SS274}, V_{DD275}, V_{SS275}, V_{DD276}, V_{SS276}, V_{DD277}, V_{SS277}, V_{DD278}, V_{SS278}, V_{DD279}, V_{SS279}, V_{DD280}, V_{SS280}, V_{DD281}, V_{SS281}, V_{DD282}, V_{SS282}, V_{DD283}, V_{SS283}, V_{DD284}, V_{SS284}, V_{DD285}, V_{SS285}, V_{DD286}, V_{SS286}, V_{DD287}, V_{SS287}, V_{DD288}, V_{SS288}, V_{DD289}, V_{SS289}, V_{DD290}, V_{SS290}, V_{DD291}, V_{SS291}, V_{DD292}, V_{SS292}, V_{DD293}, V_{SS293}, V_{DD294}, V_{SS294}, V_{DD295}, V_{SS295}, V_{DD296}, V_{SS296}, V_{DD297}, V_{SS297}, V_{DD298}, V_{SS298}, V_{DD299}, V_{SS299}, V_{DD300}, V_{SS300}, V_{DD301}, V_{SS301}, V_{DD302}, V_{SS302}, V_{DD303}, V_{SS303}, V_{DD304}, V_{SS304}, V_{DD305}, V_{SS305}, V_{DD306}, V_{SS306}, V_{DD307}, V_{SS307}, V_{DD308}, V_{SS308}, V_{DD309}, V_{SS309}, V_{DD310}, V_{SS310}, V_{DD311}, V_{SS311}, V_{DD312}, V_{SS312}, V_{DD313}, V_{SS313}, V_{DD314}, V_{SS314}, V_{DD315}, V_{SS315}, V_{DD316}, V_{SS316}, V_{DD317}, V_{SS317}, V_{DD318}, V_{SS318}, V_{DD319}, V_{SS319}, V_{DD320}, V_{SS320}, V_{DD321}, V_{SS321}, V_{DD322}, V_{SS322}, V_{DD323}, V_{SS323}, V_{DD324}, V_{SS324}, V_{DD325}, V_{SS325}, V_{DD326}, V_{SS326}, V_{DD327}, V_{SS327}, V_{DD328}, V_{SS328}, V_{DD329}, V_{SS329}, V_{DD330}, V_{SS330}, V_{DD331}, V_{SS331}, V_{DD332}, V_{SS332}, V_{DD333}, V_{SS333}, V_{DD334}, V_{SS334}, V_{DD335}, V_{SS335}, V_{DD336}, V_{SS336}, V_{DD337}, V_{SS337}, V_{DD338}, V_{SS338}, V_{DD339}, V_{SS339}, V_{DD340}, V_{SS340}, V_{DD341}, V_{SS341}, V_{DD342}, V_{SS342}, V_{DD343}, V_{SS343}, V_{DD344}, V_{SS344}, V_{DD345}, V_{SS345}, V_{DD346}, V_{SS346}, V_{DD347}, V_{SS347}, V_{DD348}, V_{SS348}, V_{DD349}, V_{SS349}, V_{DD350}, V_{SS350}, V_{DD351}, V_{SS351}, V_{DD352}, V_{SS352}, V_{DD353}, V_{SS353}, V_{DD354}, V_{SS354}, V_{DD355}, V_{SS355}, V_{DD356}, V_{SS356}, V_{DD357}, V_{SS357}, V_{DD358}, V_{SS358}, V_{DD359}, V_{SS359}, V_{DD360}, V_{SS360}, V_{DD361}, V_{SS361}, V_{DD362}, V_{SS362}, V_{DD363}, V_{SS363}, V_{DD364}, V_{SS364}, V_{DD365}, V_{SS365}, V_{DD366}, V_{SS366}, V_{DD367}, V_{SS367}, V_{DD368}, V_{SS368}, V_{DD369}, V_{SS369}, V_{DD370}, V_{SS370}, V_{DD371}, V_{SS371}, V_{DD372}, V_{SS372}, V_{DD373}, V_{SS373}, V_{DD374}, V_{SS374}, V_{DD375}, V_{SS375}, V_{DD376}, V_{SS376}, V_{DD377}, V_{SS377}, V_{DD378}, V_{SS378}, V_{DD379}, V_{SS379}, V_{DD380}, V_{SS380}, V_{DD381}, V_{SS381}, V_{DD382}, V_{SS382}, V_{DD383}, V_{SS383}, V_{DD384}, V_{SS384}, V_{DD385}, V_{SS385}, V_{DD386}, V_{SS386}, V_{DD387}, V_{SS387}, V_{DD388}, V_{SS388}, V_{DD389}, V_{SS389}, V_{DD390}, V_{SS390}, V_{DD391}, V_{SS391}, V_{DD392}, V_{SS392}, V_{DD393}, V_{SS393}, V_{DD394}, V_{SS394}, V_{DD395}, V_{SS395}, V_{DD396}, V_{SS396}, V_{DD397}, V_{SS397}, V_{DD398}, V_{SS398}, V_{DD399}, V_{SS399}, V_{DD400}, V_{SS400}, V_{DD401}, V_{SS401}, V_{DD402}, V_{SS402}, V_{DD403}, V_{SS403}, V_{DD404}, V_{SS404}, V_{DD405}, V_{SS405}, V_{DD406}, V_{SS406}, V_{DD407}, V_{SS407}, V_{DD408}, V_{SS408}, V_{DD409}, V_{SS409}, V_{DD410}, V_{SS410}, V_{DD411}, V_{SS411}, V_{DD412}, V_{SS412}, V_{DD413}, V_{SS413}, V_{DD414}, V_{SS414}, V_{DD415}, V_{SS415}, V_{DD416}, V_{SS416}, V_{DD417}, V_{SS417}, V_{DD418}, V_{SS418}, V_{DD419}, V_{SS419}, V_{DD420}, V_{SS420}, V_{DD421}, V_{SS421}, V_{DD422}, V_{SS422}, V_{DD423}, V_{SS423}, V_{DD424}, V_{SS424}, V_{DD425}, V_{SS425}, V_{DD426}, V_{SS426}, V_{DD427}, V_{SS427}, V_{DD428}, V_{SS428}, V_{DD429}, V_{SS429}, V_{DD430}, V_{SS430}, V_{DD431}, V_{SS431}, V_{DD432}, V_{SS432}, V_{DD433}, V_{SS433}, V_{DD434}, V_{SS434}, V_{DD435}, V_{SS435}, V_{DD436}, V_{SS436}, V_{DD437}, V_{SS437}, V_{DD438}, V_{SS438}, V_{DD439}, V_{SS439}, V_{DD440}, V_{SS440}, V_{DD441}, V_{SS441}, V_{DD442}, V_{SS442}, V_{DD443}, V_{SS443}, V_{DD444}, V_{SS444}, V_{DD445}, V_{SS445}, V_{DD446}, V_{SS446}, V_{DD447}, V_{SS447}, V_{DD448}, V_{SS448}, V_{DD449}, V_{SS449}, V_{DD450}, V_{SS450}, V_{DD451}, V_{SS451}, V_{DD452}, V_{SS452}, V_{DD453}, V_{SS453}, V_{DD454}, V_{SS454}, V_{DD455}, V_{SS455}, V_{DD456}, V_{SS456}, V_{DD457}, V_{SS457}, V_{DD458}, V_{SS458}, V_{DD459}, V_{SS459}, V_{DD460}, V_{SS460}, V_{DD461}, V_{SS461}, V_{DD462}, V_{SS462}, V_{DD463}, V_{SS463}, V_{DD464}, V_{SS464}, V_{DD465}, V_{SS465}, V_{DD466}, V_{SS466}, V_{DD467}, V_{SS467}, V_{DD468}, V_{SS468}, V_{DD469}, V_{SS469}, V_{DD470}, V_{SS470}, V_{DD471}, V_{SS471}, V_{DD472}, V_{SS472}, V_{DD473}, V_{SS473}, V_{DD474}, V_{SS474}, V_{DD475}, V_{SS475}, V_{DD476}, V_{SS476}, V_{DD477}, V_{SS477}, V_{DD478}, V_{SS478}, V_{DD479}, V_{SS479}, V_{DD480}, V_{SS480}, V_{DD481}, V_{SS481}, V_{DD482}, V_{SS482}, V_{DD483}, V_{SS483}, V_{DD484}, V_{SS484}, V_{DD485}, V_{SS485}, V_{DD486}, V_{SS486}, V_{DD487}, V_{SS487}, V_{DD488}, V_{SS488}, V_{DD489}, V_{SS489}, V_{DD490}, V_{SS490}

We are basically connecting from D8 to D13 to various pins of the LCD display, what we are going here to do is that, we are connecting this LM35 VCC output to pin A1. This is all about the connection and this part of the connection we already discussed earlier. So, it will sense the temperature, do the calibration and display the temperature in this LCD. We will be demonstrating this example, but before that I will be showing you the code that is required to be executed after making the circuit.

(Refer Slide Time: 17:23)



Mbed Program Code

```
#include "mbed.h"
#include "string.h"
#include "TextLCDScroll.h"

TextLCDScroll lcd (D13, D12, D11, D10, D9,
                    D8, TextLCD::LCD16x2);
Serial pc (USBTX,USBRX);
AnalogIn sensor(A1);

int main(){
    int val=297; // after calibration
    char buf[20];
    float p;
    lcd.setLine(0, "Temp in Degree C");
    while(1) {
        p = sensor.read();
        sprintf(buf, "%4.2f", p*val);
        lcd.setLine(1, buf);
        wait(1);
    }
}
```

So, this is the mbed program code, we know that we have to use TextLCDScroll.

We use here serial USBTX RX, this is for serial communication that we do such that we can display it in the any of the hyper terminal, which we have already discussed. This is the analog input value in the name of sensor, we are reading through port A1 and this is after calibration, we get a value like this ... value is 297.

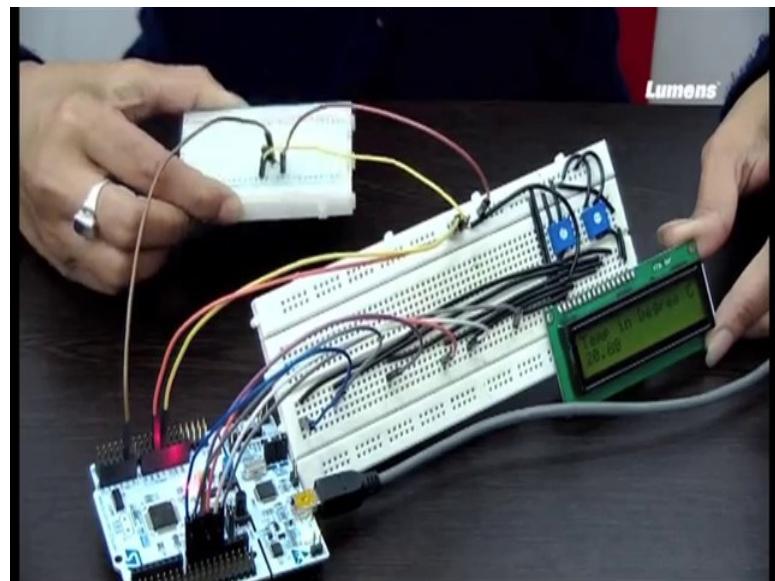
We take a character buffer here and we are setting the 0th line to "Temp in Degree C". In the while loop, we are reading the value first using sensor.read and then we are calculating the value $p * val$. p is whatever we have got that we are multiplying with this value 297 and we are storing it in this buffer, and finally we are displaying it in the next line. We are waiting for 1 second and again we are doing the same thing repeatedly.

So now, that I have already discussed the whole code how do we interface it and how do we display it. Now we will be showing you the experiment the demonstration of this

particular code, where we will be interfacing this LM35 with STM and will display the temperature in the LCD. I will be connecting LM35 with ST microelectronics port and then I will be displaying the current temperature.

What was the current temperature of this room? Let me tell you the temperature out here in Shillong is quite low; the outside temperature currently is 17 degrees. So, let us see what is coming here in our sensing unit.

(Refer Slide Time: 21:01)



So, this is the temperature sensor, this temperature sensor is LM35 temperature sensor. The flat end of this the left pin should be connected to 5 volt Vcc. The right end of this LM 35 will be connected to ground, and from the middle position it will be connected to the analog port A1. This analog port internally is connected to an AD converter.

You can see the digital value using any of the serial communication software like CoolTerm. So now, I will be doing the connection.

So, we have already done the calibration and as you already know that in this particular sensor that is LM35, 10 millivolt change in voltage will be equivalent to 1 degree change in temperature. So, we have done the calibration accordingly and now I will be dumping the code, which will display the current temperature of this particular room. So, let me connect it. The temperature what it is showing, you can see this, the temperature in

degree centigrade is 18.20 ... 19.40. It is little bit fluctuating Let me increase the temperature a bit and see what value comes.

You can see that it has become 21.60. If you rub this particular sensor a bit, the value changes or you can just blow a little bit in front of it to see the change in temperature. This is a simple experiment with LM35, where it is reading the temperature and is displaying it. You can see that the connection with LM35 is fairly straightforward and fairly simple.

The point is you need this calibration, the calibration step is really very important. In this calibration step, you need to read the current value of the sensor, what it is giving and you can see that in CoolTerm and then with respect to that value, what is the current temperature of this particular room. Accordingly, you can change your code to incorporate this particular scenario.

There are various other application of LM35, we have shown you one thing, which you can incorporate and do it for other experiments.

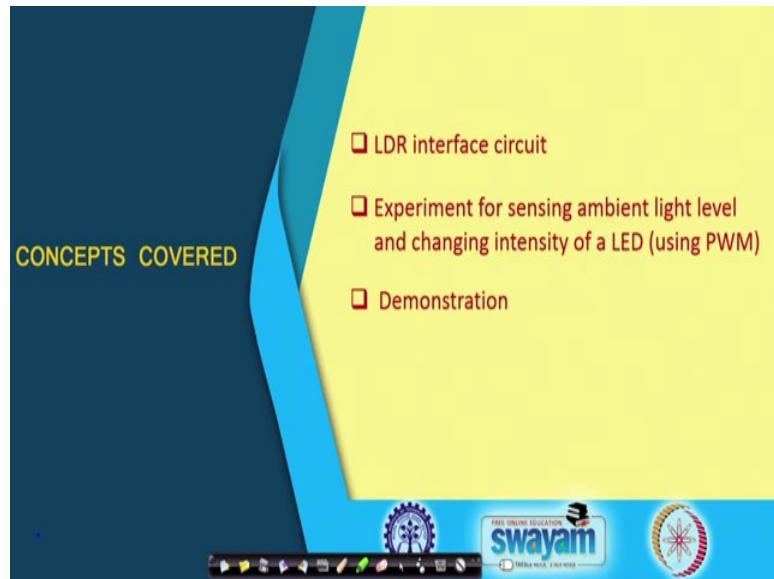
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institutes of Technology, Meghalaya

Lecture – 26
Experiment with LDR Light Sensor (Part I)

Welcome to lecture 26. In this lecture we will be interfacing LDR with the STM board. We will be first looking into what is LDR. And of course, the circuit diagram to perform the experiment with LDR. We will be showing two experiments with LDR.

(Refer Slide Time: 01:05)



As I said, we will discuss the LDR interfacing circuit. In this experiment we will be sensing the ambient light level. And, we will change the intensity of a LED through PWM port.

(Refer Slide Time: 01:31)

Light Dependent Resistor (LDR)

- A LDR is a variable resistor (a passive component) whose resistance value changes depending upon the amount of light falling on it.
 - More the amount of light, less will be the resistance and vice versa.
 - The variation of resistance with light intensity is non-linear.
- A simple resistance divider can generate an analog voltage that depends on R_{LDR} .

MCU

Analog Input

Voltage Output

5V

LDR

R1

The slide also features a video frame of a teacher speaking, a decorative banner with the text 'FREE ONLINE EDUCATION swayam', and a toolbar with various icons.

So what is light dependent resistor. It is a variable resistor that is a passive component, whose resistance value changes depending upon the amount of light falling on it. Whenever light falls on it, its resistance value changes. More the amount of light, less will be the resistance, and vice versa. So, if more light falls into the device, then its resistance decreases, and if less light falls on it; that means, if it is dark, then the resistance value will be high.

The variation of this resistance with light intensity is non-linear; like for LM35 we have seen that with 10 mV change there is one degree centigrade increase in temperature. So, there was a linear change; here that is not there. The variation of the resistance with light intensity is not linear. A simple resistance divider can generate an analog voltage that depends on the resistance of this LDR.

This is a typical diagram. This is an LDR and you see that one port is connected with 5V another port through a resistance is connected to ground. The voltage output from where we are taking is from here. We are connecting to the analog input this is how the connection goes.

(Refer Slide Time: 03:55)

LDR Interfacing

- First test the LDR by measuring its resistance.
- Consider for example, $R_{LDR} = 150 \text{ K}\Omega$ (no light), and $R_{LDR} = 10 \text{ K}\Omega$ (light)

$$V=IR$$

$V = 5V$

R_{LDR}

$R1$

V_{out}

$$\boxed{\begin{aligned} I &= V / (R_{LDR} + R1) \\ V_{out} &= I * R1 \\ &= V * R1 / (R_{LDR} + R1) \\ &= 5 * R1 / (R_{LDR} + R1) \end{aligned}}$$

FREE ONLINE EDUCATION **swayam**

Now, let us see this LDR interfacing. We need to first test the LDR by measuring its resistance. See in this room there is an ambient light, maybe in other room the intensity of light might be different.

So, we consider this, the resistance of LDR when there is no light is 150 kilo ohm and when there is light it is 10 kilo ohm, let us say this is what we are getting. But you have to actually take the reading using a multimeter. Let us say we have done that and we got a value in this range.

As shown, the output voltage from the LDR will be $5 * R1 / (R_{LDR} + R1)$.

Based on this equation, we will be looking into how to choose this particular $R1$ depending on the previous calculation that we have already made.

(Refer Slide Time: 06:25)

How to choose R1?

- Within the range of light intensity for some application, the variation in output voltage should be appreciable.
- Consider for example, $R_{LDR} = 150 \text{ K}\Omega$ (no light), and $R_{LDR} = 10 \text{ k}\Omega$ (light)

$$V_{out} = 5 * R1 / (R_{LDR} + R1)$$

R1	V_{out} (light)	V_{out} (no light)
1 k\Omega	0.45 V	0.03 V
5 k\Omega	1.67 V	0.16 V
10 k\Omega	2.50 V	0.31 V
20 k\Omega	3.33 V	0.56 V
50 k\Omega	4.17 V	1.25 V

So, within the range of light intensity for some application the variation in output voltage should be appreciable, because I want that variation should be more. When there is light and when there is no light we need to choose R1 in such a fashion that it is quite visible.

Consider for example, consider for example R_{LDR} is 150 kilo ohm for no light, and 10 kilo ohm when there is light. And we put it in this particular formula to calculate V_{out} for light and V_{out} for no light when $R1$ is 1 kilo ohm. The values are shown. Similar calculations are shown for different values of $R1$.

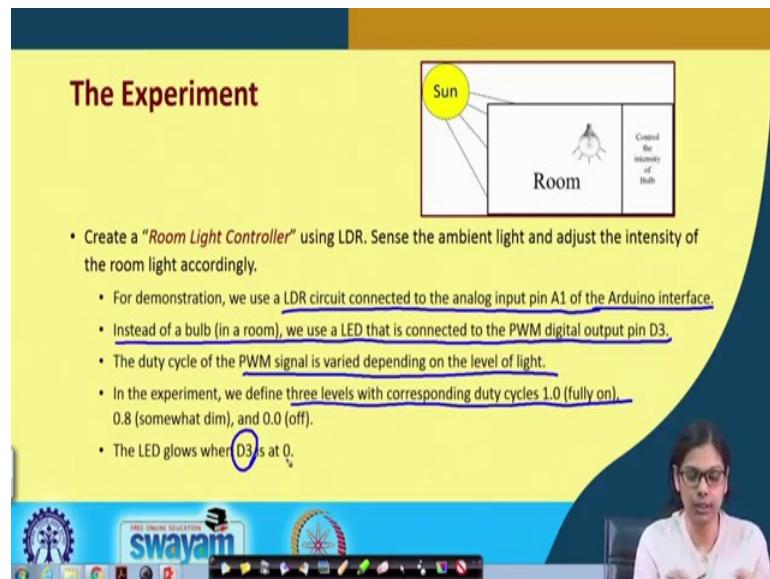
(Refer Slide Time: 09:07)

Nonlinear Variation of Resistance with Light

The graph illustrates the non-linear relationship between the resistance of an LDR and the illumination level. The Y-axis represents Resistance (Ω) on a logarithmic scale, with major ticks at 10 and 10^6 . The X-axis represents Illumination (LUX) on a logarithmic scale, with major ticks at 10^{-1} , 10^0 , and 10^3 . Three curves are plotted: a blue curve for 'Dark' conditions, an orange curve for 'Daylight', and a red curve for 'Sunlight'. All three curves show a significant decrease in resistance as the illumination level increases. The 'Dark' curve is the highest, followed by 'Daylight', and then 'Sunlight'.

You see the non-linear variation of the resistance with light. When it is dark its resistance is high, and its resistance decreases when there is light.

(Refer Slide Time: 09:41)



The Experiment

• Create a "Room Light Controller" using LDR. Sense the ambient light and adjust the intensity of the room light accordingly.

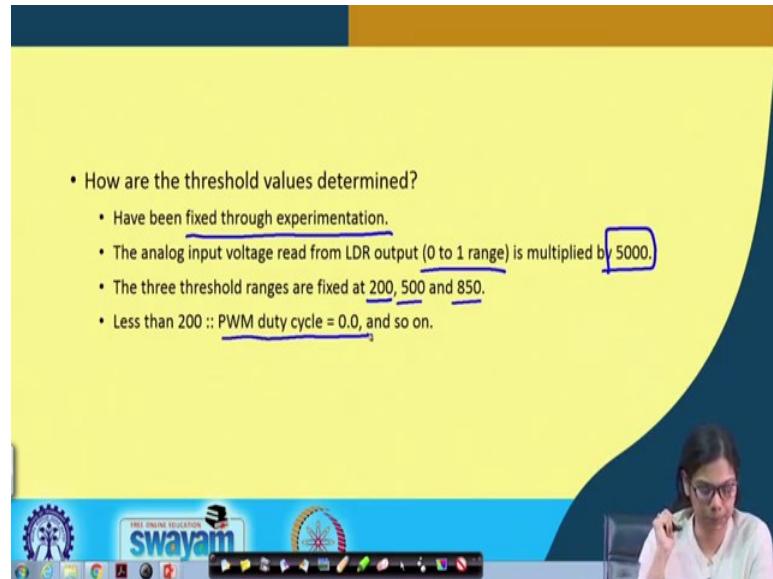
- For demonstration, we use a LDR circuit connected to the analog input pin A1 of the Arduino interface.
- Instead of a bulb (in a room), we use a LED that is connected to the PWM digital output pin D3.
- The duty cycle of the PWM signal is varied depending on the level of light.
- In the experiment, we define three levels with corresponding duty cycles 1.0 (fully on), 0.8 (somewhat dim), and 0.0 (off).
- The LED glows when D3 is 0.

The experiment that we will be doing today is, we will create a room light controller using LDR. We will sense the ambient light and adjust the intensity of the room light accordingly. So, how do we simulate here? For simulation purpose we have just use an LDR. We have used an external LDR that will be connecting with this STM board and depending on the ambient light the LED will glow fully if the light is little bit, if the light is very bright, then the LED will not glow, and if the light is little bright the LED will glow, but very lightly and so on.

Let us see for demonstration what we have done basically. We use an LDR circuit connected to the analog input pin A1. So, instead of a bulb as I said we are using an LED that is connected to the PWM digital output pin D2. We have already discussed specifically what is PWM pin, what are the functions that are attached to the PWM pin, what we can do through that PWM pin. In this case we will be using one of the PWM pins to glow the LED to any desired intensity. The duty cycle of the PWM signal is varied depending on the level of the light we are having.

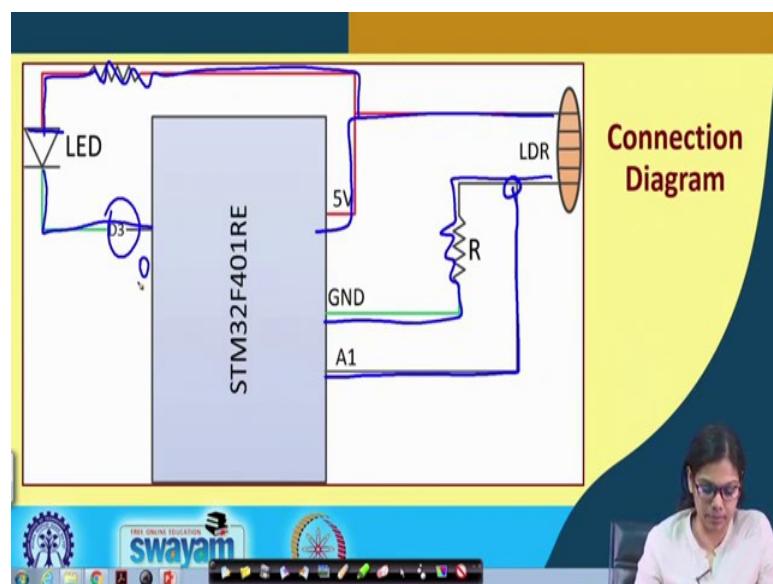
In the experiment we define three levels with corresponding duty cycle of 1, 0.8 and 0; these are the three things we have used. The LED will glow when the D3 port line is 0.

(Refer Slide Time: 12:01)



How are the threshold values determined? They have been fixed through experimentation. We did an experiment and found out the corresponding values. The analog input voltage read from LDR output is in the range of 0 to 1, which is multiplied by 5000, such that we can find out what is light and what is no light. The three threshold ranges are fixed at 200, 500 and 850. These are the three ranges that we have kept; and for less than 200 PWM duty cycle will be 0, and if it is 500 and 850 we change it accordingly.

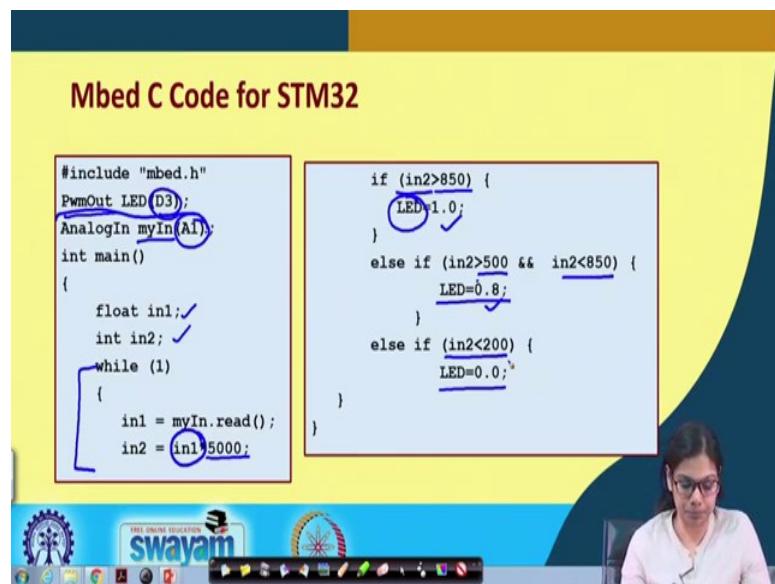
(Refer Slide Time: 13:13)



This is the circuit diagram. This is a single LDR, one end is connected to 5 volt. Another end of the LDR through a resistance is connected to ground. And the analog value that we are taking is from this point A1, and as I said the LED simulates a bulb. For the LED this is the anode that is connected through a resistance to 5 volt, and the cathode is connected to port D3. If this value is 0 then the LED will glow.

Now let us move to the code.

(Refer Slide Time: 14:33)



```
#include "mbed.h"
PwmOut LED(D3);
AnalogIn myIn(A1);
int main()
{
    float in1;
    int in2;
    while (1)
    {
        in1 = myIn.read();
        in2 = in1*5000;
        if (in2>850) {
            LED=1.0;
        }
        else if (in2>500 && in2<850) {
            LED=0.8;
        }
        else if (in2<200) {
            LED=0.0;
        }
    }
}
```

The code is self explanatory.

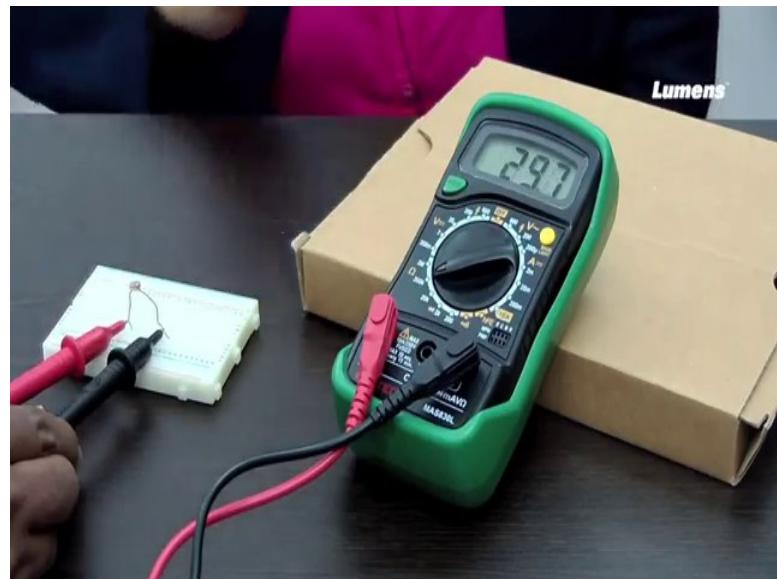
In the while(1) loop what we are doing? We are reading with the function myIn.read(), then I am multiplying this by 5000. If this value is greater than 850, then I am making that LED on; if it is greater than 5000 and less than 850 then I am glowing it, but not with full intensity (with duty cycle 0.8); and if it is less than 200 I am making it off. So, these are the three levels that we have made for this LED.

What we are outputting? We are outputting to this LED that is the PWM output. For the three levels the brightness will vary.

Now, I will be showing you the experiment with LDR. The experiment that I will be showing is that, depending on ambient light we will see that how we can change the intensity of a LED.

We have taken an LDR and an LED. The LED will glow in three levels depending on ambient light.

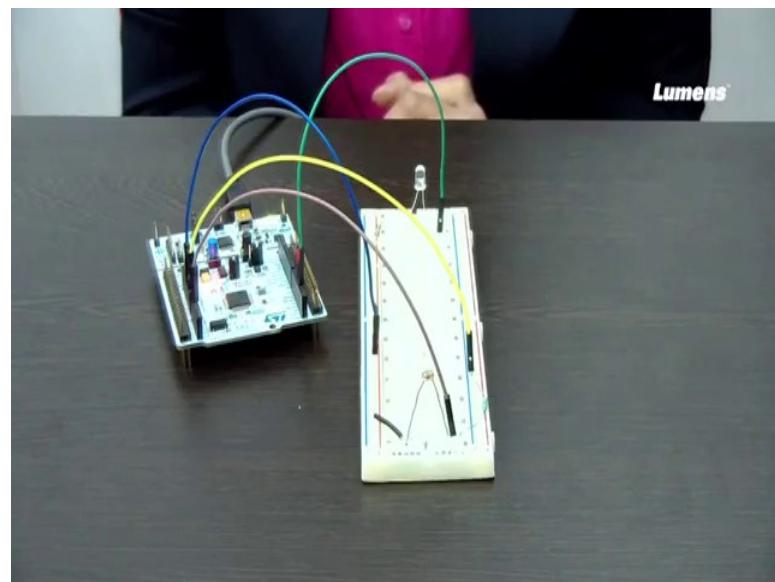
(Refer Slide Time: 09:15)



So, this is an LDR and this is a multimeter. Now we will see that what value this will give when we connect using this particular ambient light that we have. What value it is giving? It is giving around 29. I will put my hand, you see the resistor value changes to 130 or so, again I take out it will settle down 29.

Now, I will make this ambient light of this room little bit dimmer. Now you see that the value initially I was getting was 29 when there was full light, and now we are getting a value 41.7 when I put my hand there. Similarly resistance value changes.

(Refer Slide Time: 22:27)



After making all the connections, I will dump the code that I have already discussed with you. Now the intensity of LED changes with light falling on LDR.

In this experiment we basically shown you how you will connect LDR with STM board and depending on the ambient light, how you can actually do some kind of operation.

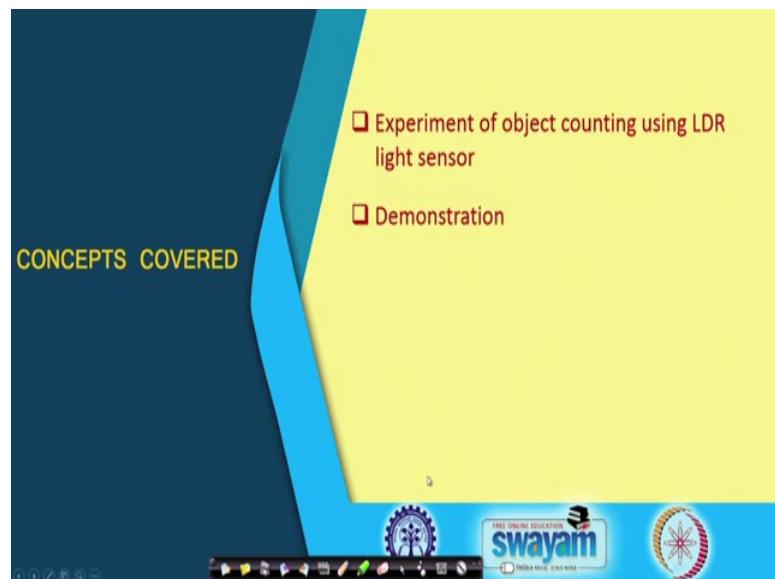
Thank you.

Embedded System Design with Arm
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 27
Experiment With LDR Light Sensor (Part II)

Welcome to lecture 27. In the previous lecture I have discussed about LDR and have shown you that depending on the ambient light how we can change the intensity of an LED. In this experiment we will be using LDR again, we will be doing one more experiment in which we will be counting the number of person present in this room. The assumption is that there is an entry point and there is a different exit point. So, the person coming from one door will come in, and the person will be going out from the other door.

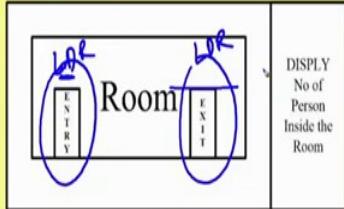
(Refer Slide Time: 01:19)



(Refer Slide Time: 01:29)

The Experiment

- Design a counter (using LDR and LED's), which will display the number of persons present in a room on a 7-segment LED display



- [Hint: Use two LDRs to detect direction of motion i.e., whether a person is entering or leaving]

DISPLAY
No of Person Inside the Room

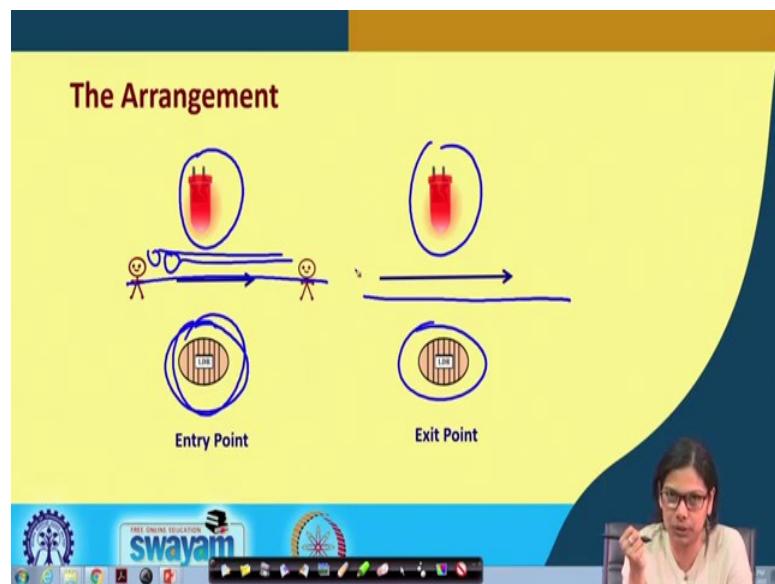
swayam



So, this is the entry point this is the exit point. In this entry point there will be an LDR and in this exit point there will be another LDR, and the number of person present in a room will be displayed in a 7-segment display. Instead of displaying it in LCD, in this experiment we will be displaying it in a 7-segment display. We use 2 LDRs to detect the direction of motion ,that is whether a person is entering or leaving.

We do not have to take into consideration how they are entering, only thing is that if it crosses this LDR then a person has entered, if it crosses this LDR that means the person has exited.

(Refer Slide Time: 02:37)



This is how the arrangement goes.

Whenever there is an interruption from here; that means, a person has entered. Whenever there is an interruption from here, a person has exited. So, these are the 2 scenarios. LDR can be used for many other applications also; you can also use it to count the number of tablets entering into a bottle. So, in that case you have to do a similar arrangement. You can keep this LDR in the front end of the tube and you can put the tablets one by one like this, and each time there is an interruption the counter will get incremented.

Here we are implementing a person counter.

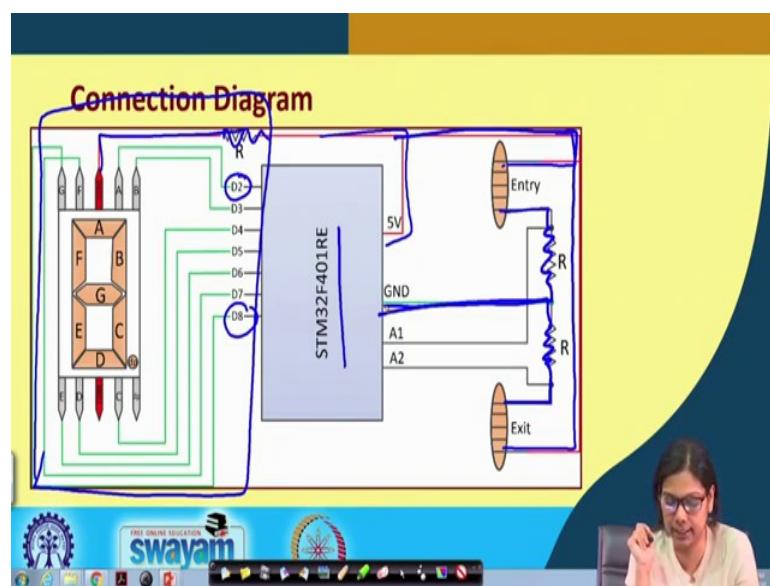
(Refer Slide Time: 04:15)

Basic Concept

- Initialize counter to 0.
- If the light intensity read by LDR from entry point decreases, increment the counter.
- If the light intensity read by LDR from exit point decreases, decrement the counter.

What is the basic concept here? We initialize a counter to 0. Whenever there is an interruption, the light intensity decreases that increments the counter. If there is a change in the LDR value from the entry point, each time this happens we increment the counter. If the light intensity read by the LDR from the exit point decreases again we have another point that is the exit point; if it goes from the entry point it increases; if it goes from the exit point, then decrement the counter.

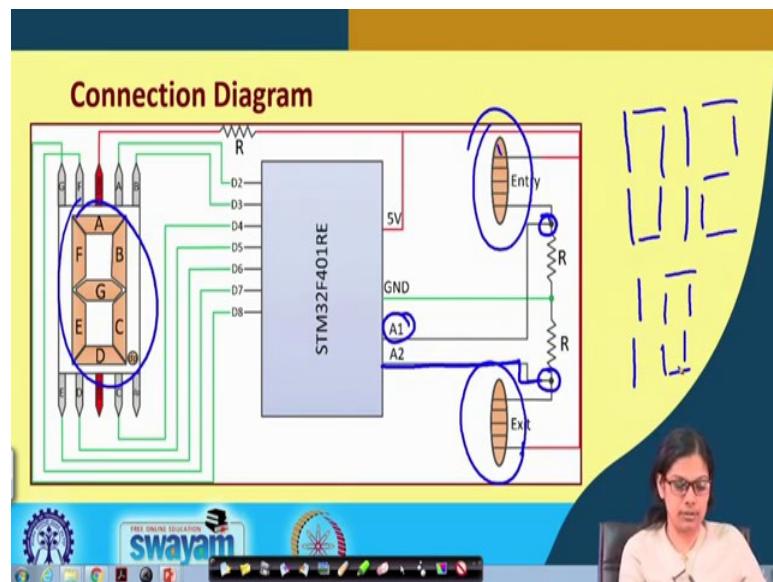
(Refer Slide Time: 06:21)



This is the connection diagram that is fairly straightforward. This part of the connection diagram we have already discussed, where all the segments of this display are connected from pin number D2 to D8 of the STM. From the common point through a resistance, the common anode is connected to 5 volt. Whenever there is an interruption this display should display the number of person present in a particular room.

Now let us see the connection of this LDR; one end of the LDR is connected to 5 volt directly. And another end is connected through a resistance to ground.

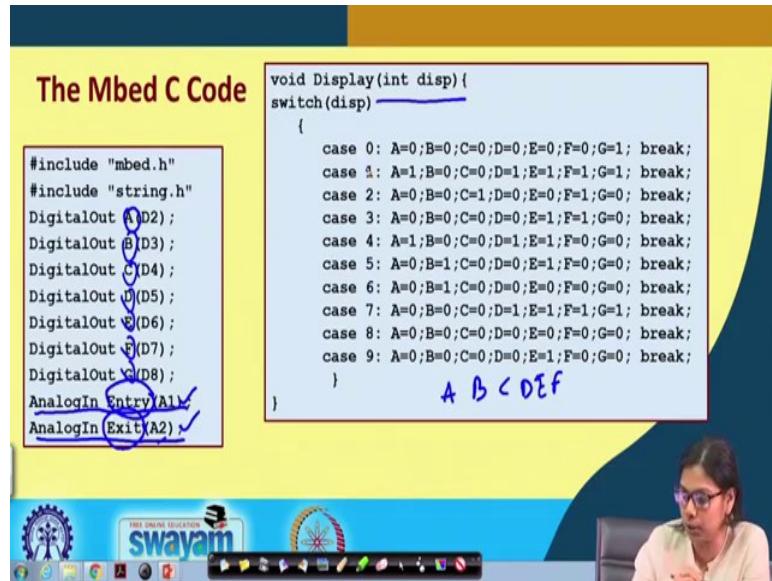
(Refer Slide Time: 07:53)



From the entry point it is connected to port A1. And from exit point it is connected to port A2, this is all about the connection.

Now what will happen whenever there is an interruption in this particular LDR? This display will get incremented, 1, 2, 3, etc. Now somebody exits the count will decrease.

(Refer Slide Time: 09:29)



The Mbed C Code

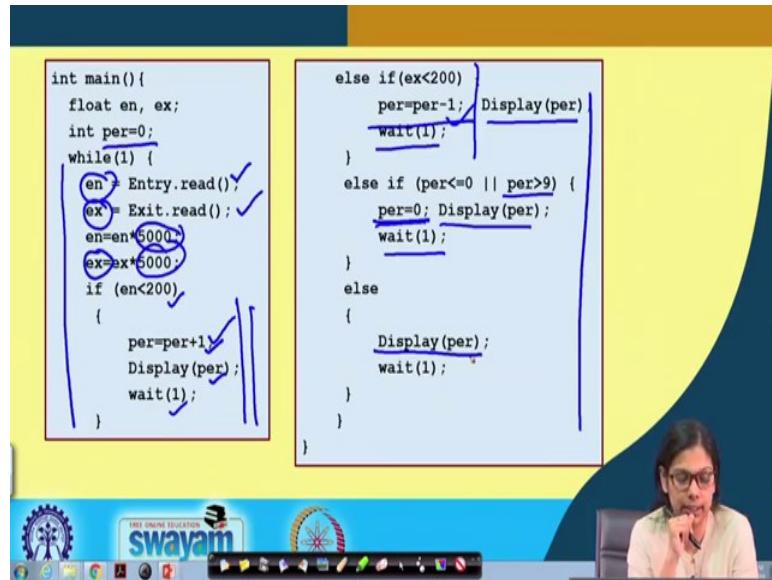
```
#include "mbed.h"
#include "string.h"
DigitalOut D2;
DigitalOut D3;
DigitalOut D4;
DigitalOut D5;
DigitalOut D6;
DigitalOut D7;
DigitalOut D8;
AnalogIn Entry(A1);
AnalogIn Exit(A2);
```

```
void Display(int disp){
switch(disp){
case 0: A=0;B=0;C=0;D=0;E=0;F=0;G=1; break;
case 1: A=1;B=0;C=0;D=1;E=1;F=1;G=1; break;
case 2: A=0;B=0;C=1;D=0;E=0;F=1;G=0; break;
case 3: A=0;B=0;C=0;D=0;E=1;F=1;G=0; break;
case 4: A=1;B=0;C=0;D=1;E=1;F=0;G=0; break;
case 5: A=0;B=1;C=0;D=0;E=1;F=0;G=0; break;
case 6: A=0;B=1;C=0;D=0;E=0;F=0;G=0; break;
case 7: A=0;B=0;C=0;D=1;E=1;F=1;G=1; break;
case 8: A=0;B=0;C=0;D=0;E=0;F=0;G=0; break;
case 9: A=0;B=0;C=0;D=0;E=1;F=0;G=0; break;
}
}
```

A B C D E F

The code is again self-explanatory. We have 2 analog ports, one is A1, other is A2.

(Refer Slide Time: 10:45)



```
int main(){
float en, ex;
int per=0;
while(1) {
en=Entry.read();
ex=Exit.read();
en=en*5000;
ex=ex*5000;
if (en<200)
{
per=per+1;
Display(per);
wait(1);
}
else if(en>200)
per=per-1;
Display(per);
wait(1);
else if (per<=0 || per>9)
per=0;
Display(per);
wait(1);
else
Display(per);
wait(1);
}
}
```

Now, let us see the main program. The number of person what we have made initially is 0. In the while(1) loop, Entry.read() will give some value for the entry point, and Exit.read() will give the value from the exit point. Both the values are multiplied with 5000.

If "en" is less than 200, then number of persons is incremented by 1. If the resistance decreases beyond 200, then there is an interruption, and we have incremented it, we send

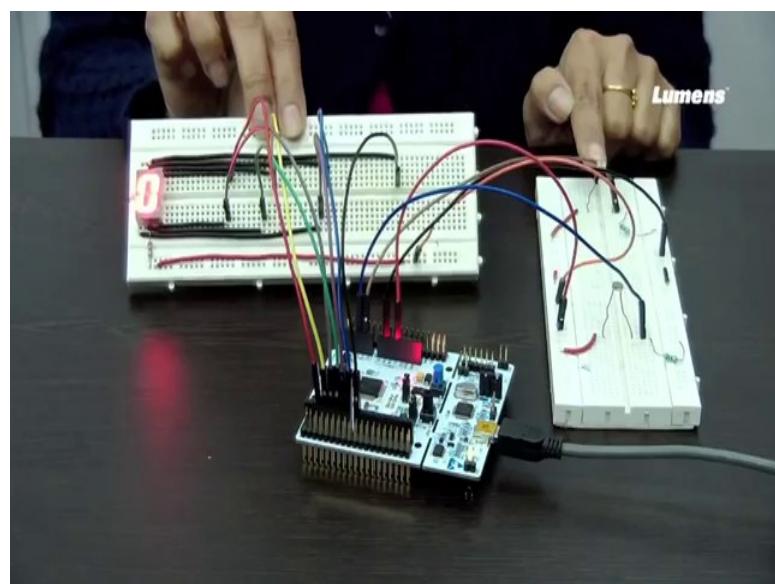
it to Display function and wait for 1 second. Else if we have to also check from the exit point. If “ex” is less than 200, then the person count is decremented by 1k.

After this we will be showing you the demonstration of the same code that I have discussed just now.

I will now be showing you the experiment in which you can count the number of persons present in a particular room.

You have 2 LDRs put up in one in entry point and another in exit point. Whenever there is an interruption the counter will get increased if somebody is crossing the entry point, and if somebody is crossing the exit point that particular number will get decremented. At any particular time the number of persons that are present in a room will be displayed in a 7 segment unit.

(Refer Slide Time: 16:13)



This is how you have to put the 2 LDRs; let us say this is the entry point and this one is the exit point. Now as I will display this in a 7 segment LED, we already know how to connect with a 7 segment display unit. All the connections are made.

So, this one is my entry and this one is my exit. Let us see how it works. Let me connect it and dump the code; initially it is displaying 0 count. Now as I said this LDR is my

entry point, let us see if I interrupt what happens. It has incremented by 1. I will make another interruption, it has incremented to 2, another interruption to 3.

Let us see what happens if there is an interruption at the exit point, it has become 2, then it has become 1. I will again make some entry. It works correctly.

I hope this experiment with LDR is clear to you all.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 28
Experiment with Speaker

Welcome to lecture 28. In this lecture, we will be showing you some experiments with Speaker. We will be interfacing a speaker with the STM board and will generate 4 different kinds of tones. Firstly, we will look into how the connection diagram will be. PWM port we have already discussed in detail in week 3. I will be using some of the functions that have been already discussed, which I will be using along with the speaker and then now I will be showing you how we can make different kinds of signals, the different kinds of sound through a speaker by changing the code.

(Refer Slide Time: 01:14)

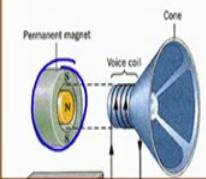


Firstly, we will show the circuit and then we will look into the various experiments with speaker, and finally I will demonstrate.

(Refer Slide Time: 01:27)

How does a speaker work?

- Speakers work by converting electrical energy to mechanical energy (motion).
- The mechanical energy compresses air and converts the motion into sound energy or sound pressure level.



FREE ONLINE EDUCATION **swayam**

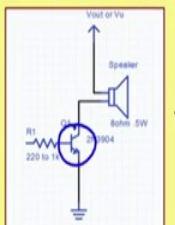
So, how does a speaker work? It has been already discussed in previous week. A speaker works by converting electric energy to mechanical energy. The mechanical energy compresses the air, and converts the motion into sound energy.

Inside it there is a permanent magnet, and there is a coil and diaphragm through which it generates some kind of sound.

(Refer Slide Time: 02:21)

How to Interface a Speaker?

- Any waveform in the audio frequency range from an output port can drive the speaker.
 - May require an amplifier circuit to generate adequate power for the electromagnet.
- A simple transistor based amplifier circuit:



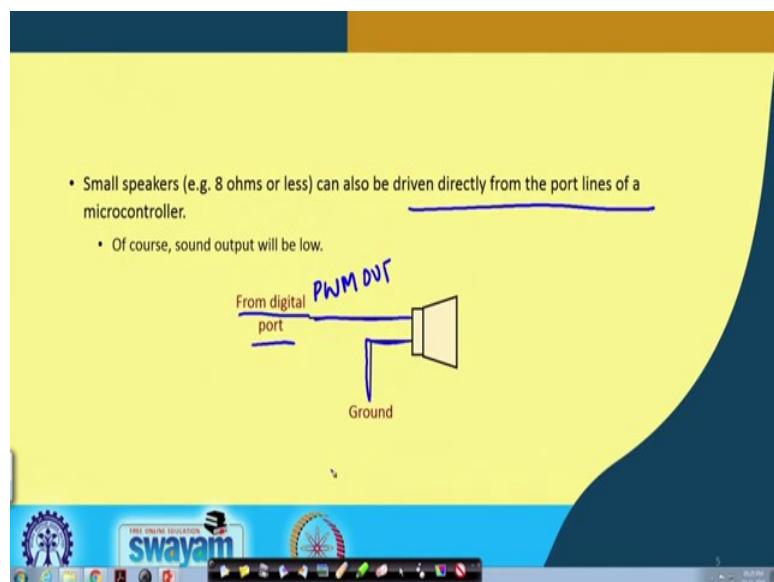
FREE ONLINE EDUCATION **swayam**

How do we interface a speaker? If you think of any waveform in audio frequency range from an output port can drive the speaker, but sometimes we see that we may require an

amplifier circuit to generate the adequate power for the electromagnet that is inside the speaker.

This is a typical circuit diagram, a simple transistor based amplifier circuit. Although in our case, we have not used any kind of circuit, but it might be required to use such kind of circuit, when you interface a speaker.

(Refer Slide Time: 03:29)



We have interfaced a small speaker, which we have directly connected it to a port line. We have connect to a PWM port line. These are the two ends of the speaker; one is connected to ground, one will be connected to PWMout. Of course, we will not be able to generate a very high volume sound, but a reasonable level of sound, which you can hear.

(Refer Slide Time: 04:23)

Experiment 1

- Produce a fixed tone on the speaker by generating a square wave of a particular frequency, and use it to drive the speaker.
 - Audible sound is in the frequency range 20 Hz to 20 KHz.
 - In this experiment, we generate a tone with frequency 250 Hz.
 - We can use a pulse width modulated (PWM) port to generate the speaker control signal, by specifying:
 - a) The time period of the generated waveform (i.e. 1 / frequency)
 - b) The duty cycle of the generated waveform $\frac{T_{on}}{T_{on} + T_{off}}$

FREE ONLINE EDUCATION **swayam** for all

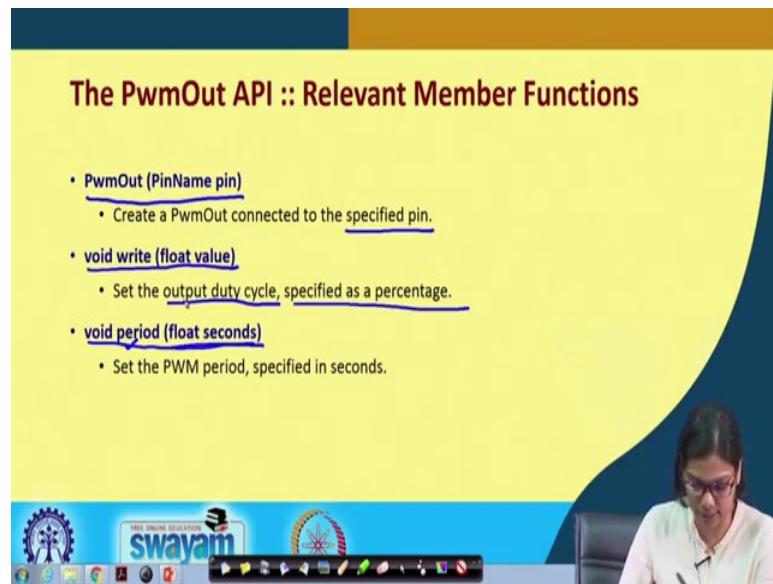
340

As I told you we will be doing a couple of experiments.

In the first experiment, we will be generating a fixed tone on the speaker by generating a square wave of a particular frequency, which we will use it to drive the speaker. We all know that the audible frequency range is 20 hertz to 20 kilohertz.

In this experiment we generate a frequency of 250 hertz, which will be audible to us. We can use the PWM port to generate the speaker control signal by specifying the following. We need to specify the time period of the generated waveform, and the duty cycle of the waveform.

(Refer Slide Time: 05:43)



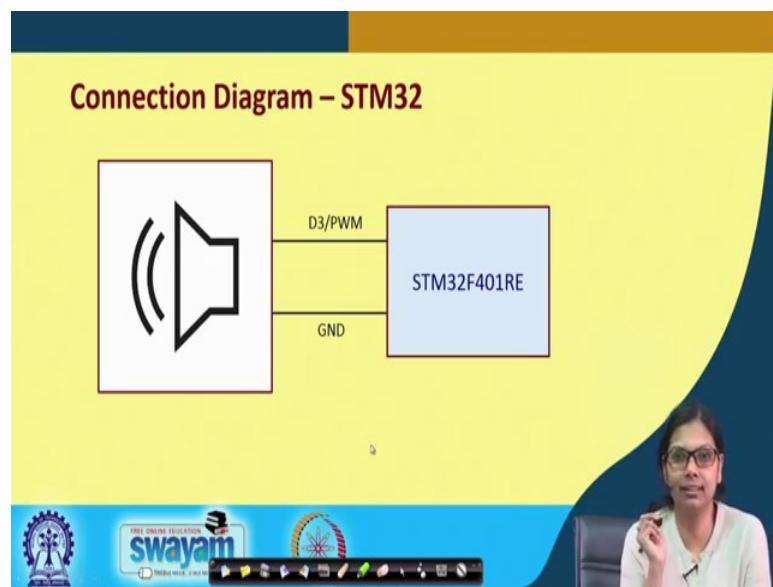
The PwmOut API :: Relevant Member Functions

- PwmOut (PinName pin)
 - Create a PwmOut connected to the specified pin.
- void write (float value)
 - Set the output duty cycle, specified as a percentage.
- void period (float seconds)
 - Set the PWM period, specified in seconds.

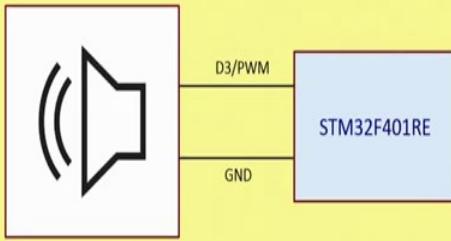
These functions we have already discussed in previous weeks. PwmOut creates a PwmOut signal connected to the specified pin.

In void write(), it will set the output duty cycle specified as a percentage, the percentage time it is on. Similarly, void period() can set the PWM period in seconds. There are many more functions, which have been discussed earlier in detail.

(Refer Slide Time: 07:14)



Connection Diagram – STM32



Now, this is the circuit diagram, you will see that when I demonstrate I will simply connect one end of the speaker to ground and another end with PWM port that is the D3.

(Refer Slide Time: 07:44)

Mbed program for Speaker (beep)

```
#include "mbed.h"
DigitalOut speaker(D3);
int main() {
    while(1) {
        speaker = 0;
        wait (0.002); // Wait 2 msec
        speaker = 1;
        wait (0.002); // Wait 2 msec
    }
}
```

This program does not use PWM. It simply outputs 0 and 1 alternately, with time period 4 msec, i.e. frequency of 250 Hz.

The slide is from the Swayam platform, as indicated by the logo at the bottom.

Now, I will be showing you a series of programs. The first program will generate a fixed tone of frequency 250 hertz. We are outputting speaker with 0 with a weight of 0.002 second, which is 2 milliseconds, and speaker is set to 1 with weight of 0.002 second, which is 2 millisecond. This code generates a square wave on frequency 250 hertz.

We will demonstrate this experiment at the end of this lecture.

(Refer Slide Time: 09:28)

Experiment 2

- Generate a two-frequency tone on the speaker, alternating between 333 Hz and 455 Hz; playing each frequency tone for 0.5 second.
 - Play 333 Hz tone for 0.5 second (time period = 3.0 msec).
 - Play 455 Hz tone for 0.5 second (time period = 2.2 msec).
- Repeat the process in a loop.

Handwritten calculations on the slide:

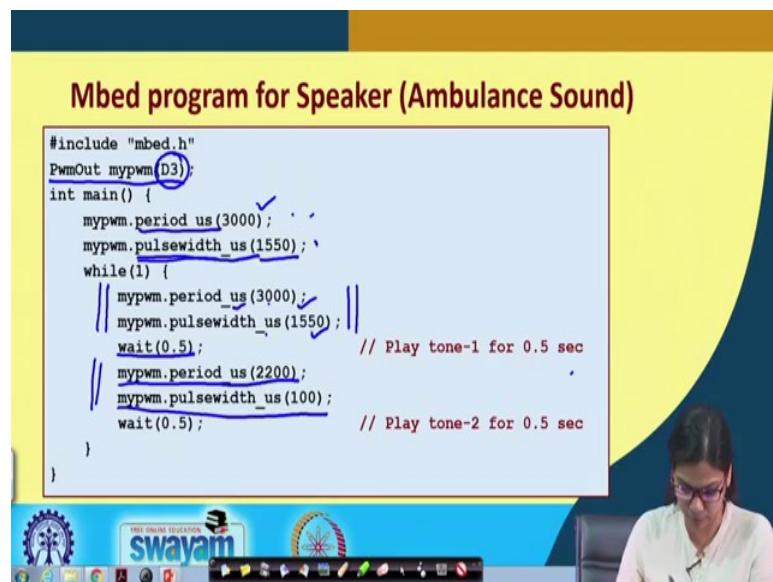
$$\frac{1}{333} = 3 \text{ ms}$$
$$\frac{1}{455} = 2.2 \text{ ms}$$

The slide is from the Swayam platform, as indicated by the logo at the bottom.

In this experiment, we will generate a 2-frequency tone on the speaker, alternatively between 333 hertz and 455 hertz, for 0.5 second each. If it is 333 hertz the time period will be 1 divided by 333; 1 divided by 333, which comes to 3 milliseconds.

For 455 hertz, 1 divided by 455 comes to 2.2 milliseconds.

(Refer Slide Time: 10:39)

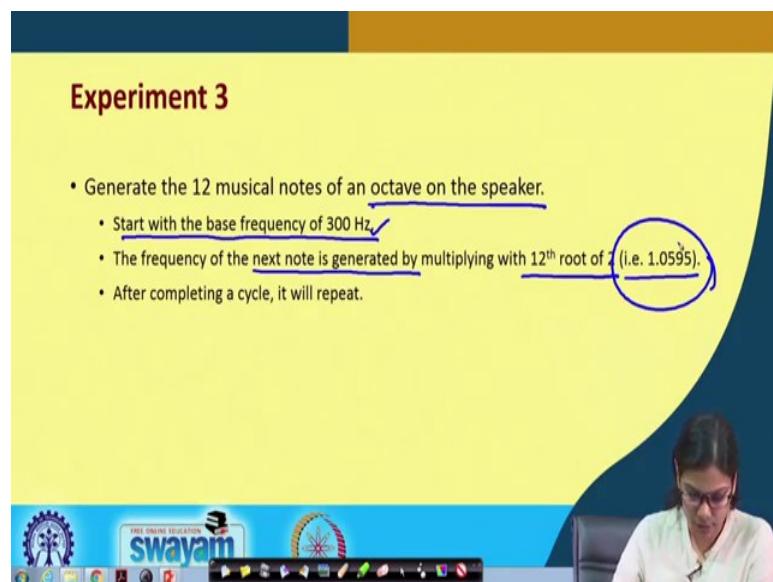


```
#include "mbed.h"
PwmOut mypwm(D3);

int main() {
    mypwm.period_us(3000);
    mypwm.pulsewidth_us(1550);
    while(1) {
        mypwm.period_us(3000);
        mypwm.pulsewidth_us(1550);
        wait(0.5); // Play tone-1 for 0.5 sec
        mypwm.period_us(2200);
        mypwm.pulsewidth_us(100);
        wait(0.5); // Play tone-2 for 0.5 sec
    }
}
```

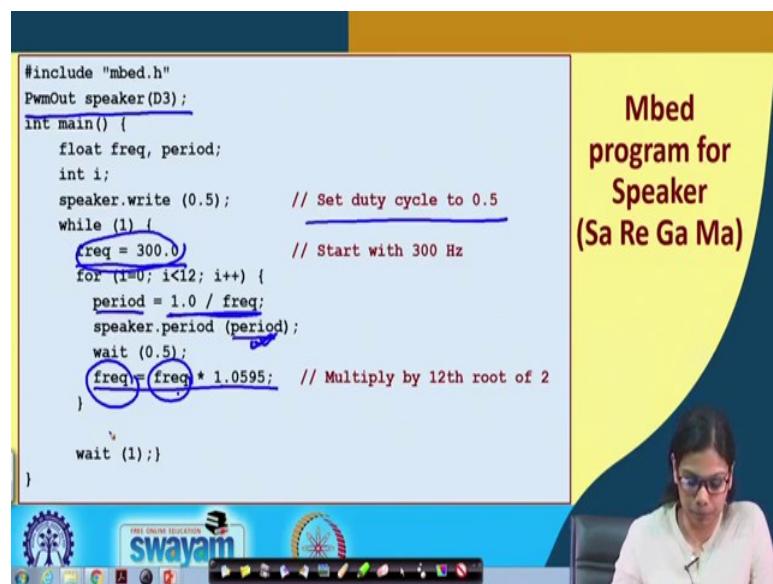
Let us see the code. The code is self-explanatory.

(Refer Slide Time: 12:34)



In the next experiment we will generate 12 musical notes of an octave on the speaker. Basically, it is Sa Re Ga Ma Pa Da Ni Sa. We start with the base frequency, let us say 300 hertz, and then the frequency of the next tone is generated by multiplying it with 12th root of 2, which is 1.0595. This repeats.

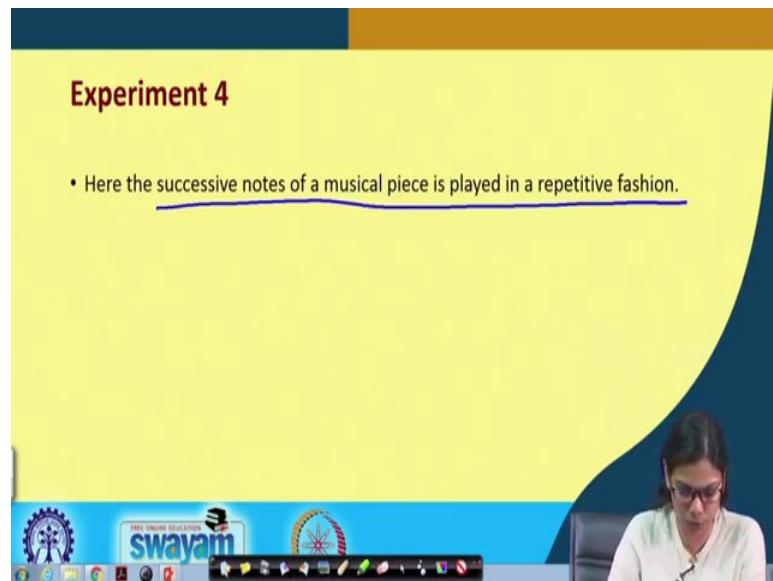
(Refer Slide Time: 13:26)



```
#include "mbed.h"
PwmOut speaker(D3);
int main() {
    float freq, period;
    int i;
    speaker.write (0.5); // Set duty cycle to 0.5
    while (1) {
        freq = 300.0; // Start with 300 Hz
        for (i=0; i<12; i++) {
            period = 1.0 / freq;
            speaker.period (period);
            wait (0.5);
            freq = freq * 1.0595; // Multiply by 12th root of 2
        }
        wait (1);
    }
}
```

This code is also straightforward.

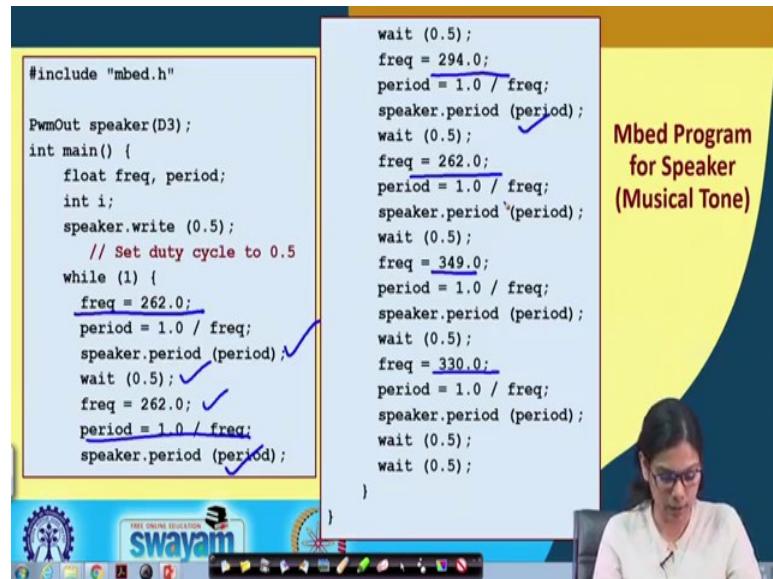
(Refer Slide Time: 15:58)



In the next experiment, we will be playing the notes of a musical piece.

We have selected few notes (frequencies) that we will play for certain periods to generate that particular tone.

(Refer Slide Time: 15:33)



```
#include "mbed.h"

PwmOut speaker(D3);

int main() {
    float freq, period;
    int i;
    speaker.write(0.5);
    // Set duty cycle to 0.5
    while (1) {
        freq = 262.0;
        period = 1.0 / freq;
        speaker.period(period);
        wait(0.5);
        freq = 262.0;
        period = 1.0 / freq;
        speaker.period(period);
        wait(0.5);
        freq = 294.0;
        period = 1.0 / freq;
        speaker.period(period);
        wait(0.5);
        freq = 349.0;
        period = 1.0 / freq;
        speaker.period(period);
        wait(0.5);
        freq = 330.0;
        period = 1.0 / freq;
        speaker.period(period);
        wait(0.5);
        wait(0.5);
    }
}
```

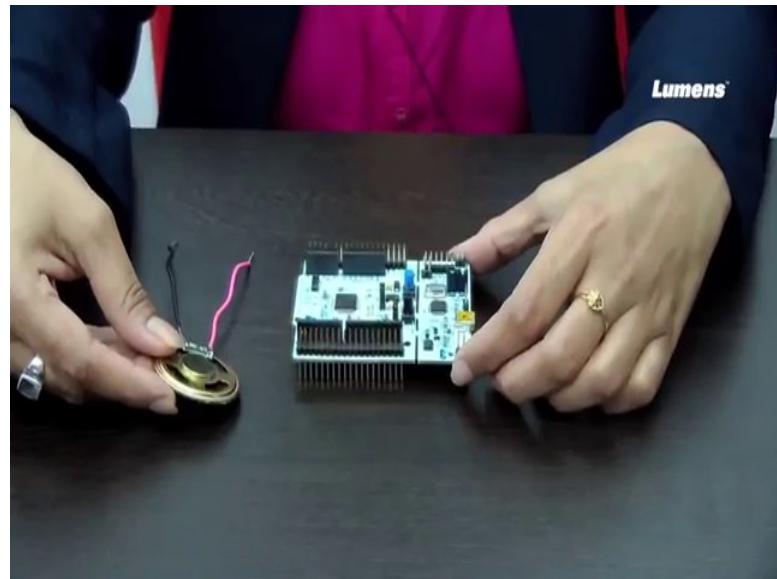
Mbed Program for Speaker (Musical Tone)

So, so this is basically all about that I have to tell you regarding speaker and then I will move on and I will show you how do I interface this, how do I interface this particular 4, all the 4 codes using a speaker and a microcontroller board. So now, I will be showing you the 4 codes on speaker that I have discussed just now ok.

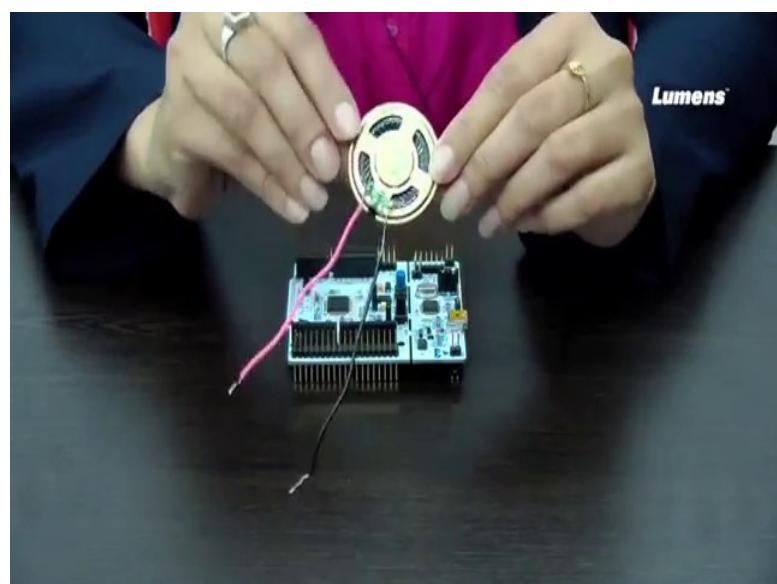
So, those 4 codes what they are doing? In the first code, we will just make some random beep sound for a longer period. What the next code is doing? It is making a sound like siren. Then the next code plays 12 octaves one by one.

Let us look into the circuit.

(Refer Slide Time: 19:06)

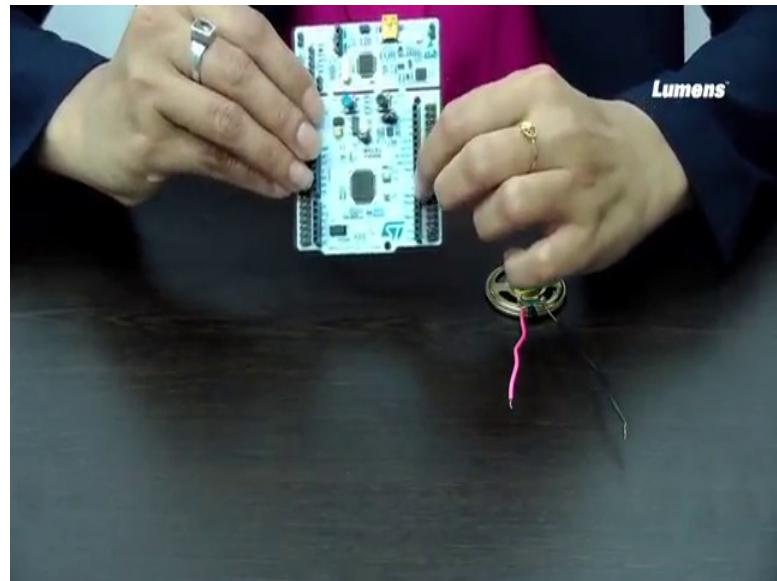


(Refer Slide Time: 19:11)



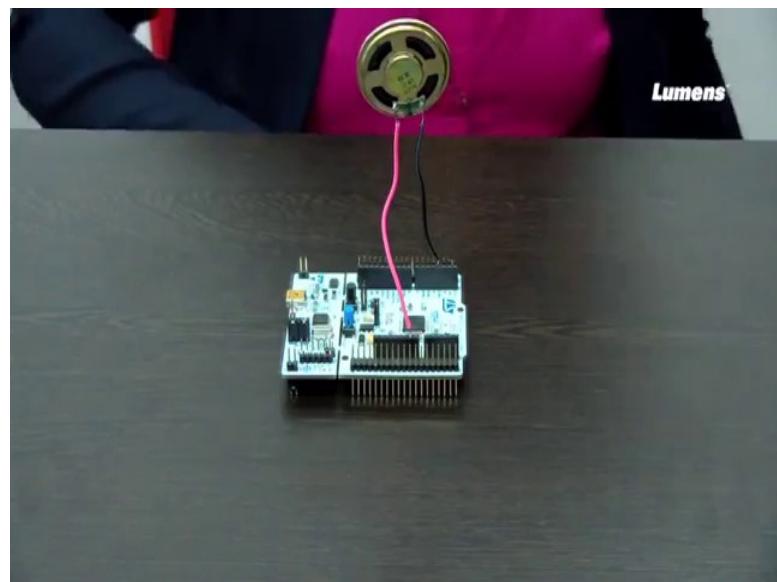
This is a speaker that we discussed. What I will do is, connect one of these ends to ground and the other end to the PWM port.

(Refer Slide Time: 19:47)



In the board, there are many PWM ports, out of which I will be using the D3 port. Then I will be dumping the four codes as I said. Let me connect the speaker now.

(Refer Slide Time: 20:38)



I will be connecting one end to the ground and another end to D3.

The first code will generate a constant sound, which you can hear.

I will now be dumping the second code. Basically this is the sound of a siren.

Now, we move on with the third one, which is basically Sa Re Ga Ma sound.

Here next, I will dump the last code which plays the happy birthday tone

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 29
Experiment with Microphone

Welcome to lecture 29. In this lecture I will be showing you some experiments with microphone. We know how what a microphone does. I will show you two experiments. In one experiment it will display the number of claps.

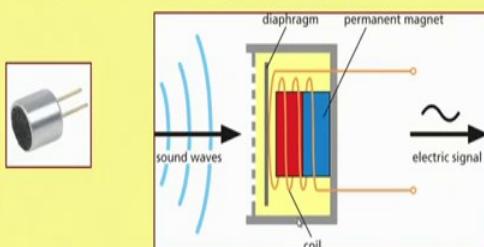
(Refer Slide Time: 00:55)



(Refer Slide Time: 01:11)

What is a Microphone?

- A microphone is a type of transducer, that converts acoustical energy (sound waves) into electrical energy (the audio signal).

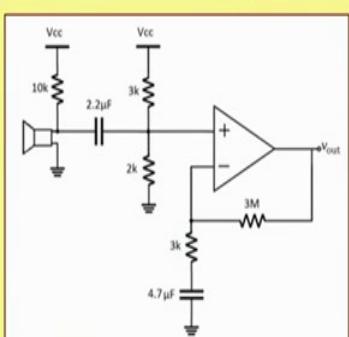


The diagram illustrates the internal mechanism of a microphone. It shows a cross-section of the device with a diaphragm on top, a permanent magnet in the middle, and a coil at the bottom. Sound waves enter through the diaphragm, causing it to vibrate. This vibration interacts with the permanent magnet, creating a changing magnetic field that induces an electric signal in the coil. The coil is connected to an output terminal labeled 'electric signal'.

Microphone is a type of transducer that converts acoustical energy into electrical energy. We see here there are sound waves coming, and there is a diaphragm here and a permanent magnet. Some operation happens here and these sound waves actually generate some kind of electrical signal here.

(Refer Slide Time: 01:51)

Typical Microphone Interfacing Circuit



The circuit diagram shows a typical microphone interfacing setup. A microphone is connected to the non-inverting input of a operational amplifier (op-amp). The inverting input is connected to ground through a 3MΩ resistor. A feedback path is formed by a 3kΩ resistor and a 4.7μF capacitor connected to ground. The output of the op-amp is labeled V_{out} . The power supply is indicated by V_{cc} and V_{ee} . Various resistors (10kΩ, 3kΩ, 2kΩ) and capacitors (2.2μF, 2kΩ) are used for biasing and filtering.

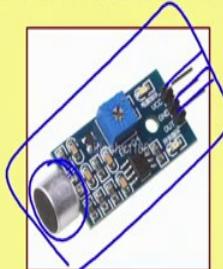
This is a typical microphone interfacing circuit. If you see this you have a number of connections here with some resistance with ground and all other things, but what

generally is done is that the whole set of interfacing that is shown here is actually put inside this particular chip.

(Refer Slide Time: 02:13)

Microphone Module with Interface

- For ease of interfacing, microphone with driver circuit is available as a module.



- Can work with power supply of 3.3V to 5.0V.
- Digital output (0 or 1), depending on whether sound is detected or not.
- Can be directly interfaced with the microcontroller.

SWAYAM



So, this is the microphone, if you use a single microphone you have to connect it with the circuit as I have shown you in the previous slide for the connection. But if you use this particular microphone it comes with driver circuit as a whole module. And there is a Vcc, GND and an output pin connection. It can work with power supply of 3.3 volt to 5 volt, and the digital output is 0 or 1 depending on whether the sound is detected or not. We can directly connect it to the microcontroller.

(Refer Slide Time: 03:22)

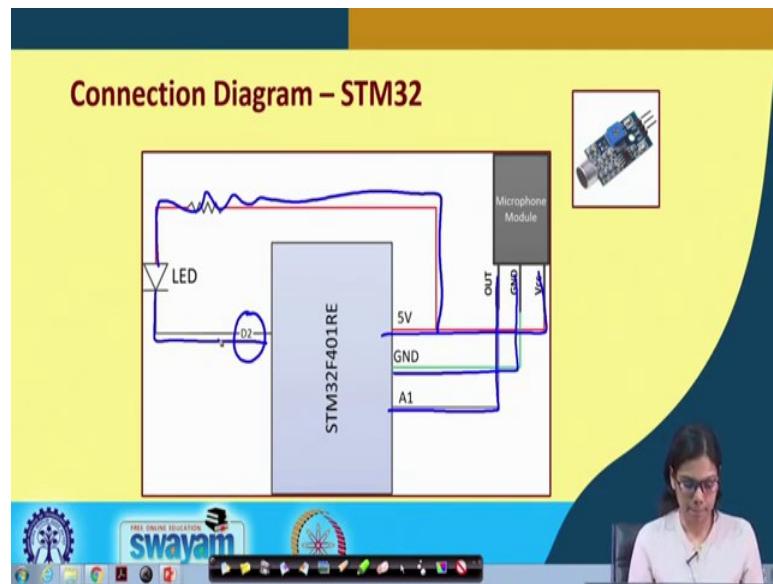
Experiment 1

- Implement a clap switch, where a LED can be turned ON using the sound from a clap.
 - The microphone circuit is interfaced to the AnalogIn pin A1.
 - The AnalogIn object converts the input voltage to floating-point numbers from 0.0 to 1.0.
 - Through experimentation, we can determine the threshold value, which would depend on the noise of the environment.
 - The microphone module used, however, generates a 2-level digital output.
 - A LED is connected to the DigitalOut pin D2, where the LED glows if the D2 pin is at 0.
 - The LED turns on for 3 seconds, and then turns off.

Soon I will come to the experiment because the connection is fairly straightforward here. I will be implementing a clap switch, where an LED can be turned on or off using the sound from a clap. Here the microphone circuit is first interfaced with analog pin A1, this AnalogIn object converts the input voltage to a floating point number between 0 and 1, we have already seen that. And through experimentation we can determine this threshold value when you clap what value you get. This would depend on the noise of the environment of course, the microphone module used.

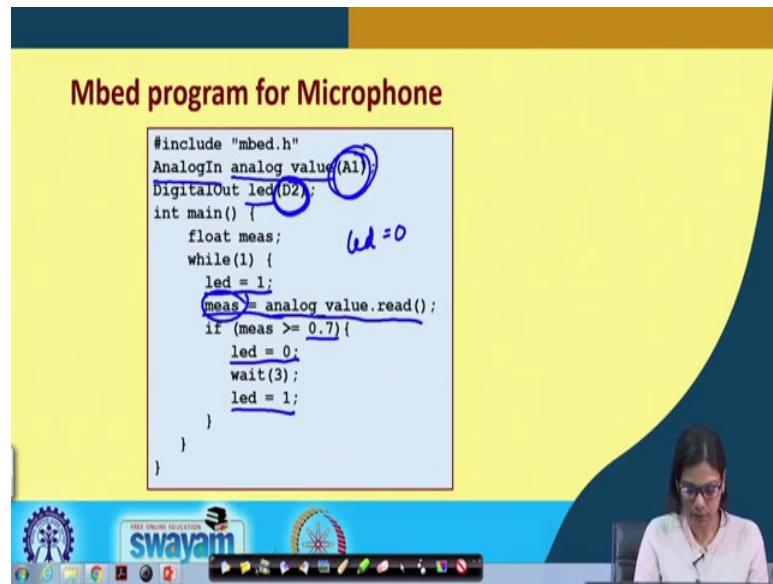
An LED is connected to the digital out pin D2, where the LED will glow if D2 pin is at 0. Whenever it detects sound through clap, we make the LED turn on for 3 seconds and then it is turned off.

(Refer Slide Time: 04:51)



This is the connection diagram. So, this OUT is connected with the analog pin A1, this GND is connected to the ground, and this is connected to 5 volt. And for this LED the anode is connected through a resistance to Vcc, and the cathode is connected to pin D2.

(Refer Slide Time: 05:23)

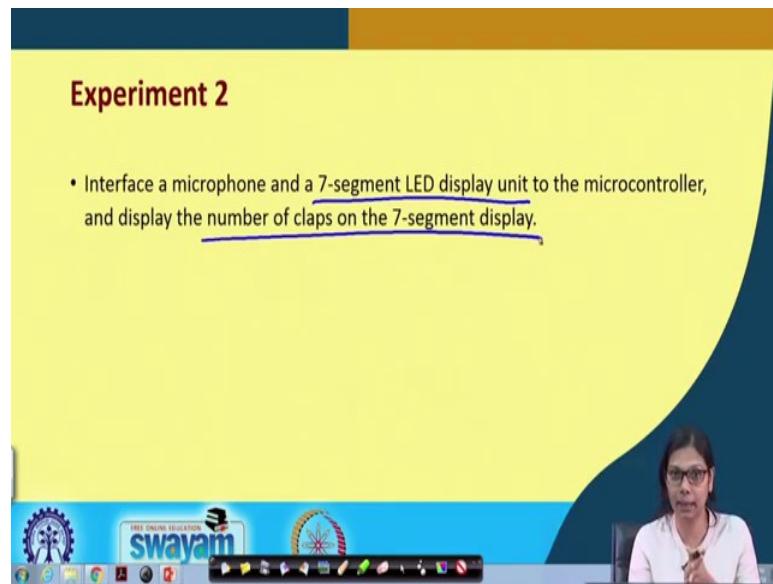


This is the program that is self-explanatory.

(Refer Slide Time: 06:55)

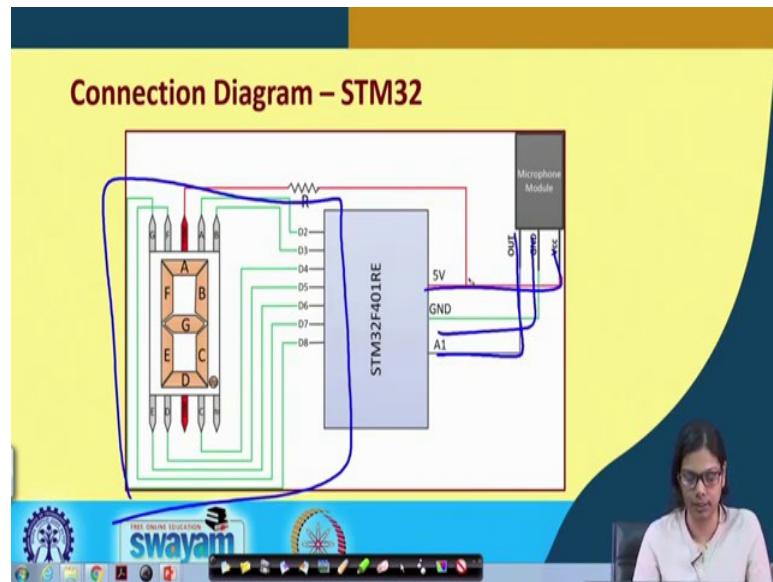
Experiment 2

- Interface a microphone and a 7-segment LED display unit to the microcontroller, and display the number of claps on the 7-segment display.



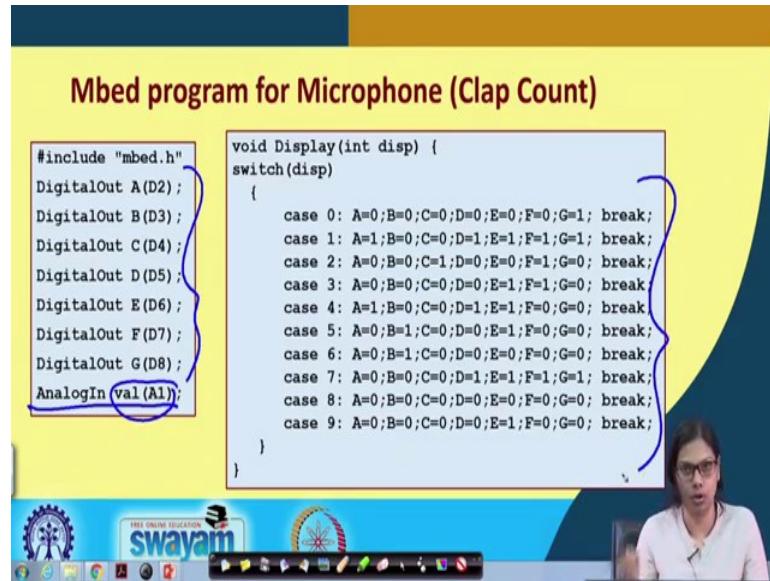
In the next experiment is we interface this microphone and a 7 segment display, and display the number of claps on a 7 segment display. Whenever I make a clap the 7 segment display value will get incremented, and when it reaches 9 it will turn back to 0 again.

(Refer Slide Time: 07:41)



This is the circuit diagram, which is straightforward.

(Refer Slide Time: 08:04)

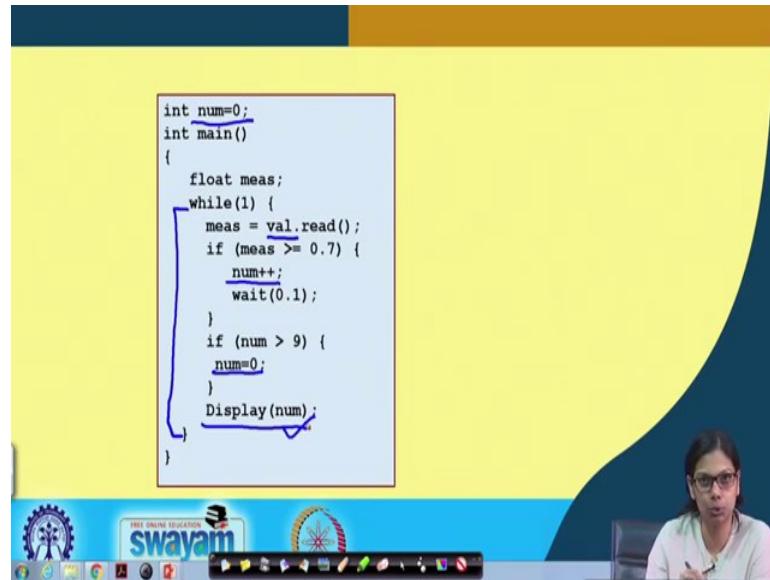


```
#include "mbed.h"
DigitalOut A(D2);
DigitalOut B(D3);
DigitalOut C(D4);
DigitalOut D(D5);
DigitalOut E(D6);
DigitalOut F(D7);
DigitalOut G(D8);
AnalogIn val(A1);

void Display(int disp) {
    switch(disp) {
        case 0: A=0;B=0;C=0;D=0;E=0;F=0;G=1; break;
        case 1: A=1;B=0;C=0;D=1;E=1;F=1;G=1; break;
        case 2: A=0;B=0;C=1;D=0;E=0;F=1;G=0; break;
        case 3: A=0;B=0;C=0;D=0;E=1;F=1;G=0; break;
        case 4: A=1;B=0;C=0;D=1;E=1;F=0;G=0; break;
        case 5: A=0;B=1;C=0;D=0;E=1;F=0;G=0; break;
        case 6: A=0;B=1;C=0;D=0;E=0;F=0;G=0; break;
        case 7: A=0;B=0;C=0;D=1;E=1;F=1;G=1; break;
        case 8: A=0;B=0;C=0;D=0;E=0;F=0;G=0; break;
        case 9: A=0;B=0;C=0;D=0;E=1;F=0;G=0; break;
    }
}
```

Now, this is the code that is straightforward.

(Refer Slide Time: 08:30)



```
int num=0;
int main()
{
    float meas;
    while(1) {
        meas = val.read();
        if (meas >= 0.7) {
            num++;
            wait(0.1);
        }
        if (num > 9) {
            num=0;
        }
        Display(num);
    }
}
```

Now, look into this code in the main. In this while(1) loop, we read through port A1, if the value is greater or equal to 0.7, num gets incremented. We wait for 0.1 second and then we again continue.

So, now we will be demonstrating the experiments using the microphone.

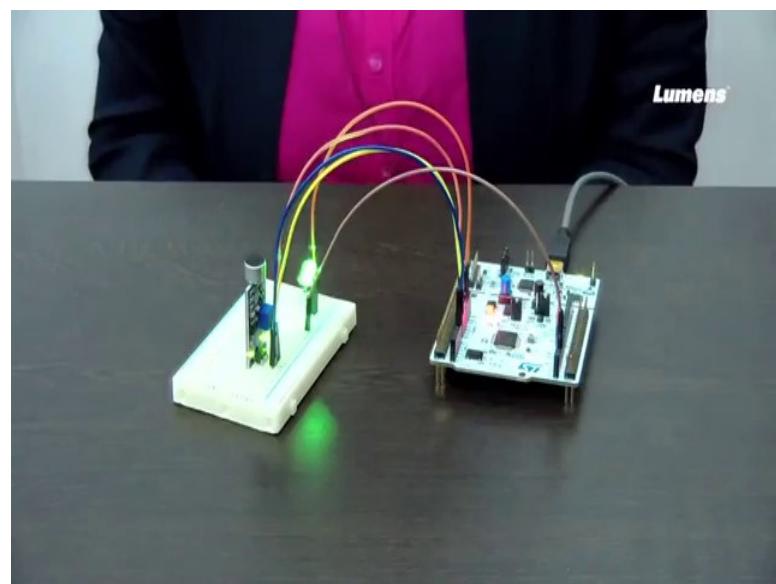
I will now show you the experiments using this microphone and with STM board.

(Refer Slide Time: 11:39)



This is the microphone. It has got three lines, Vcc, GND and OUT. We make the connections.

(Refer Slide Time: 12:52)



I have to connect this LED. The anode I am connecting to 3.3 volt, and cathode I am connecting it to port D2. The three pins in the microphone are Vcc, GND, and OUT. OUT is connected to analog port A2.

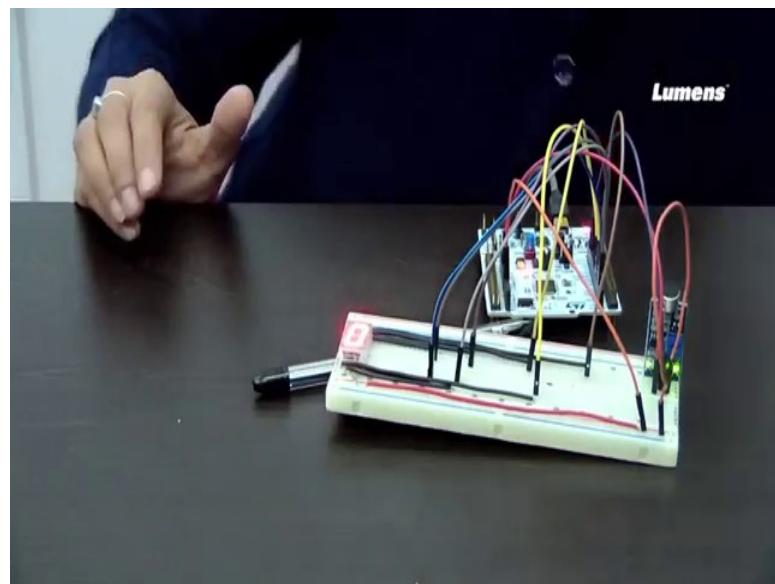
I will be dumping the code into this board now.

I will clap. You see when I clapp the LED is glowing for 3 second. I will do it once more. The experiment is very straightforward we just have to make sure that the connections are all right and then depending on the sound level the LED will glow. So, where is the application of this small thing? You can think of a notice board where you do not have to put light during the night time. When somebody comes near to the notice board you can actually make a clap, and with that a light inside the notice board will glow.

In continuation to the previous experiment using microphone, I will be showing you one more experiment where I have already made all the connection for 7 segment display.

Let us see here.

(Refer Slide Time: 19:11)



Whenever I will clap the number of claps will be displayed in this 7 segment display and when the clap reaches 9 again it will be reset to 0.

Thank you.

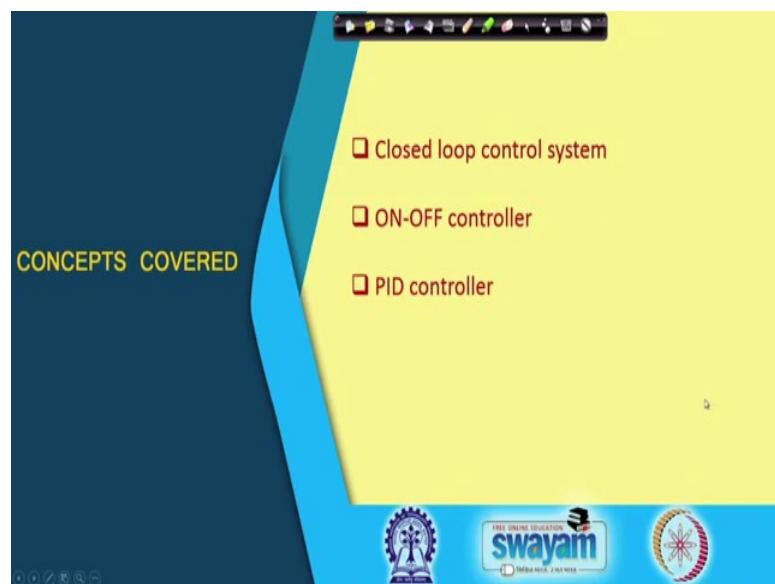
Embedded System Design with ARM
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 30
Design of Control System

You have seen through the various demonstration experiments that we have discussed so far. Like for example, you can turn on or turn off a light, you can glow an LED, you can turn it off depending on so many things. There are so many kinds of sensors, you can read some parameters from the outside world and using actuators and output devices, you can output some data to the outside world.

Now in this lecture, we shall be talking about something called Control Systems very briefly. We shall look into the basic idea about control systems and why it is important in the context of embedded system design.

(Refer Slide Time: 01:15)



In this lecture, we shall be talking particularly about something called closed-loop control systems, and the different kinds of controlling mechanism like ON-OFF control and PID control.

Let me tell you here suppose I am interfacing a heater with a microcontroller. I am turning it off if I feel hot, I turn it on if I feel cold again, I turn it on again. But, suppose I

want to automate this process, I want to design my system in such a way that whenever the temperature falls below a certain level, let us say 25 degree centigrade, I should turn on the heater, if it is above I should turn off.

But, for that mechanism there has to be a feedback in place. Like not only I should be able to control my heater, I should also be able to read the value of the current temperature. I should be able to know automatically whether my temperature is lower or higher, that is the notion of a closed loop control system that comes into the picture.

(Refer Slide Time: 02:41)

Introduction

- In many embedded system applications, we have to sense the value of some external parameter and take corrective actions to maintain it within acceptable limits.
 - Temperature of an oven, speed of a motor, etc.
 - Essentially a control system.
- Two types of control systems:
 - a) Open loop: where there is no feedback with respect to the measured value.
 - b) Closed loop: more sophisticated, corrective actions applied with feedback.

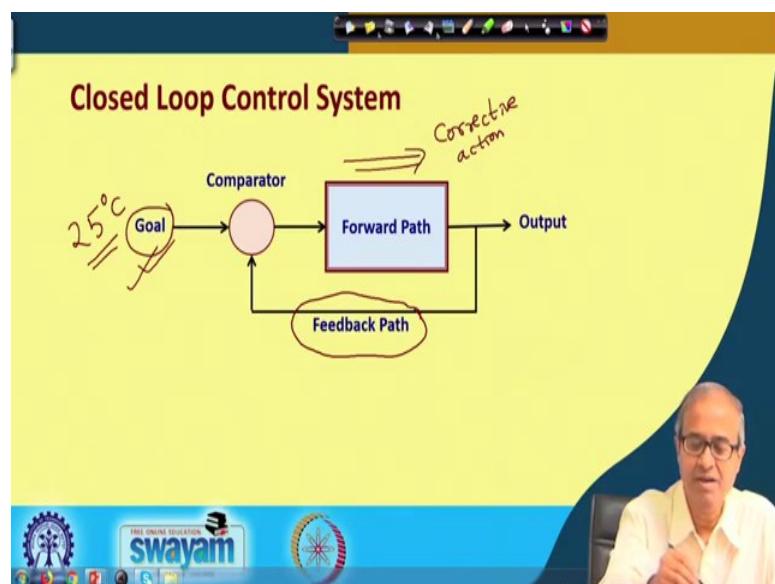
25°-27°C

What we are saying is that there are many embedded system applications like the one I just now talked about. In the example, I took the parameters as temperature, humidity, it can be anything. Now depending on the value you have sensed, you can take a corrective action. Why? To maintain it within acceptable limits.

Like let us say, for temperature you may say that my acceptable limit is 25 to 27 degree Celsius. If you find that the temperature is falling below it, you should turn on the heater. If you find the temperature is going above it, if there is an air conditioning mechanism, you should be turning it on. This is the notion of sensing and feedback I am talking about. There can be many applications you can think of; temperature of the room or an oven, speed of a motor, etc.

These are examples of control systems; broadly speaking control systems can be either open loop or closed loop. Open loop means there is no feedback mechanism; like you are only turning on or turning off the heater, but you are not reading the value of the temperature. But closed loop means that not only you are controlling, there is also a feedback mechanism. Naturally, this is more sophisticated because, based on the feedback you can send some corrective signal. Let us say here, to maintain the temperature within this range, you have to send the heater control accordingly. This is the basic idea.

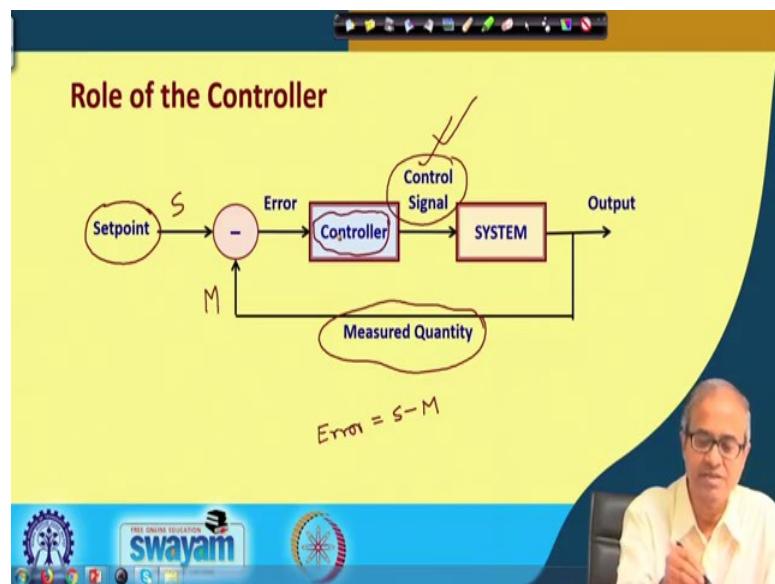
(Refer Slide Time: 05:09)



Let us look at a diagram of a typical closed loop control system. What does this diagram contain? Here we start with a goal, goal means we want to maintain the parameter at some particular level; for temperature let us say, it is 25 degrees Celsius.

And from the output let us say the room where I am sitting, there will be a feedback path; that means, there will be some sensors, which will be reading the temperature and it will be sending back some feedback value. There will be a comparator, which will be checking whether this feedback, whatever you are reading is less than or greater than the set goal. Depending on that you will have to send some kind of a corrective action along the forward path so that the output value of the parameter is as close as possible to the set goal. This is the overall purpose of having a closed loop control system.

(Refer Slide Time: 06:49)

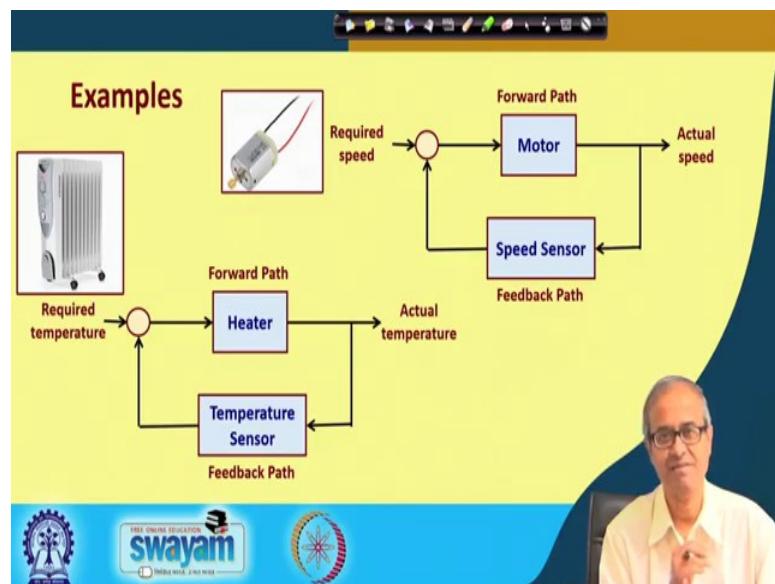


Let us expand the diagram a little bit more. Let us talk about the goal here, we are calling it setpoint ... same thing we are setting a level, where you want to maintain the value of a physical parameter to and the output through some sensor we are measuring it.

Let us say my setpoint is S , my measured value is M , let us error to be $S - M$. If they are equal it is fine, as my output is just at the set point. But depending on the error whether it is less than or greater than, there will be a controller which will be sitting here, controller will take a decision that how to generate a control signal for my system.

So, my system here is my heater or AC machine or whatever I am trying to control. There is a microcontroller will be acting as the controller mechanism that will take an intelligent decision, how to generate the control signal so that my output is as close as possible to my set point. This is where the role of the controller comes in. There can be various alternate designs of controllers we can think of.

(Refer Slide Time: 08:35)



First let us look at some examples. This is a very simple example of a room heater. Suppose I have a room heater where I have set a required temperature. Here I am assuming that I am in a place which is quite cold, it means if you are residing in a place which is hot you cannot appreciate this example, suppose I am staying in a place which is very cold.

So, I need a room heater. There will be a temperature sensor in the feedback path. The actual temperature will be sensed, the comparator will be finding a difference, and in the forward path the controller will be controlling a heater. Take another example. Suppose I want to control the speed of a DC motor. Some required speed is specified, let us say 100 revolutions per minute or 100 RPM.

There will be a speed sensor in the feedback path, it will be sensing the speed in some way. Depending on the difference the controller will be activating the power supply of the motor to turn it on and off, so that speed is maintained.

(Refer Slide Time: 10:03)

Choice of Controller Type

a) ON-OFF Controller:

- An ON-OFF controller is the simplest type of controller, where the control signal has only two levels.
 - For example, in a heater, if the actual temperature is less than the set temperature, the heater is turned ON; otherwise, it is turned OFF.
- This type of controller is inexpensive, but often causes oscillation in the output variable.
- It is often used in simple appliances such as oven, iron, refrigerators, etc. where oscillations can be tolerated.

Output
ON
OFF

Setting

Temperature

Heater OFF

Heater ON

Heater OFF

Heater ON

Time

Controller → 0 or 1

FREE ONLINE EDUCATION
swayam

Talking of the role of controller, there can be various different types of control mechanisms. Let us start with this simplest kind of controller called ON-OFF controller. ON-OFF controller means we either turn on or turn off the device depending on whether the measured value is less than or greater than the preset.

As I said this is the simplest type of controller, where the control signal has only 2 levels, if you say that here I have the controller. So, controller will be generating a single output, which is 0 or 1, 0 means off and 1 means on. But it can cause oscillation in the output variable. How? You see this diagram on the right. Here, the graph that I am showing is a temperature versus time graph. Suppose in a room where I am sitting, I am measuring the temperature and I have a set temperature, let us say 27 degree Celsius.

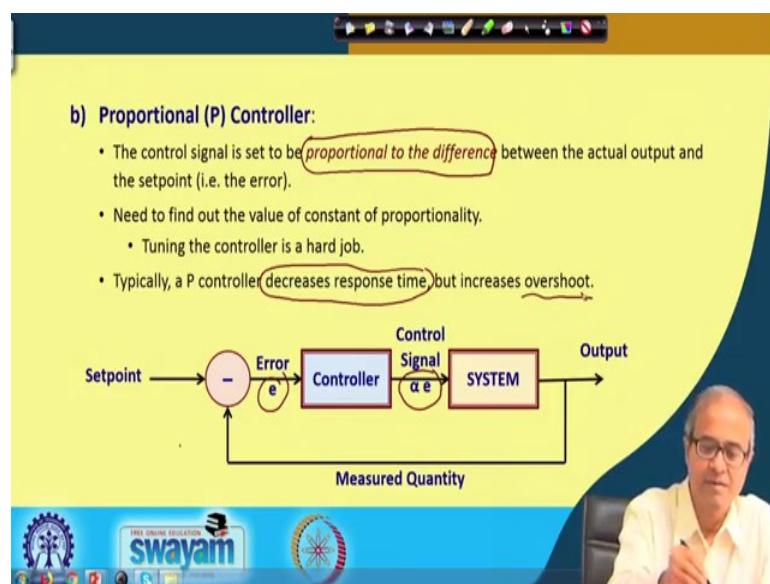
When the temperature is less than 27, this is my controller output; the heater is on. So, the temperature will slowly go up, now as soon as it crosses this set level, I turn off the heater. Now you see as soon as we turn off the heater, the temperature suddenly does not start to fall because, heater is already glowing. So, temperature will still increase for some time then it will stop increasing any further.

Thus, there will be an overshoot like this here, here and then when you turn off the heater the temperature will again slowly start to go down. Later, you again turn on the heater, again heater will take some time for the coil to become hot. So, this will again go down and again this process will repeat. So, you will see there will be a continuous oscillation

like this up down, up down, up down this will go on, it is not that it will be very stable at a certain level, it will be going 26, 28, 26, 28 something like this will happen.

This kind of a controller is quite good for applications like room heating, where small degree difference does not matter. But, there are some applications where you need to control the parameter very closely, there you cannot use this kind of a simple controller.

(Refer Slide Time: 13:19)



You can have something that is slightly more sophisticated. We can use something called a proportional controller or P controller.

What does this do? Here my controller is not generating a digital signal, but is generating an analog signal proportional to my error. If I see my room temperature and my set temperature is quite different, I sent a high voltage to the heater. If I see my difference is very small I sent a lower voltage to the heater so that the heater does not suddenly become very hot and there is an overshoot, but if the difference is very large I sent a large voltage. So, the heater heats up quickly so that it very quickly attains the set value.

Looking into that same diagram again, if your error is e your control signal will be proportional to e . So, if $e = 0$, then you are not sending any corrective signal. Whatever level was there you leave it, this is how it works.

Now what is the advantage you are getting out of this proportional controller as compared to the on-off controller? If the error is large you send a larger corrective signal

so that the error can be reduced very quickly. So response time is decreased, but overshoot still remains as it can go up and it can again go down, this kind of overshooting will still be there. Because, see you are still sending a control signal that will change the value of the parameter in one way and you cannot instantaneously shut it off, it will take some time for it and there will be some overshoot.

(Refer Slide Time: 15:53)

c) **Proportional-Derivative (PD) Controller:**

- To reduce the overshoot, we can take into account how fast we are approaching the setpoint.
 - We add D control in addition to P control.
- D is estimated as the difference between the current measure and the previous measure.
- PD controllers are slower than P controllers, but generates less oscillation, and smaller overshoot/ripple.
- Drawback:**
 - Output is close to the setpoint, and so the error is very small.
 - Errors add up over time; we can define the integral (I) of the error:
$$\Sigma_{\text{time}} (\text{setpoint} - \text{output})$$

So, what you do in the next step is that you add another flavor to it, add a derivative concept proportional and derivative controller, because in proportional controller I mentioned that there was the problem of overshoot.

In the derivative part you are also taking care of the fact that how fast you are approaching the set point that also you are taking care of. Derivative means the difference; last value of the measured variable minus the present value of the measured variable. So, derivative is nothing but a measure of the difference between the current measure and the previous measure, you continuously sense the output variable, you saw that last time the temperature was let us say 21 now it is 23. So, it has increased by 2 degrees; that means, increasing very fast. So, you should decrease the rate such that overshoot will not be there.

Here the control signal that you are generating will be proportional to the error plus it will also be the proportional to the difference; that means, you are measured value previous minus measured value present, this is your derivative. So, this will be

proportional to both of these of course, the proportionality constant may be different for the two cases because, you may want to tune such that you will be giving more importance to proportional or more importance to derivative.

PD controller works better in terms of reducing overshoot because as you are approaching the goal faster, you reduce the control signal so that overshoot will be less.

(Refer Slide Time: 18:37)

c) Proportional-Derivative (PD) Controller:

- To reduce the overshoot, we can take into account how fast we are approaching the setpoint.
 - We add D control in addition to P control.
 - D is estimated as the *difference* between the current measure and the previous measure.
 - PD controllers are slower than P controllers, but generates less oscillation, and smaller overshoot/ripple.
- **Drawback:**
 - Output is close to the setpoint, and so the error is very small.
 - Errors add up over time; we can define the integral (I) of the error:
$$\sum_{\text{time}} (\text{setpoint} - \text{output})$$

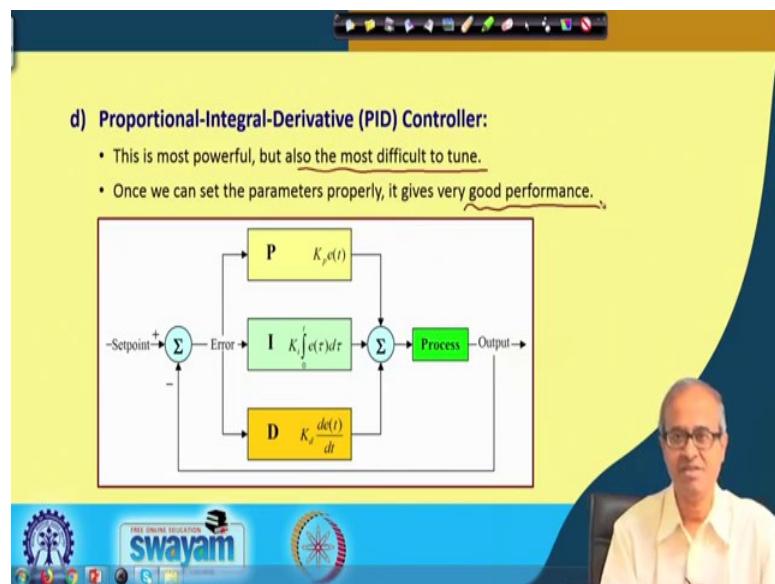
$e_1 + e_2 + e_3 + \dots$

I

But one thing is there; these PD controllers are slower than P controller, because you are deliberately slowing the rate of convergence. If it is closer you make it slower that is why it is lower, but oscillation will be less and also overshoot is less. But one drawback is that that in the PD control the output becomes close to the set point; that means, the error is small, but errors tend to accumulate over a period of time. If we allow for a small amount of error, this errors will go on adding.

So, you also need to consider a third parameter that is the integral. For integral the mathematical concept is summation over a certain period of time, if you just add up the errors this will define the value of the integral. So, you can add a third point, where summation over time this error, this will give you how much error you are accumulating over time. You add this third term in your control expression, one will be proportional to the error that is your P, one will be proportional to the derivative that is D, and here one will be the proportional to the cumulative error that is I; the 3 taken together is called PID controller.

(Refer Slide Time: 20:21)



This is what PID controller is. This is obviously most powerful, but now because we have 3 different things to tune, what will be the 3 constants of proportionality for a particular application is a difficult question. You have the set point, you have the measuring mechanism, you have a proportional part in the controller, integral part, derivative part, you add all of these three up. There can have three different constants of proportionalities K_p , K_i and K_d , and you generate a consolidated control signal for the process you are trying to control. Now again, I told you that this is most difficult to tune, but once you have tuned it, this will give the best performance among all.

PID controller is good in this sense and in embedded system applications, because you are talking about controlling some appliances and all of these are feedback control systems. You may choose to select the appropriate mode of control for some appliances like the refrigerator, for which on-off control is good enough. But you can have PD control or PID control in more sophisticated applications, where errors in the output cannot be tolerated beyond the certain limit.

(Refer Slide Time: 22:05)

Controller	Response Time	Overshoot	Error
ON-OFF	Smallest	Highest	Large
Proportional	Small	Large	Small
Integral	Decreases	Increases	Zero
Derivative	Increases	Decreases	Small change

The table summarizes the differences.

But all these things are really not required, you need to understand better about the parameter you are trying to control in an application and then you can select an appropriate method of control. In the experiments that we shall be showing for simplicity, we shall be using on-off kind of control, because it is easier to show and also easier to write the program. Because, you see whether you use ON-OFF or PID control depends entirely on your software, and nothing to do with the hardware.

With this we come to the end of this lecture, where we have discussed some basic concepts about control systems in particular the feedback control systems and some standard ways of generating the control signal to minimize errors and other desirable properties.

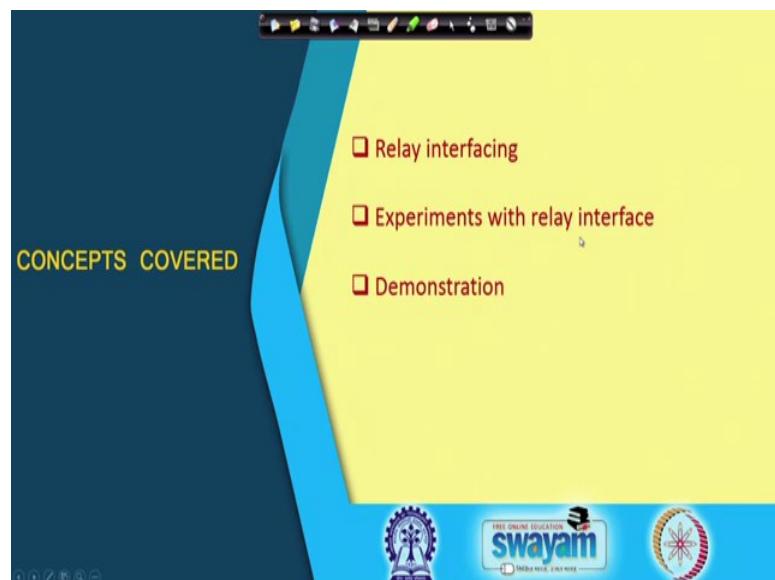
Thank you.

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

Lecture – 31
Experiments with Relay

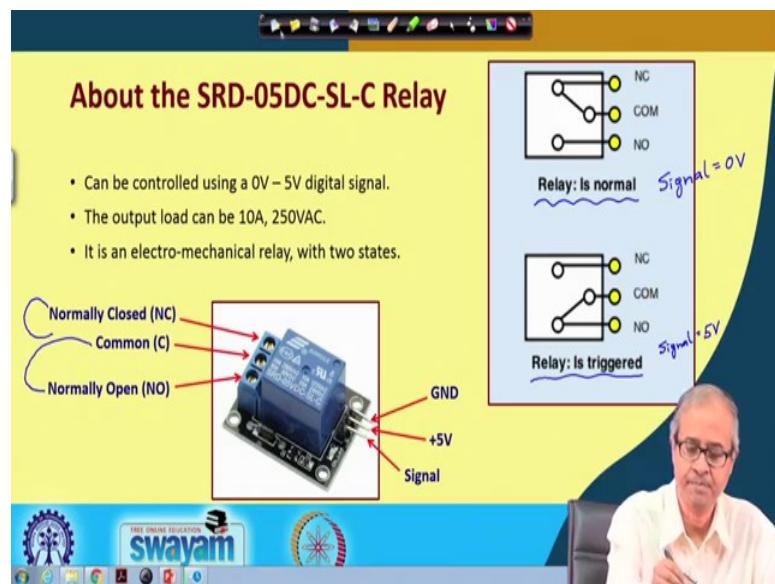
In this lecture, we shall be showing you some hands on demonstration experiments using relays. You have already seen earlier what a relay is. Relay is a device that can switch a higher power circuit through the control of a microcontroller using a much lower voltage.

(Refer Slide Time: 00:47)



We shall be talking about relay interfacing with microcontroller boards and we shall be showing you some experiments and demonstration.

(Refer Slide Time: 00:59)



The type of relay that we shall be using in our demonstration is SRD-05DC-SL-C. As you can see this is a picture of the relay board that we shall be using. You may note that although this particular relay I shall be showing in this experiment has a single relay device, there are boards available where there are 2 or 4 such relay devices integrated in a single board.

So, if you have any application where you want to control multiple electrical appliances from the same microcontroller, you can use one of those boards, but here we shall be controlling a single appliance that is why I am using a single relay module. Now talking about this relay, you see on one side, you connect this signal to the microcontroller. You see there is a GND there is a +5 volt, which provides the power to this device and there is a third pin, a digital signal using which you can switch on or OFF the relay.

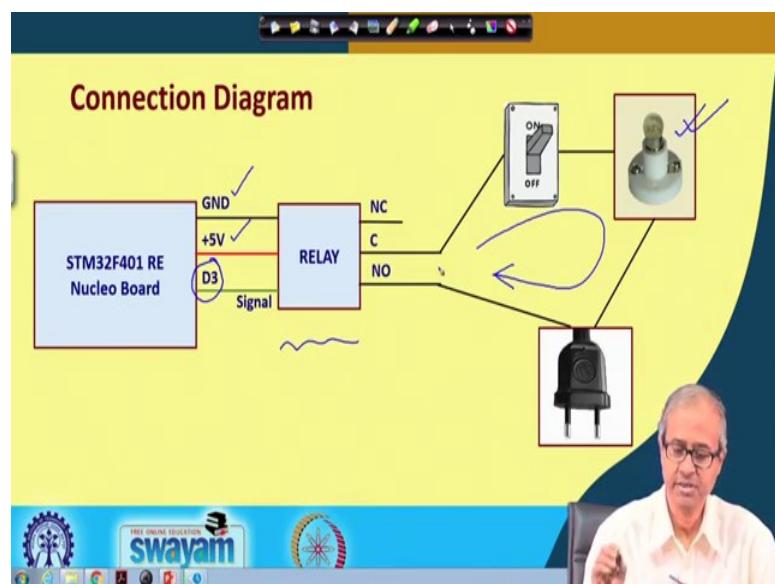
This signal can work in 0 to 5 volt range, 0 means relay will be OFF, if you apply 5 volts the relay will be on. And on the output side, you are supposed to connect a higher power device, this can be 10 ampere and up to 250 volt AC power supply, you can switch ON the output side. The point to notice is that this device is not a solid state relay, rather it is an electromechanical relay. So, when you apply a current there is an electromagnet inside, which gets magnetized. There is a metal switch, which gets attracted and if there is no current using spring loading it turns off.

When we switch a relay ON and OFF, you can also hear a sound tuck tuck tuck tuck like this. The switch will be connecting and disconnecting like that. On the output side you see there are 3 connectors, normally closed (NC), a common (COM), which is connected to ground and normally open (NO). So, when you connect any device you either connect them between the COM and NO, or between COM and NC.

The difference is that if we use NO then when the relay is not on; that means, the signal is 0, this circuit will also be open and there will be no current flowing, but when we apply a signal of 5 volts, this circuit will be turning on. But for the NC pin it is just the reverse. When you are applying a 0 volt on the signal; that means, relay is OFF, but on the output side the circuit will be on. But, when you turn on the relay the circuit will be off; that means, just the reverse convention. Pictorially it is shown in the diagram.

This is the normal state of the relay, where this signal this input signal that you are applying where signal is at 0 volts. You see for the NC connection it is internally connected in the normal state, but for NO it is open. But, when the relay is triggered that means this signal input is at 5 volts then you see the reverse happens. The NC connection becomes floating and the NO connection gets connected to COM. So, we will be connecting either between NO and COM, or between NC and COM.

(Refer Slide Time: 05:27)



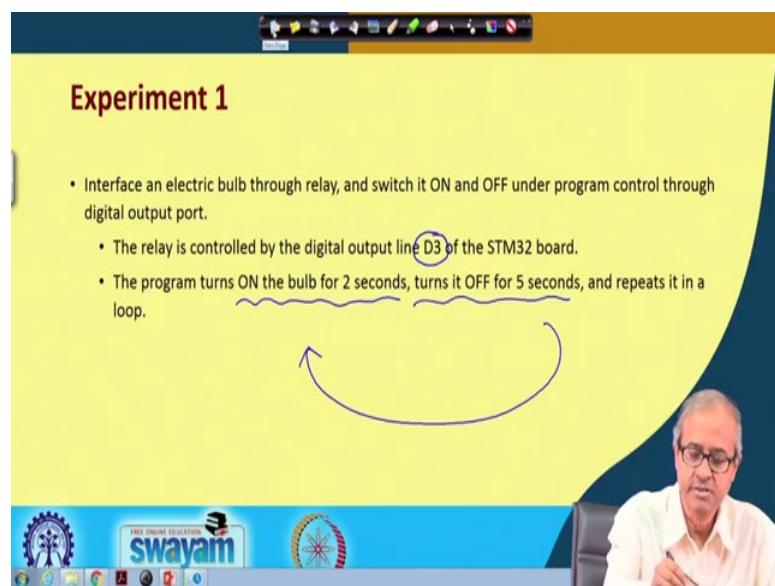
This is about the connection diagram how we will be connecting the relay to the microcontroller; we will be using STM32 board for the demonstration. This is the block

diagram of the relay I am showing in the middle. When you connect it to the STM board you require the ground connection, you require +5V and you require a signal.

In this experiment, I will be using the data line D3 for feeding the signal and on the output side for the sake of demonstration, we will be using a small LED bulb, which will be turning ON and OFF under relay control. We will be using NO, means when the relay is not switched ON the bulb will not glow.

This is the circuit through which the current will be flowing. You see there is an electric power cord, which will be connected to the mains, there is the bulb and there is an external switch. Of course we will not be requiring the switch in this experiment, this switch will be always on. Whenever the relay is turned ON, there will be a current flowing and the bulb will glow; and when the relay is OFF, there will be no current and the bulb will be turned off. So, this is the connection diagram that we will be using.

(Refer Slide Time: 06:57)

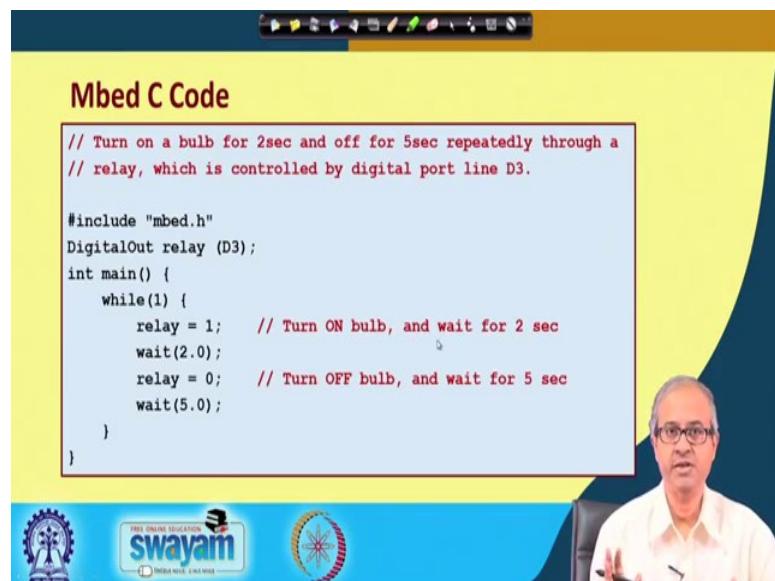


Now, let us come to the first experiment. We shall be just turning ON and OFF the relay with some time delay. If you see the statement of this experiment, we will be interfacing the electric bulb, we have already seen how we shall be interfacing the circuit, and will be switching it on and off.

We have already seen in the circuit diagram that the relay is connected to the output line D3 of the STM board, and the program is written in such a way that the relay will be

turned on for 2 seconds and turned off for 5 seconds, and this process will repeat indefinitely. Before showing you the demonstration, let me first show you the code, what mbed C code we have written to achieve this.

(Refer Slide Time: 07:59)



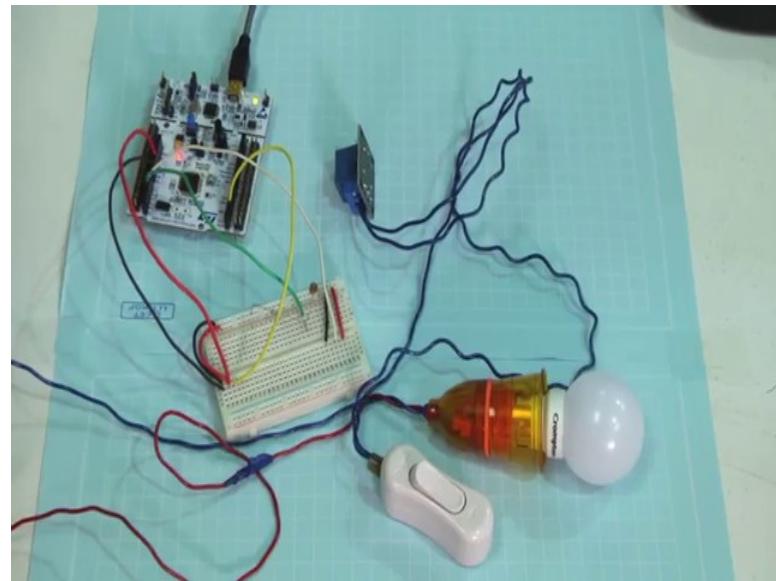
```
// Turn on a bulb for 2sec and off for 5sec repeatedly through a
// relay, which is controlled by digital port line D3.

#include "mbed.h"
DigitalOut relay (D3);
int main() {
    while(1) {
        relay = 1;      // Turn ON bulb, and wait for 2 sec
        wait(2.0);
        relay = 0;      // Turn OFF bulb, and wait for 5 sec
        wait(5.0);
    }
}
```

This is the code that is fairly simple and straightforward. If you look into the code we have included this mbed.h header, and D3 we have declared as a digital out pin. Because, here we are using a simple digital output mode of that output pin and we are calling this pin as “relay”, you can give any name here. This is our main function that runs in a continuous while loop.

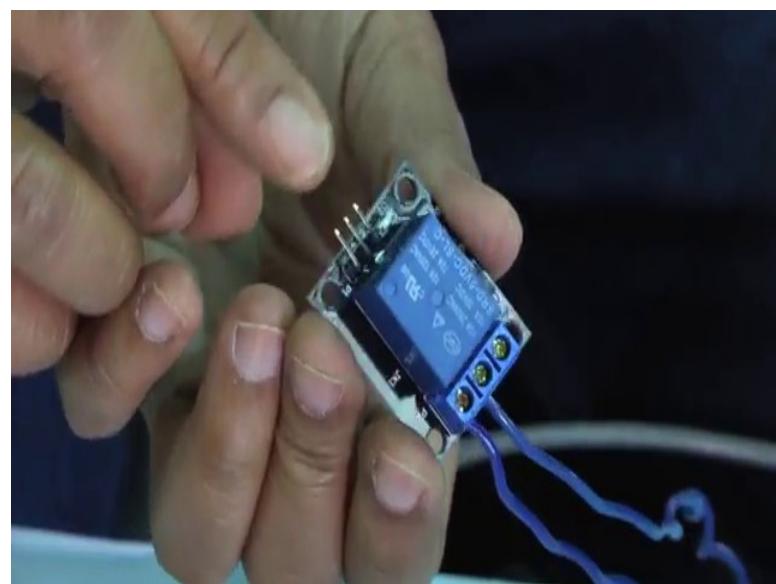
When it outputs 1 to relay, the circuit will be switched ON and the bulb will glow for 2 seconds. relay = 0 means again, the circuit will be switched OFF and will wait for 5 seconds. So, you turn ON the bulb, wait for 2 seconds, turn OFF the bulb, wait for 5 seconds in a repeated while loop. Let us now see the demonstration.

(Refer Slide Time: 09:23)



Here you look at this circuit that I am showing. As you can see this is your STM microcontroller board and the circuit for the bulb that I have shown. There is an electric power inlet, which is connected to an electric power source AC 220 volts, there is a switch which I am permanently putting as on, and this is a small LED bulb I am using, and this is the relay module that we are using.

(Refer Slide Time: 09:53)



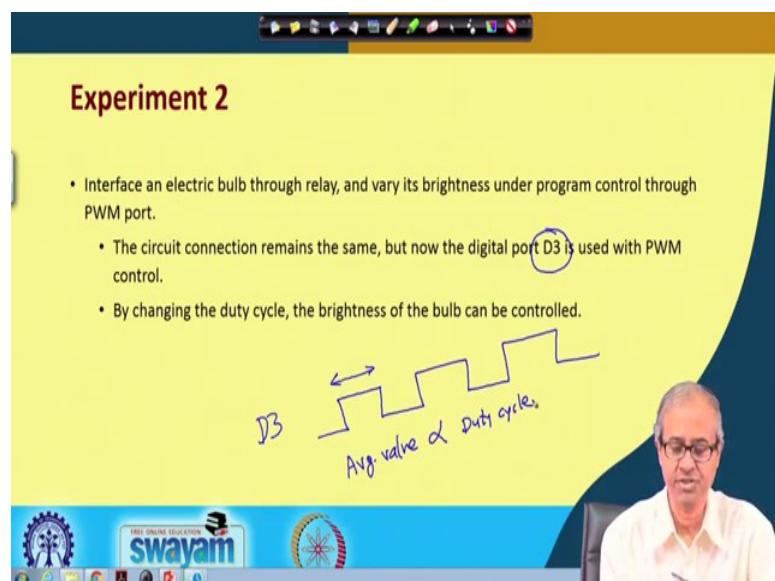
You see on one side you have connected this relay module to the NO and the COM connections, and on the other side there are 3 pins you have seen. These pins are signal,

Vcc and GND; these 3 pins have to be connected. Now on this breadboard, I have already made such connections. On the right side I will simply plug in this relay.

So I have plugged in this relay into this breadboard, let me make a solid connection ... yes there is a loose connection let me just make it right. Now, let me compile this program, you save it by default. It will get saved into the download folder and you copy and paste it to the F401 drive, the program has been downloaded.

Now, I switch on the power. You see what happens; the bulb will switch ON for 2 seconds and will get switched OFF for 5 seconds, it is switching ON 2 seconds switch OFF for 5 seconds again, it switches ON for 2 seconds again switches OFF for 5 seconds. So, interfacing of the relay is fairly simple in this case as you can see. Let us again come back to our presentation and the next experiment.

(Refer Slide Time: 12:35)



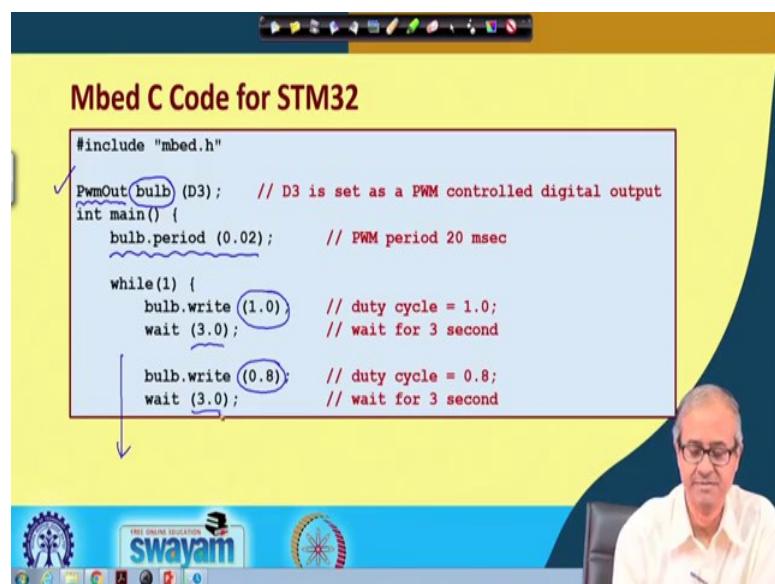
You must have noticed that when the bulb is switching ON and OFF, there is an audible sound indicating that the relay switch is turning on and off. In the next experiment, you can hear it much more clearly because, will be switching ON and OFF much faster.

Coming to the second experiment, the interface is very similar, we are again connecting the bulb through the relay using the same circuit, we are still connecting the relay to the port D3, but in the previous experiment we were using digital control, either 0 or 1, but in this experiments were using PWM control to send a continuous pulse train on D3.

So, the relay will be turned ON for certain time and turned OFF for certain time, and this will happen repeatedly. And by controlling the duty cycle, the brightness of the bulb will change, because as you know for a PWM the average value of the voltage output will be proportional to the duty cycle.

So, as we change the duty cycle the average value of voltage that gets applied to the relay and hence the bulb will vary in proportional to the duty cycle.

(Refer Slide Time: 14:33)



```
#include "mbed.h"

PwmOut bulb(D3);      // D3 is set as a PWM controlled digital output
int main() {
    bulb.period(0.02);      // PWM period 20 msec

    while(1) {
        bulb.write(1.0);    // duty cycle = 1.0;
        wait(3.0);          // wait for 3 second

        bulb.write(0.8);    // duty cycle = 0.8;
        wait(3.0);          // wait for 3 second
    }
}
```

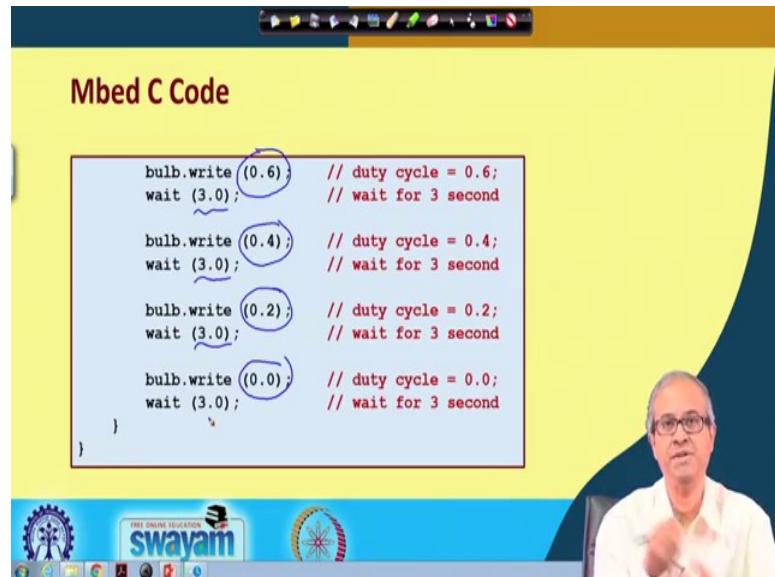
In this program, just for the sake of demonstration, we have included mbed.h, but now this D3 we have declared as PwmOut and we have given the name “bulb”. Now, in the main function we have done a few things. In the beginning, for this PwmOut, you already know that there are some functions we can use to set the period, to set the duty cycle, and so on.

First we set the time period in seconds as 0.02, which means 20 milliseconds. This means 50 hertz. So, in every second the relay will be switching ON and OFF 50 times. I have kept it as 50; the point to note is that because you using a mechanical relay, you cannot have this frequency too much faster, as the relay will not get time to switch ON and OFF.

The program continues in a while loop, we are doing certain things one by one in a repetitive fashion. bulb.write() is a function, where you can set the duty cycle. First we

are setting duty cycle to 1, which means it is continuously 1, it is never going 0, this is the maximum brightness, and you wait for 3 seconds. Then we make the duty cycle as 80%. So, a little less, then again wait for 3 seconds and this we repeat.

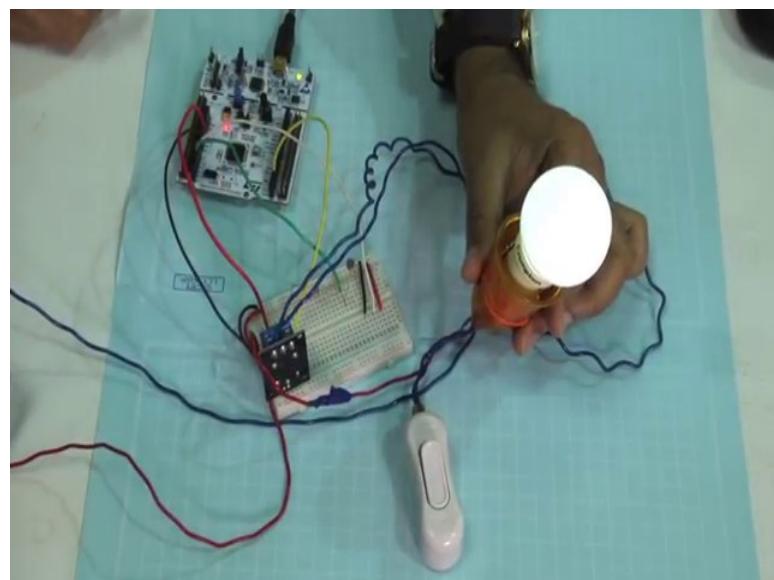
(Refer Slide Time: 16:37)



Then we make the duty cycle as 0.6 again wait for 3 seconds, then 0.4 wait for 3 seconds, 0.2 wait for 3 seconds, and finally we turn OFF completely.

So, in a loop we will see that the brightness of the bulb will progressively change in a continuous fashion. Let us see the demonstration.

(Refer Slide Time: 17:17)

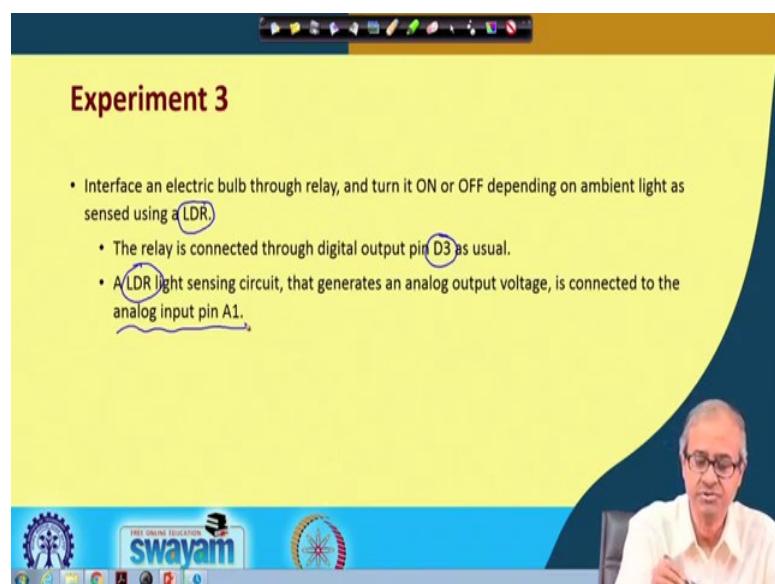


This is our second experiment. Let us again compile this program, the program that I have shown that same program is here. We save it, we copy this, we paste it and the new program is getting downloaded.

Now see what happens. In every second the relay is switching on and off 50 times. You can hear the sound very clearly, and in the bulb you can see that with gaps of 3 seconds the brightness is changed. This is the maximum brightness, then this is 0.8, this is 0.6, this is 0.4, this is 0.2 very light and then totally OFF; and this cycle repeats.

This simple experiment actually shows you how we can switch ON and OFF a bulb or any circuit for certain duty cycle, and period in a PWM controlled fashion. So, let us continue with our discussion.

(Refer Slide Time: 18:57)



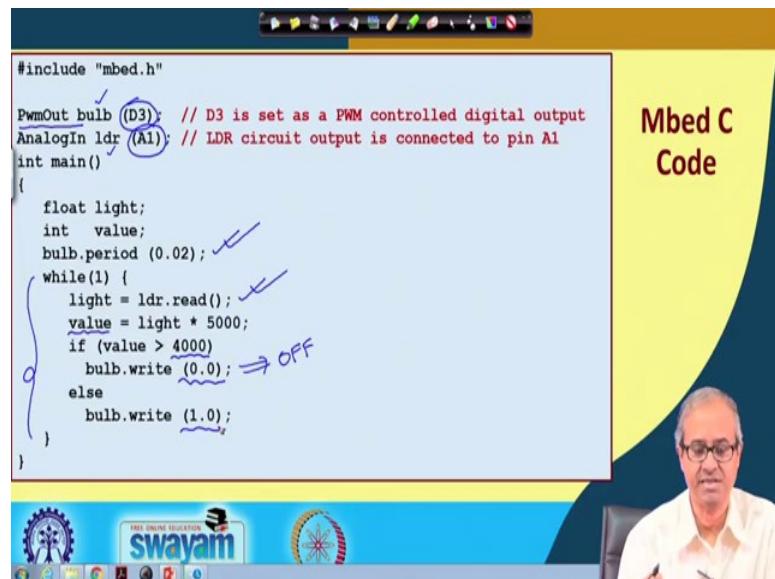
Now, for the last code that we shall be showing you, this is the same relay circuit, but we have made one more addition here, we have connected a LDR.

Our circuit is like this; the electric bulb through relay that connection is already, the relay is still connected to the pin D3, and there is an LDR light sensing unit, which we are connecting to analog input pin A1.

The relay circuit is the same, and this is the additional circuit we are using. Here, with very is an LDR and another resistance connected as a potential divider.

Now this output voltage we have connected to the analog input pin A1 of the STM32 board. Now let us see the code, what we are doing.

(Refer Slide Time: 20:37)



Mbed C Code

```
#include "mbed.h"

PwmOut bulb(D3); // D3 is set as a PWM controlled digital output
AnalogIn ldr(A1); // LDR circuit output is connected to pin A1
int main()
{
    float light;
    int value;
    bulb.period(0.02);
    while(1) {
        light = ldr.read();
        value = light * 5000;
        if (value > 4000)
            bulb.write(0.0); // OFF
        else
            bulb.write(1.0);
    }
}
```

Now, in the main program we are setting the period to 20 milliseconds, just like the previous experiment. Now in the while loop, we are reading the light value from the LDR; the value that is read is a fraction between 0 and 1. So, to scale it up to an integer value, which you can actually compare, we multiply it by 5000, and store in a variable called “value”.

Now, through experimentation, we found 4000 to be a suitable threshold value. So, greater than 4000 means we have sufficient light in the ambience, and we have to switch OFF the light. You think of a home, when there is sufficient light there is no point in switching ON the light. So, we set the duty cycle to 0.0, which means the light is switched OFF.

But when it falls less than 4000, it means that there is darkness. Now, we have to switch ON the light. So, we set the duty cycle to 1.0, which means maximum brightness and this process repeats. Let us show you the demonstration for this. We show that same code that we have shown on the slide. Let me compile this code first, save it then copy it, paste it on the nucleo board.

Now, let us see what is happening. Here in addition to the relay circuit, now you can see we also have a LDR connected out here. You can see this LDR, and there is a resistance. So, LDR and resistance are connected together in one junction one end of the LDR is connected to 5 volts, and the other end of the resistance is connected to ground, and from the middle point of the LDR, this green wire is getting connected to the analog input pin A1.

You see now there is sufficient light. So, the bulb is not switched on. Now if I press the LDR, you see the bulb is getting switched on; that means, there is darkness. I again remove my hand the bulb is switched OFF again, I press the LDR; that means, darkness the light is switched ON.

This is a very simple experiment that shows you how an LDR can be used under program control to switch ON and switch OFF any electrical appliance. This experiment you can very easily correlate with some kind of home automation system, where you want to switch OFF some electrical gadget like bulb, bulb is a very practical example depending on the value of the ambience light. Here, I have given an example of light; there can be other things like temperature, if the temperature becomes too high, you can switch ON the air conditioning machine. If the temperature is very low, you can turn ON a heater.

In this set of experiments I showed you how we can use a relay controlled by a microcontroller to switch ON or OFF some electrical appliance. This concept you can use for any kind of device, not necessary a bulb. You can switch ON and OFF an AC machine or a heater as I told, you can switch ON and OFF a refrigerator, you can switch ON a switch of any gadget not only one device as I have shown here you can have multiple such devices.

We would look at more experiments later on, where you will see some kind of home automation system, where some more sophisticated kind of control and communication mechanism will be shown. With this we come to the end of this lecture.

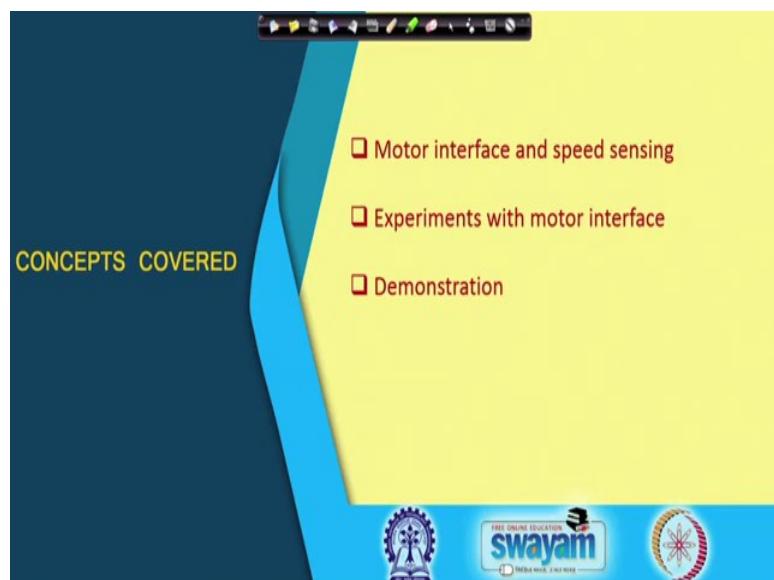
Thank you.

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

Lecture – 32
Experiments on Speed Control of DC Motor

In this lecture, we shall be showing you some more interfacing experiments; namely we shall be showing how to interface a DC motor with the STM32 microcontroller board. We have used a small DC motor as an example to show you how this kind of interfacing can be done. We shall be showing you a couple of experiments on the controlling of the motor and how the speed can be varied.

(Refer Slide Time: 00:51)



We shall be discussing how motor can be interfaced and how speed can be sensed.

(Refer Slide Time: 01:03)

About the Motor Interface and Speed Sensing

- A DC motor is used, where the rotating shaft is connected to a metal wheel cut with 8 slots.
- The wheel is attached to an optocoupler circuit, which generates optical interruptions whenever the wheel rotates.
 - 8 optical interrupts for every single rotation of the wheel.
 - An optocoupler circuit generates a pulse for every interruption.
- How is the motor driven?
 - Directly from the PWM port of the microcontroller.
 - By changing the PWM duty cycle, the speed of the motor can be varied.

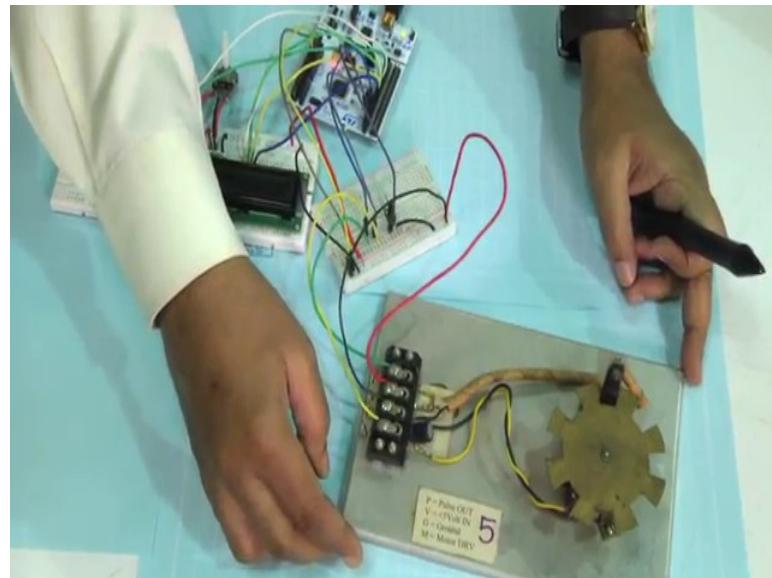
Slots

Optocoupler

swayam

Let us talk about the motor interface that I shall be showing you in this experiment.

(Refer Slide Time: 01:17)



Well, if you can see it once, this is the motor that I have interfaced. You can see there is a motor down below and there is a wheel that is connected to the shaft, which can rotate.

Coming back to this slide, in the picture on the right, I have showed the same shaft. What you see is that there are 8 slots, which are cut ... 1, 2, 3, 4, 5, 6, 7, 8. The rotating shaft is connected to the metal wheel where there are 8 slots. And, on this side you can see a black kind of a thing; there is an arrangement that is an opto-coupler circuit, which I

shall be telling you about. Earlier you had seen how an LDR circuit can be used to sense optical interruptions and count the number of objects, which are crossing a certain point, may be number of persons entering and leaving a room.

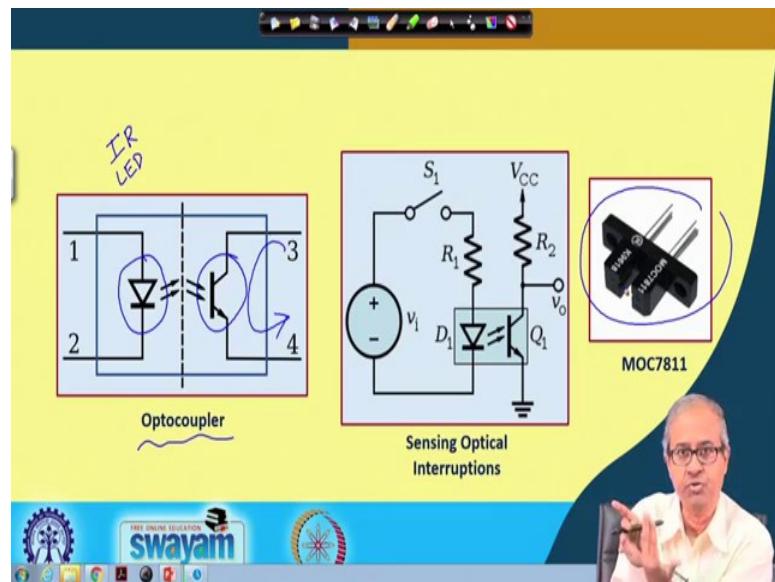
Here we could have used LDR and some kind of LED light source also, but we thought let us use some other kind of an optical sensing circuit, using which you can measure interruptions, and this is something which is called an opto-coupler.

We shall tell you that how an opto-coupler works, you see the opto-coupler is strategically placed just in the place where the wheel is rotating, because there are 8 slots in the wheel, and for one complete revolution there will be 8 interruptions.

So, there will be 8 such optical interruptions for every rotation of the wheel. And this circuit that we will be using will be generating one pulse for every interruption; for one revolution there will be 8 pulses. And the motor, because it is a small motor we are using, we can drive it directly from the microcontroller output port, we have used one PWM port. But, if it is a larger motor, of course you need to use a driver circuit, which can supply the required voltage and current to drive the larger motor, but this being a small motor we can drive it directly.

Also because we have used the PWM port, by changing the duty cycle the average value of the control that you are sending to the motor is changing. By doing this, the average speed of the motor can be controlled.

(Refer Slide Time: 04:19)

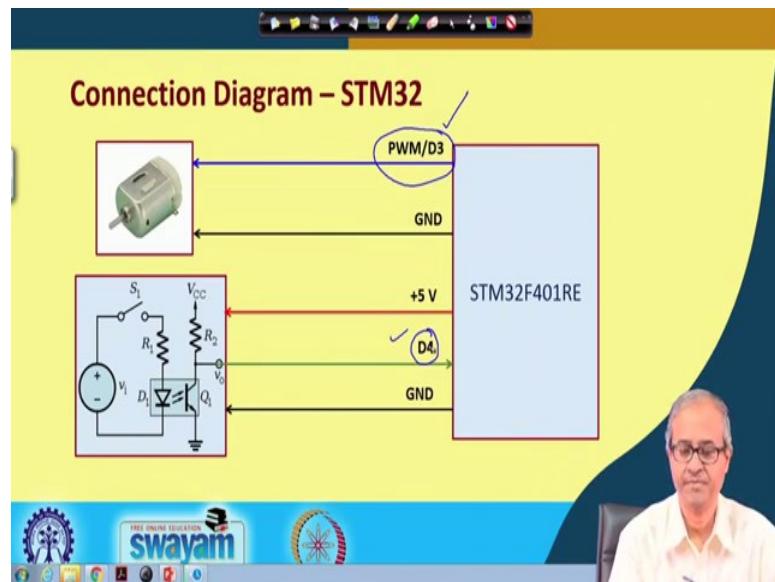


Let us see how this optical interruption works using this opto-coupler circuit. You see basically an opto-coupler is the arrangement of two things: one is some kind of a light source, well in the circuit that we are using this light source is an infrared light emitting diode (IR LED). It emits a light, but in the infrared frequency band. And on the other side, there is a photo transistor. A photo transistor is a kind of a transistor, where there is no base connection like in a normal transistor, there are two terminals only. But the base is replaced by an optical mechanism, whenever there is a light the transistor turns on, and when there is no light, the transistor is off.

So, whenever this IR LED throws a light on this transistor, the transistor conducts and there will be a current flowing path from 3 to 4. So, the circuit that you can use is something like this. In the LED part there will be some kind of a LED driving circuit. As you know, it will consist of a resistance in series with a power supply and this you can switch on and off as required. And on the other side, there will be another circuit using a resistance and this transistor, whenever the transistor is conducting this output voltage will be low, and when it is off this output voltage will be close to V_{CC} .

So, this opto-coupler mechanism looks like this. You see there is a small hole in between and the wheel is actually placed in between those two plates here. This is how it is arranged.

(Refer Slide Time: 06:22)



Now, talking about the connection diagram it is fairly simple. This motor will be driven directly from a PWM output port. Here we have used the D3 port, there is another terminal to connect to ground. And for driving this motor or speed sensing using this opto-coupler circuit, I am using this 5 volts and ground that are required. And this output of the opto-coupler is fed to digital input pin default, because this will be a digital output, either 0 or 1, indicating an interruption or no interruption.

So, I am connecting it to D3 for controlling the motor rotation, and D4 for sensing the optical interruption from the opto-coupler. In our first experiment we shall be using only D3, in the second experiment we shall be using both D3 and D4.

(Refer Slide Time: 07:31)

Experiment 1

- Interface the motor and a push switch. On successive presses of the switch, the following will alternate repeatedly:
 - Run the motor at high speed
 - Run the motor at low speed
 - Turn off the motor

- The push switch is interfaced to port line D2.
- PWM control is used to drive the motor.

+5V

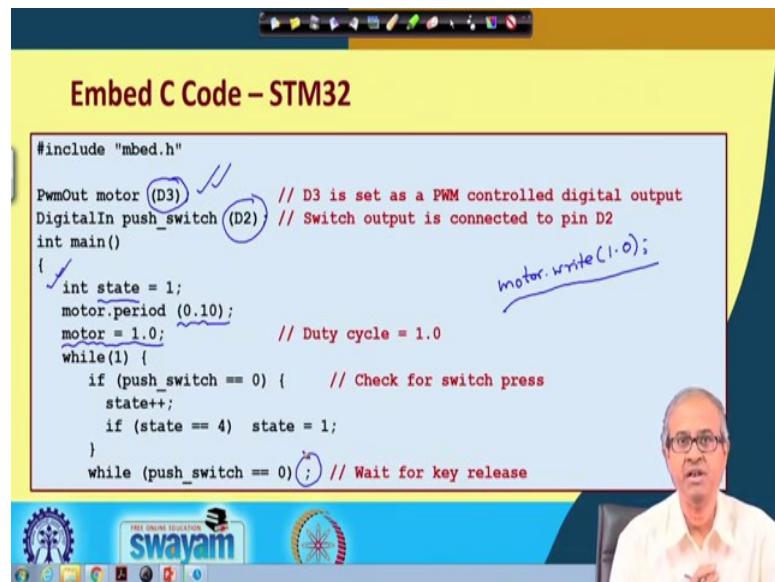
D2

5V power source, resistor, push switch, diode (D2), and ground connection.

Let us talk about the first experiment. In the first experiment, we interface the motor using the circuit that you have just now seen; in addition, we are also interfacing a push button switch like this. If the switch is not pressed, the output point will be high, if the switch is pressed it will be low; it will be connected to ground. So, whenever switch is pressed the switch output, which is connected to port line D2, will become 0, and if the switch is not pressed, it will be 1.

The experiment goes like this. We would be continuously pressing the switch; initially the motor will be rotating at some speed. Whenever we press one switch, there are 3 different levels of speed, one is the maximum speed, one is the medium speed and the other is off; motor will be turning off. So, when I go on pressing the switch, maximum, medium, off, this cycle will repeat.

(Refer Slide Time: 08:49)



Let us look at the program code, which is a little long and spans over 2 slides.

As I said, for motor control we are using PWM control output line D3, depending on the duty cycle the speed of the motor will vary. And the output of the push switch is connected to D2. This is a DigitalIn type of input. In the main program, we have defined a variable called “state”, initialized to 1. It actually indicates in which state the motor is in, because there can be 3 states, maximum speed, medium speed and off.

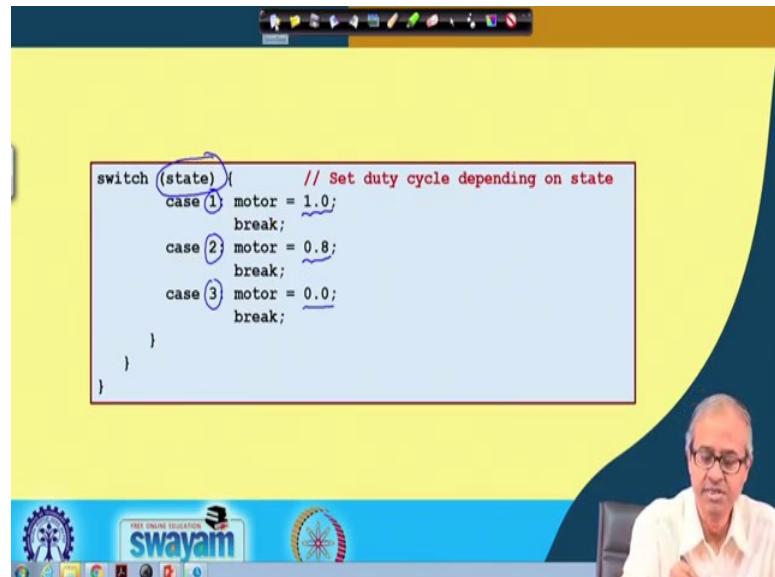
I am initializing state = 1, it will go on like 1, 2, 3, again. Now, `motor.period` is set to 0.1, which means 100 millisecond. So, with 100 millisecond period, I repeat the process. And initially I set the duty cycle to the maximum 1.0. `motor = 1.0`, this is equivalent to: `motor.write(1.0)`. So, initially it is rotating with the maximum speed.

In this while loop, we are checking for a switch press; if a switch is pressed so how do you check? If the push_switch signal is 0, it means switch has been pressed. If it is 0 then increment state by 1; then I check if it has become 4, because it can be only 1, 2, 3. So, if it tries to become 4, you again bring it back to 1. So in this loop, it will go along 1, 2, 3 again 1, 2, 3 like this.

And after this switch press is detected and you have incremented state, you wait till the switch is released. So, as long as switch remains 0, we wait in a dummy loop. So, if we

keep the switch pressed for a long time, it will wait in this loop. As you release it then only the next motor speed will take effect.

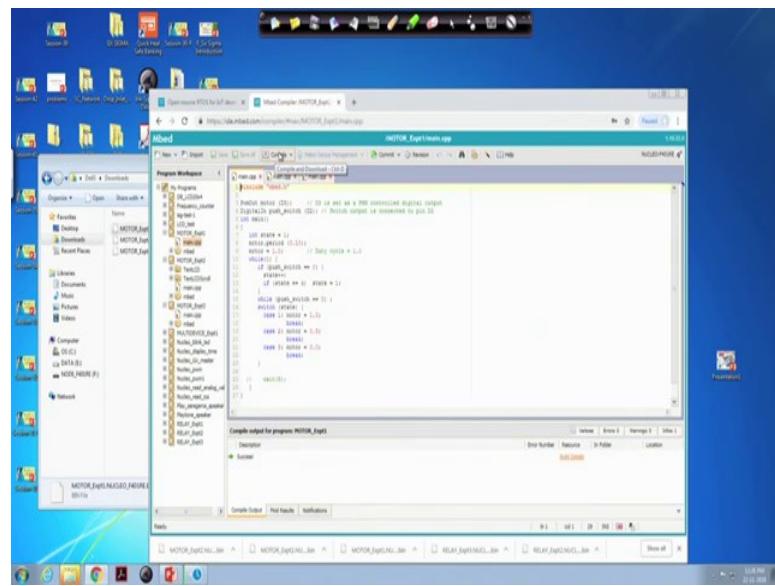
(Refer Slide Time: 11:48)



The next slide actually shows you how the speed control is done. There is a switch case statement. There is a switch statement, which actually checks the value of variable “state”, it can be either 1, 2, or 3. If it is 1, then you are using the maximum duty cycle of 1.0, if it is 2 it is 0.8 or medium speed, and if it is 3 then it is 0.0 or off. This thing goes on in a cycle.

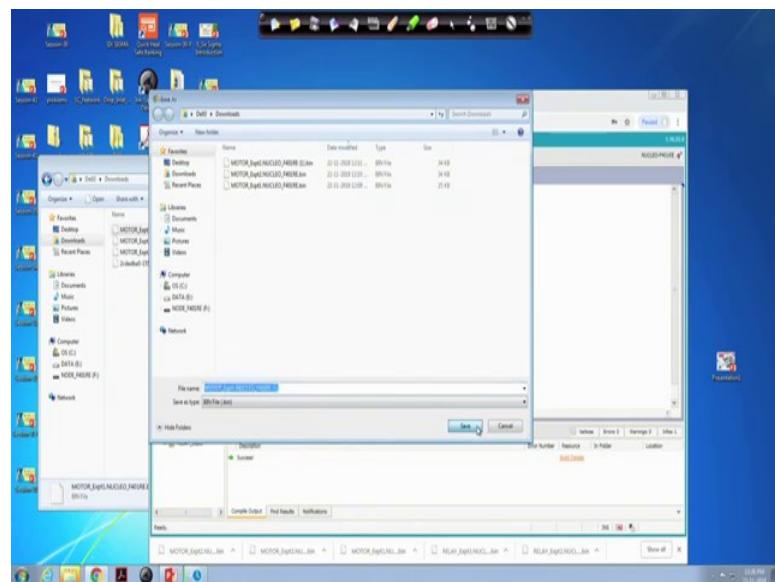
Let us now see the demonstration.

(Refer Slide Time: 12:46)



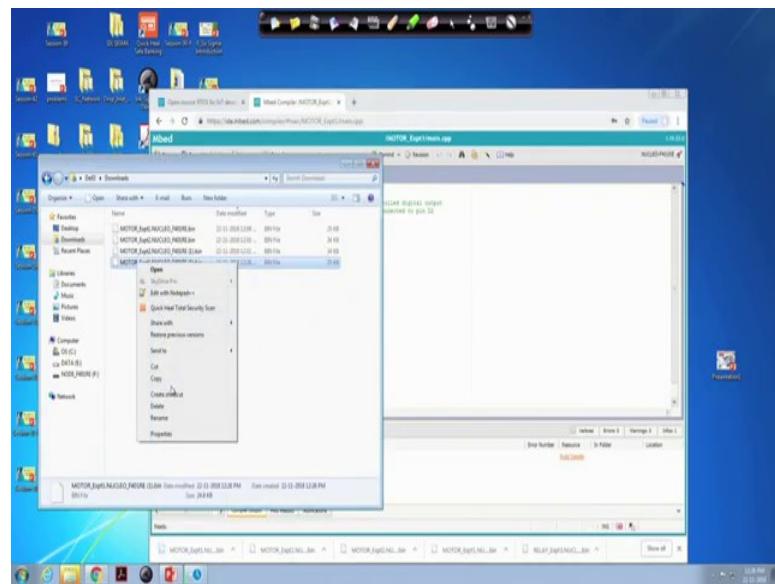
This is the experiment that we have discussed now, let us compile it.

(Refer Slide Time: 13:07)



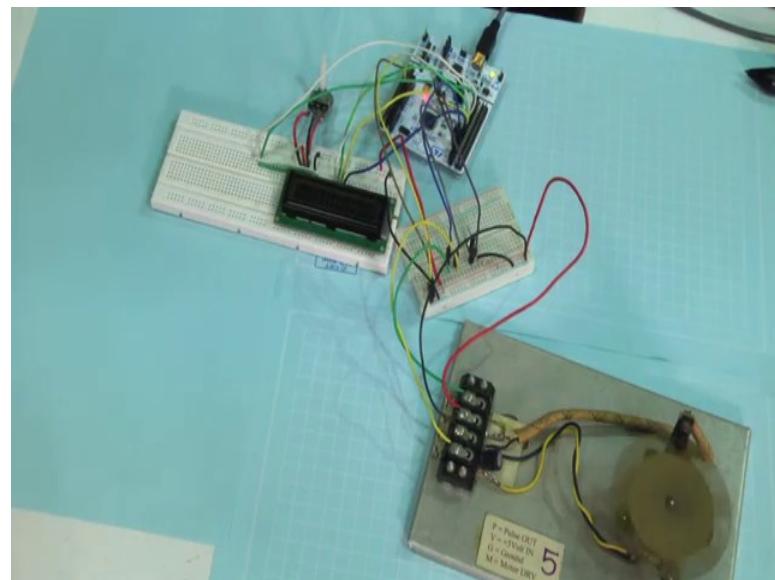
Save it.

(Refer Slide Time: 13:10)



Copy and paste.

(Refer Slide Time: 13:18)



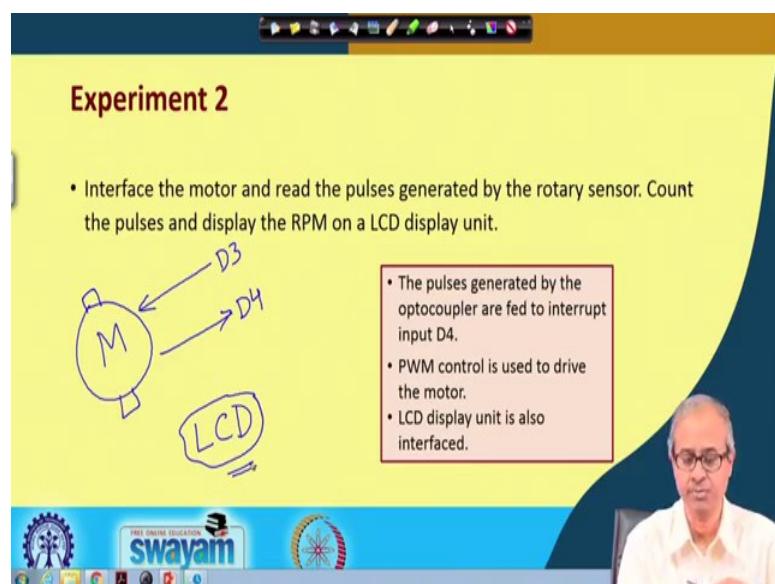
Now see, this motor starts rotating. You can see here that the motor is rotating and it is the maximum speed, and here I have interfaced a small push switch. You can see the push switch, and one terminal the push switch is connected to this resistance to Vcc, and the other terminal to ground. And the middle point of the switch is connected to the input line D2.

Regarding the interface of the motor, you see there are some terminals. This is 5 volts, which is connected to the 5 volt point, this black wire is GND, and this yellow wire is the PWM control, which is connected to D3. And this other line green one, this we shall be using in the next experiment, this will be using the opto-coupler output. But in this experiment we are not using the opto-coupler.

You see motor is rotating in the full speed. If I press this switch once the speed has decreased, now it is medium. If I press another time, motor will turn off. I press once more, it will again start rotating in the maximum speed, then I press once more medium speed. This is maximum speed, medium speed, off. This cycle will repeat.

Let us continue.

(Refer Slide Time: 15:29)



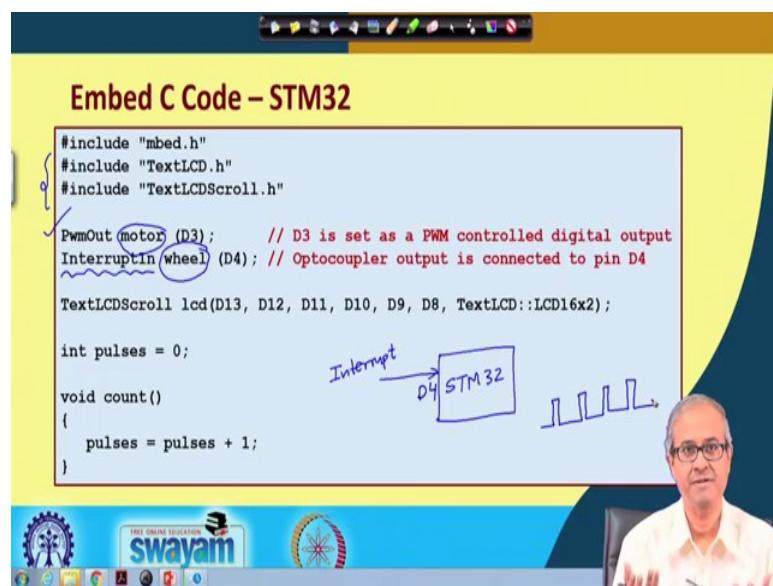
Now in the second experiment, we are doing certain things. Now you think of the motor, you have the motor out here; this is the symbol of a motor. Now normally what we have done, we are using one control line from the microcontroller to control the speed. So, that we have connected to the PWM output line D3.

Now in this experiment, we are also using the output of the opto-coupler, because we want to measure the speed of rotation, how many pulses are coming per second or per minute. The optical output we are connecting to digital input line D4. Now after counting the point is how do I know what is the speed, how do I show?

For this reason, we have also interfaced a LCD with the microcontroller board. We are not showing the details of LCD interface, because you have already studied this and saw in some earlier lecture. You know how a LCD is interfaced.

Now this is another point I shall be explaining when I show you the program.

(Refer Slide Time: 17:09)



As I have already said, this mbed.h is included and for LCD interface, I need to interface TextLCD.h. Of course, we do not need scroll, so the third one is not required, but I have also used TextLCDScroll.h. And for motor drive, I have connected to D3, and D4 is the input that is coming from the output of the opto-coupler.

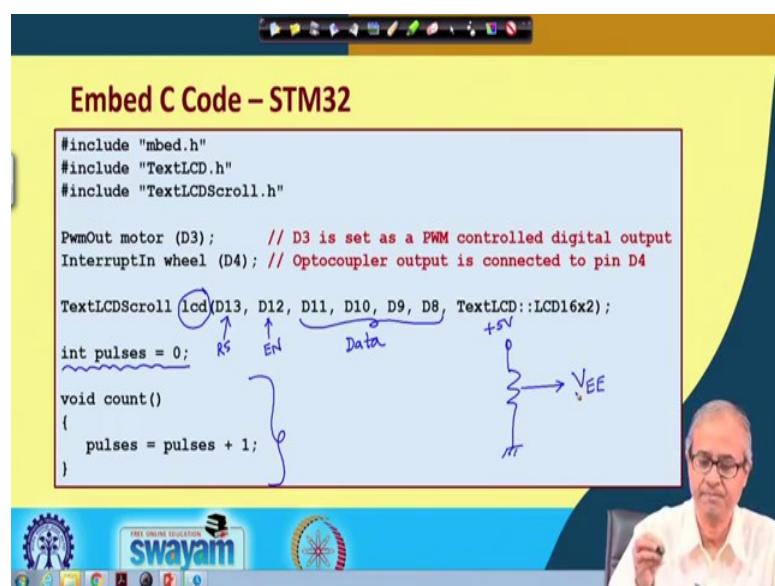
Now, let us try to understand. When I write a program, how do I count? Suppose, I want to count, how many such pulses are coming per second. The way we have implemented it in our program is that we have used the concept of interrupt. What is an interrupt? Interrupt is a mechanism like this; suppose, this is our processor board and from outside some interrupt is coming. Here, we have connected it to the digital input line D4.

When we discussed interrupts earlier, you recall I told you that any of the digital IO lines can be used as an interrupt pin. So, here we are using this D4 as an interrupt input. How do I declare that? I declare it as instead of telling it as DigitalIn, I mention it as InterruptIn, and the name of this object I have given as “wheel”, and for PWM out I am calling it “motor”.

Now, the point is that for interrupt what really happens? Means, if you have studied it earlier somewhere you know that when an interrupt signal comes to a processor, whatever the processor was executing is temporarily suspended and the control jumps to another program routine called interrupt service routine. After execution of the interrupt service routine, control comes back to the program that was suspended. This is how it works.

Now in this case, every time there is an optical interruption ... for one revolution, there will be 8 such interruptions. So, there will be some pulses coming like this. Every time there is an optical interruption, there will be a pulse and each of these pulses will be generating an interrupt signal. So, the interrupt service routine will be called so many times.

(Refer Slide Time: 20:05)



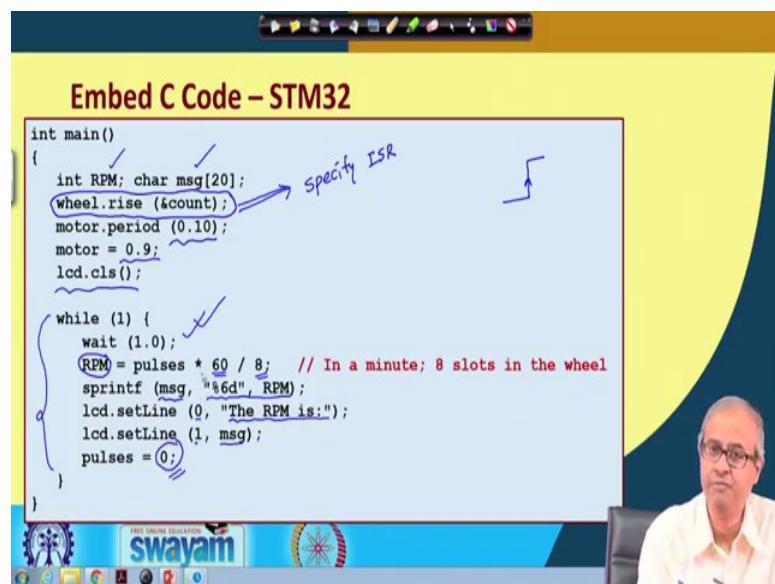
What we have done here is this. See, this is actually the interrupt service routine, this count function, I will show you how we declare it as an interrupt service routine. But, suppose for the time being this is the interrupt service routine, what I have done: we have declared a global variable integer variable called “pulses”, initialized to 0. And in the interrupt service routine, we are just increasing “pulses” by 1; so that whenever interrupt comes the variable gets incremented by 1.

The other thing to note is that we have also interfaced an LCD and these are the standard interfaces, you recall the first of these is register select, second one is enable, and last 4 are the most significant data lines for 4-bit interface.

So, we have connected the pins from the LCD to D8, D9, D10, D11 and this enable to D12 and RS to D13. And of course, Vcc, GND and you recall there is a potentiometer that we used like this, we connect it to another pin called VEE for LCD contrast arrangement. This potentiometer circuit is also there.

Now I told you this is the interrupt service routine.

(Refer Slide Time: 21:53)



Let us see how this interrupt service routine is mentioned. This is our main function. First thing is that you note this is the place where we are specifying the interrupt service routine. We are saying that wheel is that object, which is connected to the interrupt line that we called “wheel”, rise is a function means it checks every time the signal rises from 0 to 1. Whenever there is a rise, you call this function; &count means this is a pointer to a function. Recall count was the name of that function, you call count every time there is a rising edge on wheel.

That is how interrupt service routine gets called, and we have declared a variable called RPM, where you are counting revolutions per minute. There is a character array called msg. For the motor driving, we have assumed a period of 100 milliseconds. As in the

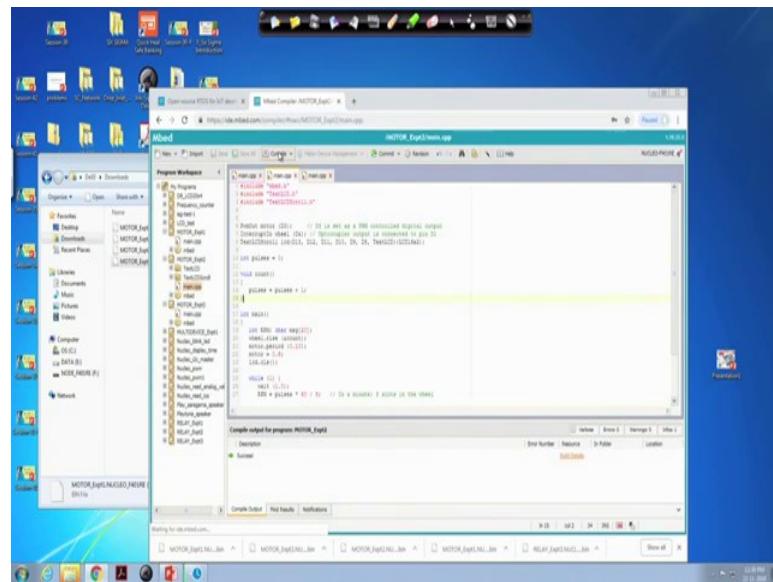
previous experiment, the duty cycle is set to 0.9 and cls function initially clears the LCD screen.

In the while loop, initially pulses is initialized to 0, now we wait for 1 second; that means, in this 1 second the motor is rotating. So, how many pulses are coming in this one second will get counted; pulses will get incremented by so many times. At the end of it we calculate revolutions per minute because, we have counted for 1 second, for 1 minute, it will be multiplied by 60, and because there are 8 holes in that wheel, we have to divide it by 8. This will give you your revolutions per minute or RPM value.

And what we do we sprintf means you can print this value into a string, you are printing the value of this RPM as an integer into this character string. Then, we are displaying on LCD.

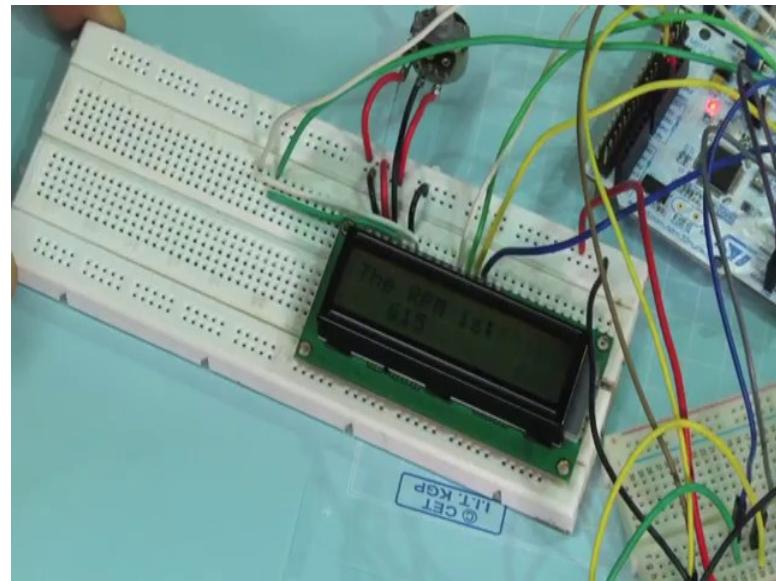
And after we do this we reinitialize the value of pulses to 0 so that in the next cycle we again carryout with the counting and we repeat this process. Again wait for 1 second, again the pulses will get counted for 1 seconds, again we calculate RPM and print it. This is the program, and let us see the demo now.

(Refer Slide Time: 25:14)



This is the same program that I have shown, this I am trying to compile. So I compile it, save it, copy it and paste it. Now, let us see what is happening.

(Refer Slide Time: 25:50)



See, in the program we have set the duty cycle to 0.9. So, see the motor is rotating at a very high speed, high speed. Now the switch circuit is not used in this experiment. So, this switch is not read, but you look at this LCD display here.

This is the potentiometer for controlling the contrast, so you can adjust it for a suitable contrast. If it is showing the RPM is 1095, you see this motor is a very cheap and small motor, so its speed is not very stable. So, it is varying a little bit, so 1027, 1100, 1095. So, the RPM is going like this. You can view this RPM value here.

Now let us see if I change the duty cycle does the RPM value changes, let me see this. You see here, the motor duty cycle was set to 0.9. Let me change it to 0.9 to 0.7 let us say, let me change it to 0.7. Let us save it, compile it again and we save this, copy, paste. So now, see the same thing is happening, but now the motor is rotating at a much slower speed. If you look into the motor, it is rotating at a slower speed. And now, you see the RPM value displayed is showing 600. So, the RPM value dropped significantly you see.

So, in this way you calculate, read the value, and if you find that the RPM value is lower, you increase the duty cycle a little bit and again read; if it is lower you again read. See earlier, we talked about this integral control on-off control, PID control all these things, so all these things can come into the picture here. But, because the program will become complex, we are not showing those in this demonstration, but you can also have it; you

can set a preset RPM value, and you can adjust the RPM adjust the motor drive to make it work accordingly.

With this we come to the end of this lecture, thank you.

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

Lecture – 33
Experiments with Multiple Sensors and Relay

If we recall in the last two lectures we have seen that how we can interface devices like relays, DC motors etc. to the microcontroller, and we can do some kind of a control and also we can sense the values and so on. Now in the experiment that we shall be showing you in this lecture, we shall demonstrate that not only one device or one sensor, the microcontroller is powerful enough to control multiple devices at the same time.

(Refer Slide Time: 01:05)



Here we shall basically be looking at how to interface multiple devices, multiple sensors and multiple output devices.

(Refer Slide Time: 01:21)

Introduction

- In this experiment, we shall consider the interfacing of multiple sensors and multiple output devices.
 - Input devices: LDR, LM35 temperature sensor
 - Output devices: Relay, driving a bulb, Speaker
- We shall demonstrate how the same microcontroller can be used to perform multiple tasks in a time multiplexed way.
- The experiment is also realistic enough such that it can be related to practical scenarios of home automation.

Let us talk about the experiment first. In this experiment as it said we shall be interfacing multiple sensors and multiple output devices; specifically the input devices that we shall be looking at are LDR for sensing ambient light and a LM35 temperature sensor for sensing the temperature of the environment.

Now, the output devices that we shall be interfacing are one relay circuit that will again be driving a LED bulb as we have seen in the earlier lecture, and a speaker. Now the kind of an environment that we are trying to show you is suppose first one is the one that we have already shown earlier depending on the ambient light you can automatically switch on a switch off a bulb through a relay.

And secondly, there can be something like a fire alarm system, there will be a system which will be sensing the temperature and whenever the temperature crosses a certain preset threshold an alarm that will be sounded, and for that alarm we have interfaced a speaker circuit.

(Refer Slide Time: 03:03)

The Experiment

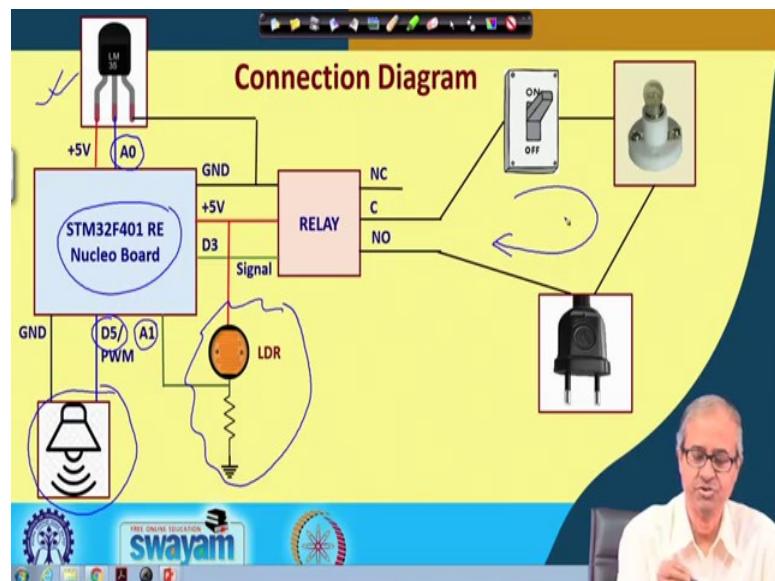
Interface relay, speaker, LDR, and LM35 to the STM32F401 Nucleo board:

- The relay circuit is used to drive a bulb (driven by D3).
- The speaker is interfaced to generate an audible output (driven by D5 using PWM).
- The LDR circuit is used to sense the level of ambient light (connected to A1).
- The LM35 sensor is used to measure the temperature (connected to A0).
- If the ambient light falls below a threshold, the bulb will turn on.
- Whenever the temperature crosses a threshold, an alarm will sound.

Specifically in this experiment we will be interfacing a relay, a speaker, a LDR and LM35. The relay will be driven by digital port line D3. This speaker will be interfaced from port line D5, the LDR circuit output will be connected to analog input A1, and the LM35 sensor will be connected to analog input A0.

Now, if the ambient light input falls below a threshold, the bulb will turn on. Also, whenever the temperature crosses a threshold, the alarm will sound.

(Refer Slide Time: 04:05)



Let us look at the connection diagram. First this is the STM board, and this is relay circuit that is the same as we had seen earlier. On one side we have the LDR circuit that will be sensing the ambient light; there is a resistance divider the output of which is connected to the analog port line A1. On the other side you have the LM35 temperature sensor that will be generating the output on analog input line A0, and for alarm you have a speaker that is connected to D5, which is a PWM control output port.

Whenever the temperature crosses a threshold the speaker will be sounded, and whenever the light falls below a level the relay will be activated.

(Refer Slide Time: 05:07)

Basic Program Logic

- The steps as shown must run in a repetitive loop.
- There can be a delay before the loop repeats.
- First, the LDR generated sensor voltage is read through A0.
- Then, LM35 generated temperature data is read through A1.

```

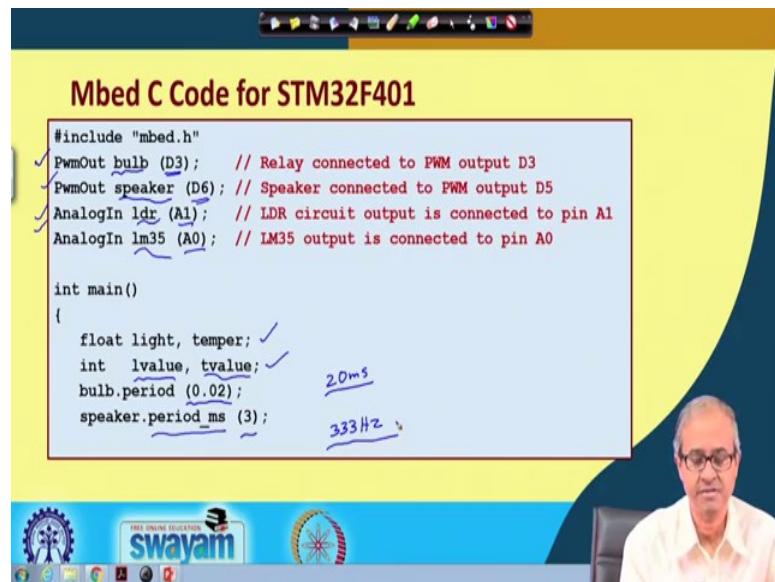
begin
  if (value (A0) > thres_light)
    D3 = 0;           // Turn OFF relay (bulb)
  else
    D3 = 1;           // Turn ON relay (bulb)
    if (value (A1) > thres_temp)
      D5 = PWM tone; // Turn ON alarm
    else
      D5 = No PWM tone; // Turn OFF alarm
end

```

Let us look at the program logic first, then we shall be showing you the code. The steps as shown here will be running in a repetitive loop. First we will have to check the value on the analog input line A0, if it exceeds the threshold level for the light that we are trying to sense you turn off the relay; that means, you have sufficient light otherwise turn on the relay; that means, the bulb will glow.

After that you check similar thing for the temperature; if the value on port line A1 is exceeding some threshold, then you play a tone on the speaker that indicates some alarm, but if it is not there will be no PWM tone; that means, you do not play a tone. Now let us look at the program.

(Refer Slide Time: 06:13)



```
Mbed C Code for STM32F401

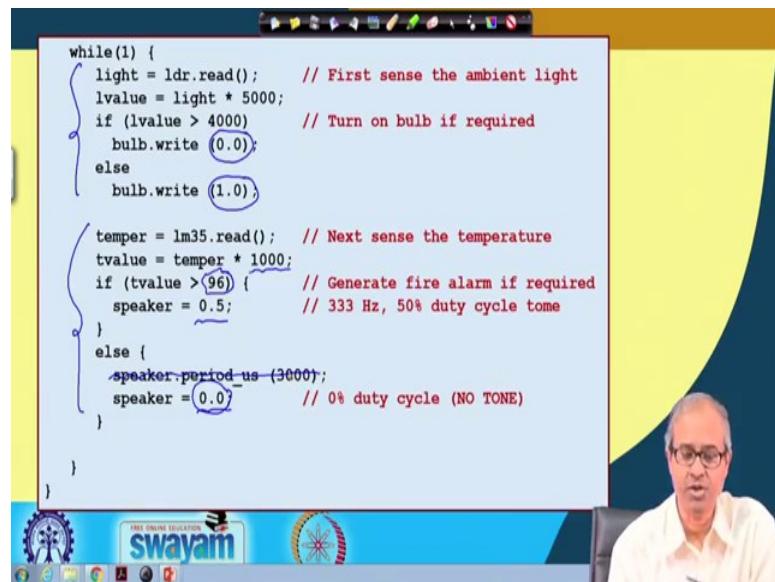
#include "mbed.h"
PwmOut bulb (D3); // Relay connected to PWM output D3
PwmOut speaker (D6); // Speaker connected to PWM output D5
AnalogIn ldr (A1); // LDR circuit output is connected to pin A1
AnalogIn lm35 (A0); // LM35 output is connected to pin A0

int main()
{
    float light, temper;
    int lvalue, tvalue;
    bulb.period (0.02);
    speaker.period_ms (3);
}
```

The program will start like this. So, you are defining a PwmOut line on D3, which you call as “bulb”, then another PwmOut ut on D6, which you call “speaker”. And there are two analog inputs A1 and A0 corresponding to LDR and LM35.

And some variables we have defined light, temper for calculating the analog inputs, and lvalue and tvalue to store temporary temperature value in two variables. And for activating the bulb we have assumed that, we have using 50 Hertz PWM, with time period 20 milliseconds. And for speaker we have set a period of 3 milliseconds that means 333 hertz of frequency will be played on the speaker.

(Refer Slide Time: 07:41)



```
while(1) {
    light = ldr.read();           // First sense the ambient light
    lvalue = light * 5000;
    if (lvalue > 4000)           // Turn on bulb if required
        bulb.write (0.0);
    else
        bulb.write (1.0);

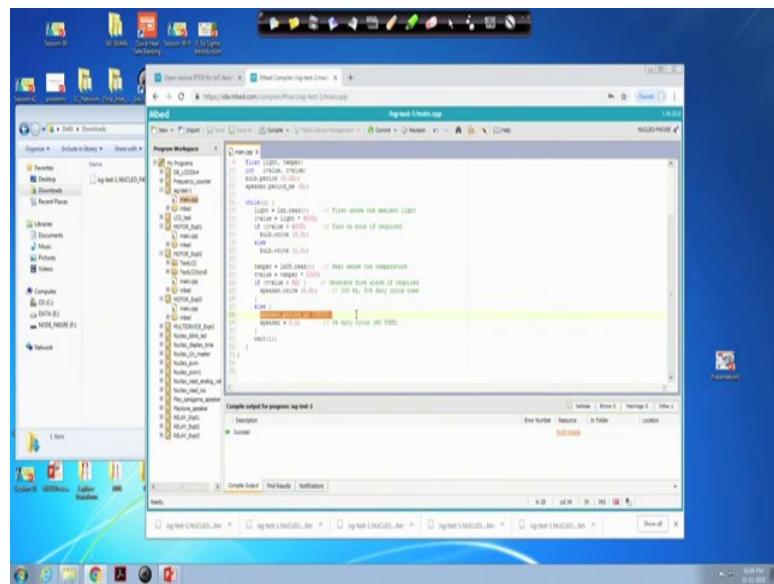
    temper = lm35.read();         // Next sense the temperature
    tvalue = temper * 1000;
    if (tvalue > 96) {           // Generate fire alarm if required
        speaker = 0.5;           // 333 Hz, 50% duty cycle tone
    }
    else {
        speaker.period_us (3000);
        speaker = 0.0;           // 0% duty cycle (NO TONE)
    }
}
```

Then let us come to the rest of the main function. In a repetitive while loop we are checking the light here, we are checking the temperature here. In the light we are using the analog input function `ldr.read()`, then you are multiplying it by a scale factor just like in the program that we showed earlier. If it exceeds some threshold value we turn off the light; if not, we turn on the light.

This is done by setting the duty cycle to 0.0 or 1.0. Similarly for the temperature you are reading the temperature, you are again multiplying it by some scale factor here. I am using 1000, and some threshold that again can be chosen through experimentation; depends on the temperature where you want to start the alarm. Accordingly we play a continuous note on the speaker, but otherwise we set the duty cycle to 0.

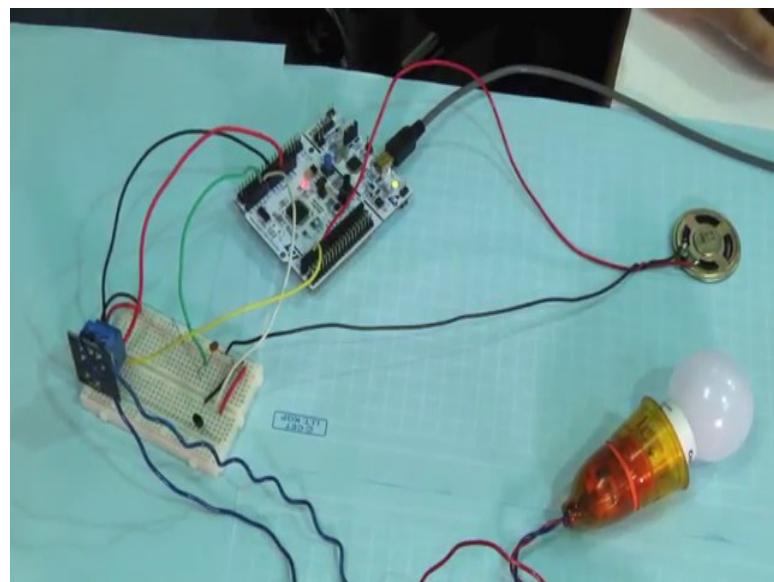
Now, this line is actually not needed this line you can also omit this is not really required because, once you set the duty cycle to 0 the period does not matter ok. Let us look at the demonstration now.

(Refer Slide Time: 09:47)



This line you can omit we do not need this. So, let us compile this code. We compile it, we save it, copy and paste.

(Refer Slide Time: 10:21)

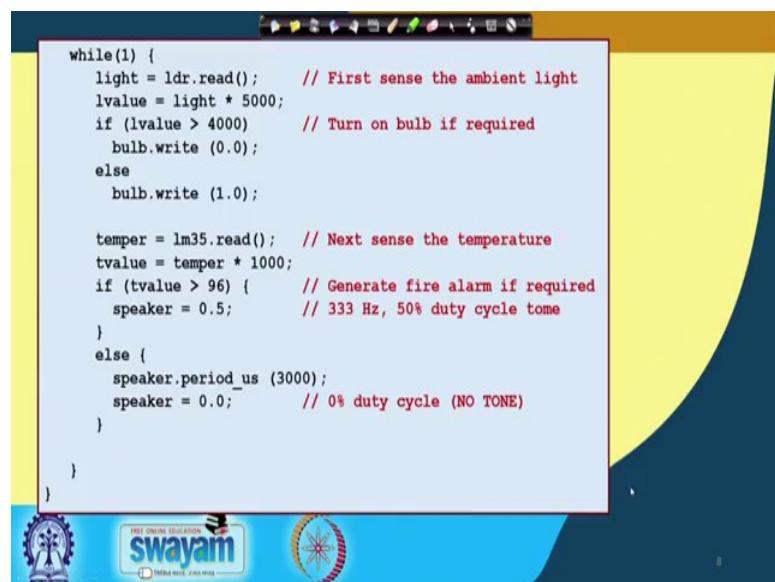


Now, let us come to the circuit. Now see we have used that same light interfacing circuitry as was shown earlier; a relay, a bulb with a switch connected to mains, and we have used the same LDR circuit. You see if LDR is interrupted, the light is glowing. You see if it is released again light will turn off. Here you have the LM35 setting; if

temperature exceeds the threshold, there will be an alarm tone here you see the temperature has just increased beyond a level and this alarm has started to ring.

I am just pressing with my finger, so that the temperature increases that little bit. If you have a heater you can show it in a very clear way. See this is alarm is sounding because it is just crossing threshold, but if we can heat it a little further then there will be a very clear sound that will be coming. So, in this experiment what we have seen is that we can have multiple devices interfaced.

(Refer Slide Time: 12:15)



```
while(1) {
    light = ldr.read();      // First sense the ambient light
    lvalue = light * 5000;
    if (lvalue > 4000)      // Turn on bulb if required
        bulb.write (0.0);
    else
        bulb.write (1.0);

    temper = lm35.read();  // Next sense the temperature
    tvalue = temper * 1000;
    if (tvalue > 96) {      // Generate fire alarm if required
        speaker = 0.5;      // 333 Hz, 50% duty cycle tone
    }
    else {
        speaker.period_us (3000);
        speaker = 0.0;      // 0% duty cycle (NO TONE)
    }
}
```

When you talk about home automation, in your home there can be so many kind of devices connected. Now these microcontrollers are powerful enough, such that a single such device will be able to control the entire thing. The only thing is that you have to write your program in such a way some of the devices will be sending the inputs and interrupt driven mode some of the devices you can just read them one by one.

It depends on the type of devices and the way they are sending you the inputs. We shall be seeing more demonstrations on these in the later lectures, where you will also be looking at how the devices can communicate with the outside world.

Thank you.

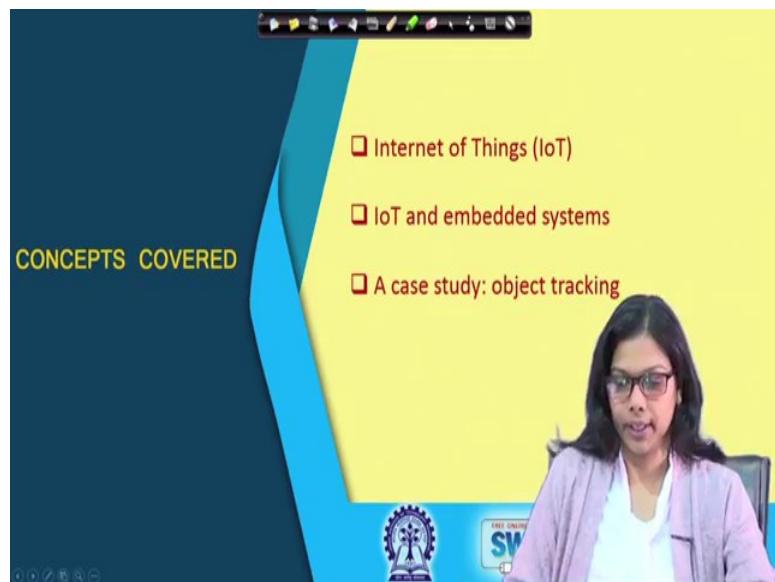
Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture - 34
Introduction to Internet of Things

In this week, I will be introducing you to a concept called Internet of Things. Of course, I will be discussing about what is internet of things giving you some examples. Then we will be taking two examples. One is a small home automation system, where we will switch ON and OFF a light through SMS control.

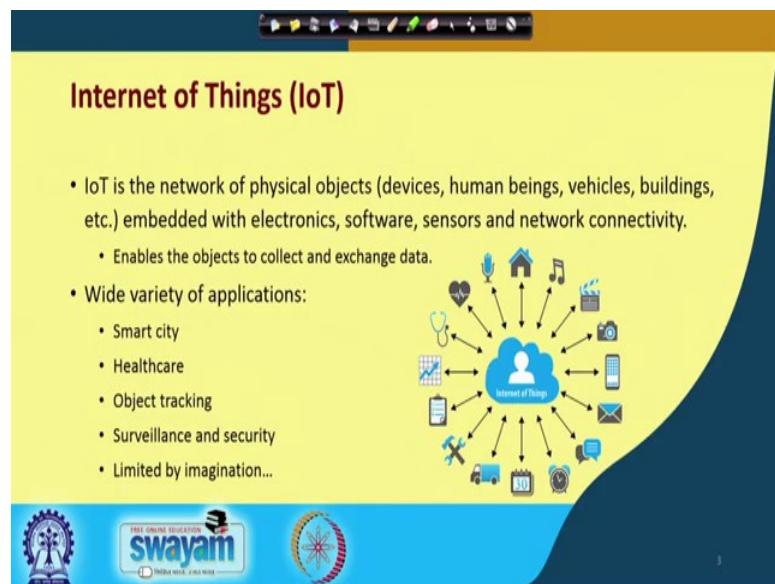
And in this week, we will also show you through a touch sensor, how we can sense some kind of events, if an unwanted people touches your screen. So, these are the two examples that we will look into. And in this lecture particularly, I will introduce you to the buzz word internet of things.

(Refer Slide Time: 01:29)



The concepts that will be covered in this week is what is internet of things, how we relate IoT and embedded systems, and we shall take a case study of object tracking.

(Refer Slide Time: 01:49)



Internet of Things (IoT)

- IoT is the network of physical objects (devices, human beings, vehicles, buildings, etc.) embedded with electronics, software, sensors and network connectivity.
 - Enables the objects to collect and exchange data.
- Wide variety of applications:
 - Smart city
 - Healthcare
 - Object tracking
 - Surveillance and security
 - Limited by imagination...

Internet of Things

Coming to what is internet of things, it is the network of physical objects, like it could be any device, it could be a human being or a vehicle, or it could be a building, embedded with electronics, software, sensors, and network connectivity. And what it enables, it enables the object to collect and exchange data. Let me give you an example.

In week 5, we discussed about temperature sensing module. So, what we were doing in that example, there was a sensor that is LM35 from where we were sensing some physical parameter that is temperature, and we were displaying it in the LCD. This is what we were doing, that is an embedded system application, which is a standalone system. It has got a communication capability, but we were not transmitting the data that we received from the sensor to any other server or any other database. So, we were not storing that temperature anywhere, we were just displaying it on the LCD.

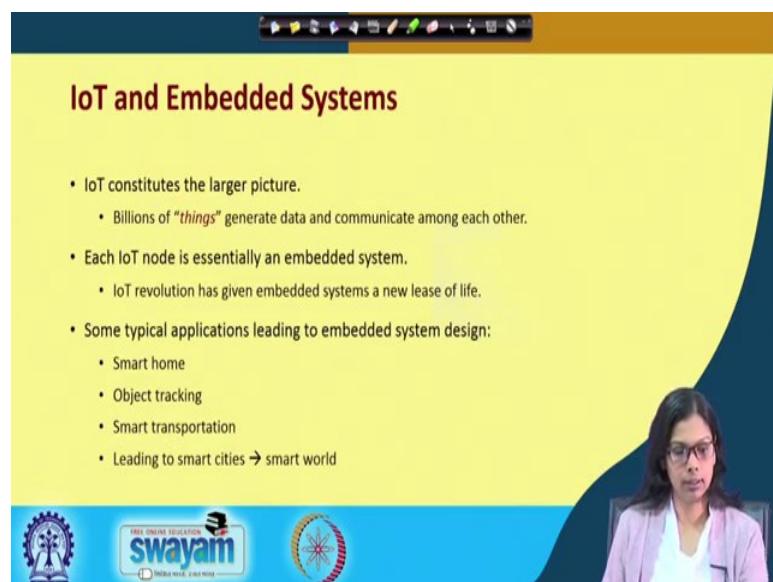
But, let us say a scenario where I gather that temperature from the sensor through microcontroller, and I pass that information or store that information in some database. If we want to save that into a database which is online, we need to have some communication capability along with the processing capability that it has got. It has got the processing capability already, but now for sending the data or storing the data in some server or in some database, we need some communication capability as well.

Whenever an embedded system not only has processing capability, but also has communication capability, it becomes an IoT device. That means, the temperature

sensing unit is not standalone anymore and not restricted to this particular place, where you are displaying the temperature only. If somebody wants to know the temperature of this particular room, you can get it through the mechanism I told you. It enables the objects to collect and as well as exchange data.

There is a wide variety of applications that we can have, one of which is smart city, another is in healthcare. Object tracking is one of the very major applications, where nowadays people are using small modules to track any object, we will look into that as a case study later. In surveillance and security, IoT has got very good applications. For security I will be taking one example with that sensor, but it has got many other applications. And of course, it is limited just by imagination, there is a huge set of applications. When an embedded system also has got communication capability, we call it as an IoT.

(Refer Slide Time: 06:13)



IoT and Embedded Systems

- IoT constitutes the larger picture.
 - Billions of "things" generate data and communicate among each other.
- Each IoT node is essentially an embedded system.
 - IoT revolution has given embedded systems a new lease of life.
- Some typical applications leading to embedded system design:
 - Smart home
 - Object tracking
 - Smart transportation
 - Leading to smart cities → smart world

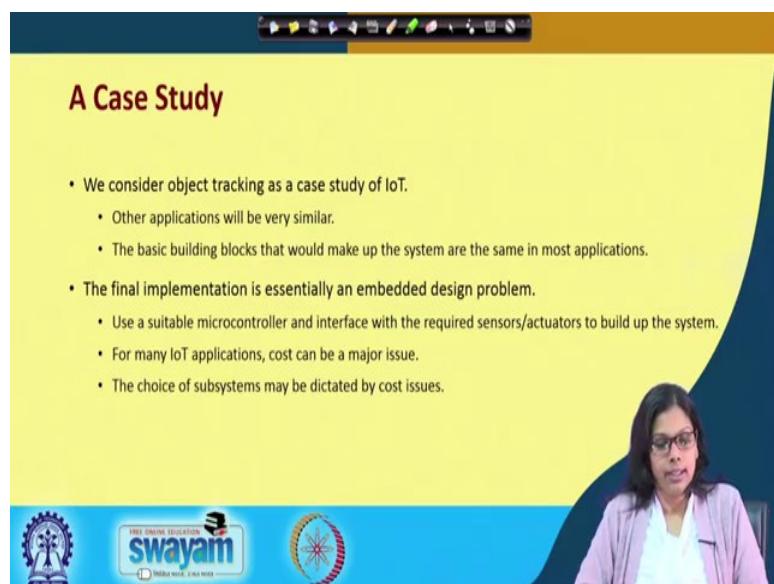
Let me tell you about IoT and embedded system, I have already told you the relation between the two. IoT constitutes the larger picture, of course it is not only one embedded system, it is a collection of many embedded systems that can communicate among each other as well.

There are billions of things in the world that can generate data and communicate with each other. It generates data from these sensors, a lot of data. And where this data will go, the data will be saved somewhere for further processing. So much data gets

generated, it is not very easy to store that huge amount of data. Of course, you need to have some architecture in place to save the data properly. But, in some cases, if the sensors are generating some unnecessary data which need not have to be kept for you know for all the period, then you can simply delete those data.

Each IoT node is essentially an embedded system, I have already told you. And this IoT revolution has given this embedded system a new lease of life. Embedded system was not new, but this buzzword IoT is there for quite some time let us say last 5 years. This IoT revolution has given the old embedded systems a new lease of life you can say. Some typical applications leading to embedded system design could be smart home, object tracking, smart transportation, and of course if you combine many of the aspects of various IoTs, then it leads to a broader picture we call it smart city, and ultimately to smart world.

(Refer Slide Time: 08:27)



A Case Study

- We consider object tracking as a case study of IoT.
 - Other applications will be very similar.
 - The basic building blocks that would make up the system are the same in most applications.
- The final implementation is essentially an embedded design problem.
 - Use a suitable microcontroller and interface with the required sensors/actuators to build up the system.
 - For many IoT applications, cost can be a major issue.
 - The choice of subsystems may be dictated by cost issues.

Now, we will consider a case study, where we will consider an object tracking module. Firstly I will talk about that what exactly we are trying to say, when we say we track an object. I could be an object, my bag could be an object. Let us say I am sending something from one place to another like a medicine, that could be also an object.

My children are going outside for playing, they can also be treated as objects. There are other applications which will be similar. We will see the basic building blocks that make up the system and are the same in most of the applications.

The final implementation is essentially an embedded system problem. If you think of an overall system of object tracking, first of all you have to build an embedded system. And then of course, it will have communication capability.

In the first step, we use a suitable microcontroller and interface with the required sensors or actuators to build up a system that is very straightforward. So, we need a suitable microcontroller for the processing, and also we need some sensors or actuators to build up the entire system.

For many IoT application, the cost can be a major issue. Now, all these things we have already discussed in the previous weeks that cost depends on how many number of units you are actually manufacturing. If you are really manufacturing a very small number of units, then the cost might be large. But, if the same kind of product is designed for a large number of units, you are producing it in a bulk, then the cost always reduces.

The choice of the subsystem may be dictated by cost issue. It depends like for how many units you are producing, which community wants that device, how will you market it such that the importance of the device goes to the huge customers. If the customer wants to buy, then you have to also do some kind of marketing. So, what kind of things you should do to sell your product, there are lot many things that are involved, but cost is an important factor.

(Refer Slide Time: 12:07)

Introduction to Object Tracking

- What is object tracking?
 - A mechanism to locate any object in real time.
 - An object can be any conceivable thing that changes its location over time:
 - a) A vehicle (bicycle, car, bus, truck, airplane)
 - b) A human being
 - c) An animal (cow, dog, wild life, etc.)
 - d) A bag
 - e) ... and many more

FREE ONLINE EDUCATION **swayam** INDIA WIDE, 24x7x365

Now, let me come to object tracking. Firstly, what is object tracking? It is a mechanism by which we can locate any object in real time, meaning let us say my kid has went out for playing at any point of the time I want to know, where my kid is playing, whether he is inside the playground or he has moved out to some other place in real time. In this process we can have some kind of application maybe it as a web application or an app in your mobile phone, it should able to tell me the current location of my child. What is required to be done, I will tell you one by one, but this is where I need this object tracker.

An object can be any conceivable thing that changes its location over time, it could be a vehicle like it could be my bicycle, my car or a bus or a truck or an airplane, or it could be a human being, as I said it could be a small kid or it could be your old parents, it could be any animal like a cow, your dog, some wildlife etc.

(Refer Slide Time: 14:19)

Why do we need object tracking?

Example 1 :: Delivery of medicines

- It is required to deliver some medicines under some required environmental conditions.
 - a) Hand over the medicines to some delivery agency.
 - b) Along with the medicines, attach some device that will sense the temperature, humidity and other relevant environmental conditions, along with its location.
 - c) The information from the various sensors shall be uploaded to a server periodically through some suitable communication mechanism (e.g. GPRS, SMS).
 - d) The concerned persons can track the consignments in real time, and may also have access to historical data.

So, why do we need object tracking? I think I have already motivated you like why do we need object tracking, but let me give you some more examples, where you will see that just tracking the object may not make sense. We also need some more some more other sensors to be attached to the object to receive some more data.

Let us take the example as delivery of medicine. Let us say it is required to deliver some medicine, under some required environmental conditions. You know medicines need certain temperature. If you want to keep the medicine in a very hot place, it might get damaged. So, you need to maintain the temperature.

So, what is generally done, if you want to send a medicine from place A to place B, we hand over the medicine to some delivery agent. Along with a medicine, we attach some device that will sense the temperature, humidity, and other relevant information along with its location as well. So, it is not only sending the location where this medicine is going through, you can also track apart from that what is the current temperature during that time etc.

The information from the various sensors shall be uploaded to a server periodically through some suitable communication mechanism. The mechanism could be GPRS or SMS, we will look into these things.

And it will upload both the location along with these other parameters into some database. We can do this of course through SMS, it will be sent to some other mobile number. But, if you want to send it through to internet, then you have to use the GPRS service. So, this is how it basically works.

The concerned person can track the consignment in real time, of course if all the data is uploaded time to time let us say every 5-minutes, we are uploading the devices sending the data to a server, then the concerned person can easily track the consignment in real time; and may also have access to historical data, because we are storing it in a database. So, you can also find out let us say I am checking right at this moment, but what happened to one hour before, what was the temperature that also we can see, because we have stored everything in the database.

(Refer Slide Time: 18:19)

Example 2 :: Tracking of living beings

The mechanism is very similar; only the sensors that are required shall vary from one situation to the next.

- a) Tracking children when they are playing outside or playing.
- b) Tracking wild life to gain information about their lifestyle.
- c) Tracking of pets and cattle.



Let us take the example of tracking living beings. The mechanism is very similar as I have discussed for medicine, it will be same here, only the sensors that are required shall vary from one situation to the next.

Tracking wildlife to gain information about their lifestyle, there could be many other information we can gather. You can have some device like accelerometer, we look into that in course of time. So, what is an accelerometer we will see that, so that kind of device you can put in to find out the activities that a particular animal is performing in a day let us say. We can always track our pets or any cattle in that way.

(Refer Slide Time: 19:13)

Requirements for building an object tracker

There are four basic requirements for developing an object tracking system:

- a) **Processing** :: A microcontroller is required for carrying out some local processing, and interface with the various sensors and the communication subsystems.
- b) **Communication Capability** :: The system must have some mechanism to communicate with some central server (e.g. GSM/GPRS module).
- c) **Location Sensing** :: The most important criterion of an object tracker is to accurately locate its position. A GPS module can be used for this purpose.
- d) **Parameter Sensing** :: Depending on the application, various physical parameters like temperature, humidity, pressure, etc. may need to be sensed. Such sensors must be interfaced with the microcontroller.

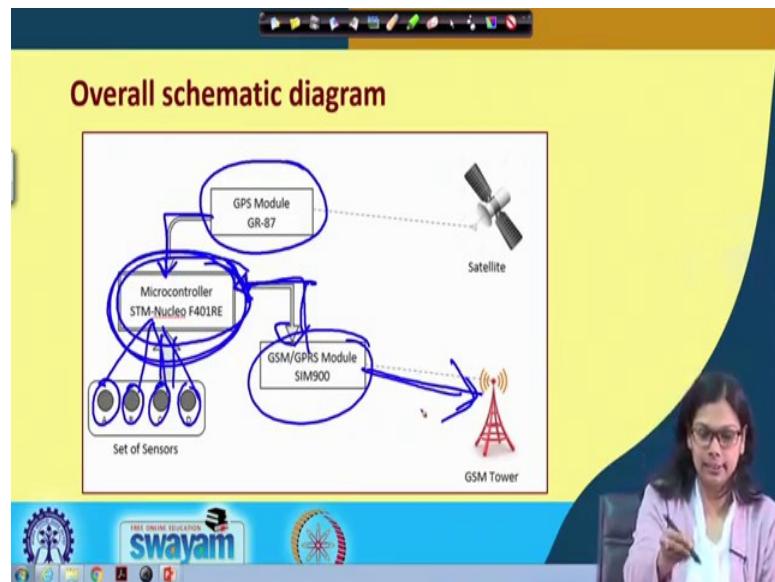
So, what is the requirement for building an object tracker? I have given you the motivation. You can build it very easily, but there are certain requirements. You need to have some devices that will track the location and will also receive some data, if it is required some sensor data can be uploaded.

Let us see what all devices are required for the processing; microcontroller is required for carrying out some local processing, and interface with the various sensors and the communication subsystem. The location has to be tracked, so there must be a code written in that microcontroller that will receive the location.

Let us say I want to track it every 5 minutes. So, you can write the code accordingly. Every 5 minute, it will read from the sensors, and it will upload to the server. So, we need a microcontroller on a first step.

Then for the communication capability, you will need a GSM module. We will discuss about this in course of time. And for location sensing, we need a GPS module, global positioning system. And for parameter sensing, you need various sensors like temperature sensor or humidity sensor or pressure sensor. So, these are the four things that are required basically to build the object tracking system.

(Refer Slide Time: 21:53)



This is the overall diagram. If you see this is the microcontroller, this is the GPS module, which will be connected with the microcontroller that will sense the coordinates. There might be some sensors that are also attached, which will be read by the microcontroller.

And as I said every 5 minutes, these data will be sent through this microcontroller using a GSM module to some server. This is basically the overall system, where along with processing capability, we have different sensors to read the data, and it will get stored here. But, here we also have some communication capability through this GSM module that is what we are sending the data to some server. Whatever data is there in the server at this point of time, that will give us the current location. And if you want to see the past location as well, you always can see that.

(Refer Slide Time: 23:19)

(a) The microcontroller

Single-chip computer system containing a processor core, memory, and programmable input/output subsystems.

- Responsible for interfacing with the sensors and location tracker, read the parameter data periodically.
- Responsible for interfacing with the communication subsystem, upload data periodically, and respond to incoming commands.
- Example: STM-Nucleo F401RE with ARM Cortex-M4 microprocessor, 96KB SRAM, 512KB flash storage, several I/O ports, etc.

The slide includes a photograph of a microcontroller board (STM-Nucleo F401RE) with various components and a breadboard. The background features a yellow and blue design with the 'swayam' logo.

So, this is the microcontroller I have already discussed, so I am not discussing in detail about this.

(Refer Slide Time: 23:27)

(b) The GPS module

Basically used for location sensing.

- The Global Positioning System (GPS) is a satellite-based navigation system made up of a network of satellites.
- It can be used anywhere in the world, 24 hours a day.
- Example: Holux GR-87 GPS module, which supports the standard NMEA-0183 format.

The slide includes a photograph of a Holux GR-87 GPS module, a diagram showing satellites orbiting Earth and their signals reaching a receiver, and a photograph of a person speaking. The background features a yellow and blue design with the 'swayam' logo.

Regarding GPS module, there are various GPS modules available. This is one example GPS module that is HOLUX GR-87 GPS module, you can use any other GPS module that is available.

(Refer Slide Time: 23:39)

(c) The GSM module

Basically used for communication with a remote location.

- Uses the same technology as used by the mobile phone networks for communication.
- It can be used provided there is a mobile tower in the range.
- Example: SIM900A GSM module, which supports the standard AT command set. It is a tri-band GSM/GPRS engine that works on frequencies 900 MHz, 1800 MHz, and 1900 MHz.

And then the GSM module; in this course, we have used the SIM900A GSM module, which I will be discussing in detail. It is basically used for communication with the remote location, and it uses the same technology as used by the mobile phones.

Whenever we use in a mobile phone, there is a SIM slot where you put a SIM card, and this is the chip of the SIM900A. This is a development board, you can see this particular chip in any development board. Here is the SIM slot, where you will put the SIM card. And of course, you have this Vcc, ground, and other pins to be connected along with transmitter and receiver.

The same module we have used for our experimentation, which also supports standard AT command set. We have to use standard AT commands for communication with this GSM module. It is a tri-band GSM, GPRS engine that works on these three frequencies, that is 900 megahertz, 1800 megahertz, and 1900 megahertz.

(Refer Slide Time: 25:29)

(d) Sensors

Depends on the application:

- Temperature sensor
- Humidity sensor
- Pressure sensor
- Vibration sensor
- Gas sensor

For communication, other methods like Bluetooth, radio communication, near field communication, etc. can also be used.

And you can have variety of sensors for integrating along with it. So, for communication we can also use Bluetooth, radio communication or near field communication.

So, we have come to the end of this lecture, where I have given you an idea of how a simple IoT system could be built along with its processing capability, along with sensor data, and I have discussed in detail about one of the applications that is object tracking. We have seen that the building the whole system is fairly very straightforward, but it has got such a variety of application, it could be used in so many places.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 35
GSM and Bluetooth

(Refer Slide Time: 00:31)



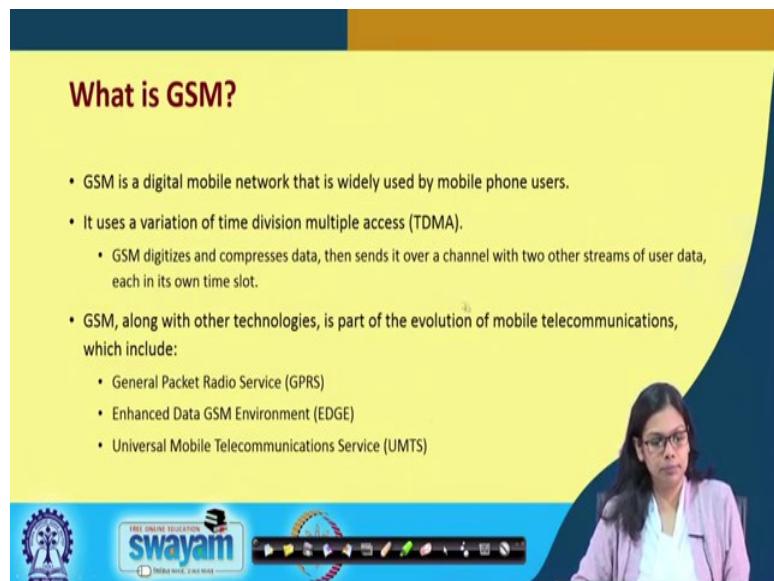
Welcome to lecture-35. In this lecture, I will be discussing about two communication protocols, one is GSM and another is Bluetooth.

(Refer Slide Time: 00:45)



GSM stands for Global System for Mobile Communication.

(Refer Slide Time: 00:53)



What is GSM?

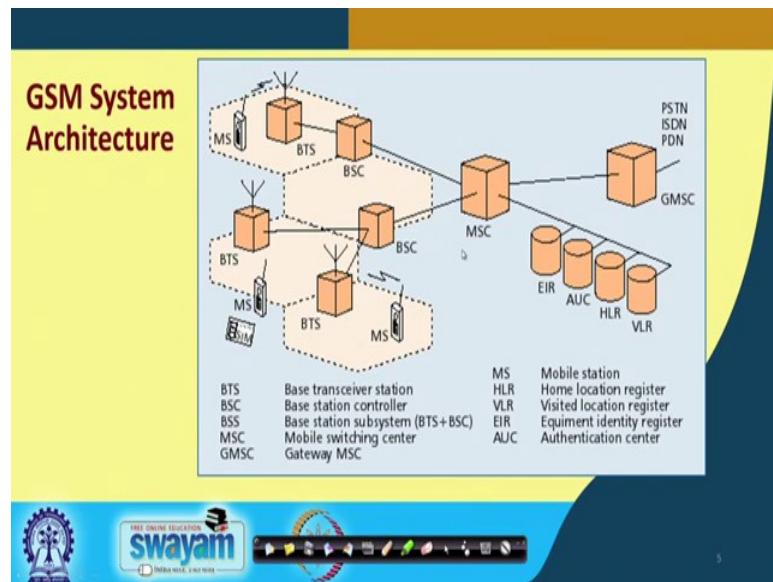
- GSM is a digital mobile network that is widely used by mobile phone users.
- It uses a variation of time division multiple access (TDMA).
 - GSM digitizes and compresses data, then sends it over a channel with two other streams of user data, each in its own time slot.
- GSM, along with other technologies, is part of the evolution of mobile telecommunications, which include:
 - General Packet Radio Service (GPRS)
 - Enhanced Data GSM Environment (EDGE)
 - Universal Mobile Telecommunications Service (UMTS)

GSM is a digital mobile network that is widely used by mobile phone users. It uses a variation of Time Division Multiple Access that is TDMA. GSM digitizes and compresses data, then it sends it over a channel with two other streams of user data, each in its own time slot. So, what does it mean? It means that it digitizes the data, compresses it, and sends through a channel where two other data are also moving. So, they are sharing that channel using time division multiple access.

GSM, along with other technologies, is part of the evolution of mobile telecommunication, which include many other protocols as well, like General Packet Radio Service that (GPRS) for sending packet data, Enhanced Data GSM Environment (EDGE), and Universal Mobile Telecommunication Service (UMTS).

In our case, we have used GSM for transmitting SMS from the microcontroller to any other mobile device, and to receive the SMS from any other mobile device to the GSM that is connected with the microcontroller. So, these are the two things that we have shown in the experiment, but other things can also be done. You can send a packet through GPRS, so accordingly you have to make your GSM work upon. You have to use the particular command to make or send the data through GPRS.

(Refer Slide Time: 03:11)



This is basically the GSM system architecture. This MS is the Mobile Station, and it consists of two components. One is the mobile equipment, and another is Subscriber Identity Module or SIM. BTS stands for Base Transceiver Station, you see this is one person. So, this is one cell you have one BTS in this cell, you have another cell where you have one more BTS. And this is used and consists of high-speed transmitter and receiver. And it provides two channels, one for signaling for transfer of signaling data, and other for data channel.

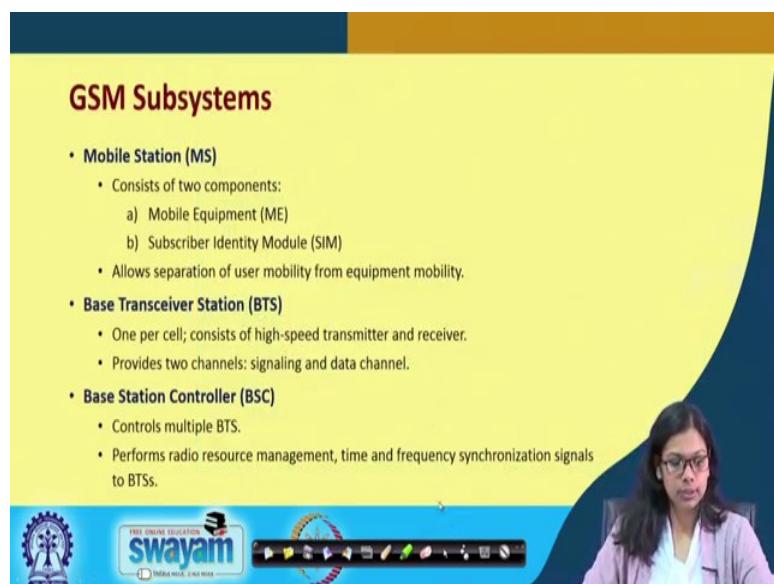
Then we have Base Station Controller or BSC. And you see that this base is coming connected with two BTS. So, this base station controller controls multiple BTS, you have one BTS here, one BTS here, so which is controlled by this BSC you can see. And it also performs radio resource management, time and frequency synchronization signals to BTS. So, we have mobile stations, each of these has got one BTS in this cell. And a number of BTS, which are in different cells, are connected with BSC or Base Station Controller.

Next we see this MSC or Mobile Switching Center. The switching node of a Public Land Mobile Network (PLMN) manages allocation of radio resources like handoff, and mobility of subscribers like location, registration etc. And there can be several MS's in this PLMN. You can have multiple such MSC's here in the public land mobile network.

Then you have one GMSC, which is Gateway MSC. You this MSC is connected with the gateway. So, you have this gateway MSC, which connects mobile network to a fixed network. And we often say that we are at home or we are roaming, for this HLR or Home Location Register, and another is VLR or Visitor Location Register are used. These are the two important aspect of this GSM.

And this MSC exchange information with this HLR. And then this VLR contains temporary information, when you move let us say from location A, which is your home location to location B, which is the visitor location. There we have this visitor register. It contains temporary information about the mobile subscriber that are currently located in that MSC service area, but whose HLR are elsewhere, but it is now a visitor for the other location.

(Refer Slide Time: 08:03)



GSM Subsystems

- **Mobile Station (MS)**
 - Consists of two components:
 - a) Mobile Equipment (ME)
 - b) Subscriber Identity Module (SIM)
 - Allows separation of user mobility from equipment mobility.
- **Base Transceiver Station (BTS)**
 - One per cell; consists of high-speed transmitter and receiver.
 - Provides two channels: signaling and data channel.
- **Base Station Controller (BSC)**
 - Controls multiple BTSs.
 - Performs radio resource management, time and frequency synchronization signals to BTSs.

FREE ONLINE EDUCATION
swayam
SWAYAM

So, this is exactly what I have already discussed about Mobile Station, Base Transceiver Station that is the BTS, Base Station Controller.

(Refer Slide Time: 08:13)

The slide has a yellow header and a blue footer. The footer contains the logo of the Indian Space Research Organisation (ISRO) and the text 'FREE ONLINE EDUCATION SWAYAM'. The slide content is as follows:

- **Mobile Switching Center (MSC)**
 - Switching node of a Public Land Mobile Network (PLMN).
 - Manages allocation of radio resources (handoff), and mobility of subscribers (location registration).
 - There can be several MSCs in a PLMN.
- **Gateway MSC (GMSC)**
 - Connects mobile network to a fixed network.
- **Home Location Register (HLR)**
 - For all registered users, it keeps user profile.
 - MSCs exchange information with HLR.
- **Visitor Location Register (VLR)**
 - Contains temporary information about mobile subscribers that are currently located in the area but whose HLR are elsewhere.

A woman with glasses and a pink jacket is visible on the right side of the slide, likely the speaker.

Mobile Switching Center, a gateway MSC that is Gateway Mobile Switching Center, Home Location Register, and Visitor Location Register, which I have already discussed.

(Refer Slide Time: 08:27)

The slide has a yellow header and a blue footer. The footer contains the logo of the Indian Space Research Organisation (ISRO) and the text 'FREE ONLINE EDUCATION SWAYAM'. The slide content is as follows:

GSM Identifiers

- International mobile subscriber identity (IMSI)
 - Unique 15 digits assigned by service provider, including home country code.
- International mobile station equipment identity (IMEI)
 - Unique 15 digits assigned by equipment manufacturer.
- Temporary mobile subscriber identity (TMSI)
 - 32-bit number assigned by VLR to uniquely identify a mobile station within a VLR's area.

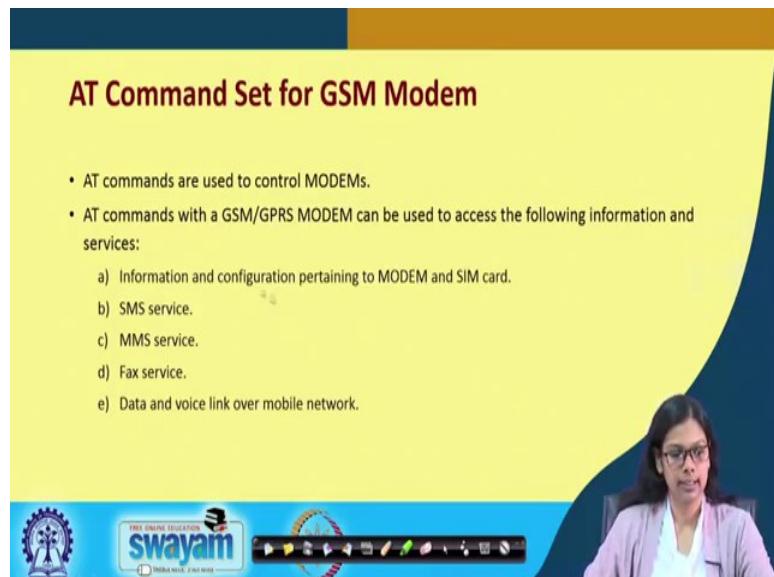
A woman with glasses and a pink jacket is visible on the right side of the slide, likely the speaker.

Now, this is an important thing what are the GSM identifiers. How a GSM is identified? It has got some unique identity. The first one is International Mobile Subscriber Identity, which is called IMSI. This is a unique 15 digit number assigned by the service provider. So, you are actually taking the service from some service provider, it is assigned by them

including home country code. When you move to some other country and you use it, then that unique country code will be attached to it.

The next one is International Mobile station Equipment Identity, or IMEI number. This is a unique 15 digit number assigned by the equipment manufacturer. When the manufacturer builds that particular device, it gives a unique number to it that is called IMEI number. Then there could be a Temporary Mobile Subscriber Identity, which is a 32-bit number that is assigned when you move to some other location by VLR to uniquely identify a mobile station within a VLR's region. So, this will also keep track of how many people from other region is moving here. They give this 32-bit number to get to those mobiles that are moving there.

(Refer Slide Time: 10:17)



AT Command Set for GSM Modem

- AT commands are used to control MODEMs.
- AT commands with a GSM/GPRS MODEM can be used to access the following information and services:
 - a) Information and configuration pertaining to MODEM and SIM card.
 - b) SMS service.
 - c) MMS service.
 - d) Fax service.
 - e) Data and voice link over mobile network.

Now, these AT commands are used for to set up the GSM module. The GSM modem that we will be using use AT commands, which we have also used in our code when we show you next. AT commands are used to control the MODEM's functionality. And these commands with GSM or GPRS modem can be used to access the following information and services. What are the following information, information and configuration pertaining to MODEM and SIM card what is the information, I mean what kind of SIM card we are using about other configuration regarding the MODEM etc.

SMS service, MMS service, fax service, and of course data and voice link over mobile network. So, it provides all these services. In our case, we have used SMS service; you

can also use GPRS service for sending packet as well, but in our case we will show you how we can use SMS service.

(Refer Slide Time: 11:29)

SIM900A GPS/GPRS MODEM

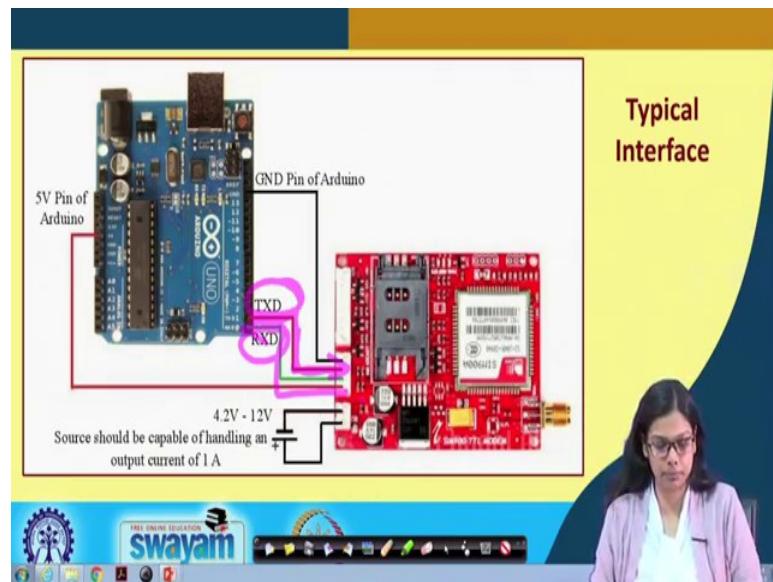
- The SIM900A is a quad-band GPS/GPRS engine, which works on frequencies 850 MHz, 900 MHz, 1800 MHz and 1900 MHz.
- Can be directly interfaced to the serial port of PC.
 - The baud rate can be configured from 9600-115200 through AT commands.
- Connections:
 - Digital I/O pins **RXD** and **TXD** connected to digital port lines.
 - An external power supply through adapter.
 - +5V and GND pins for other circuits.

SIM900A

This is SIM900A GPS, GPRS MODEM. So, you see that whatever is the chip this is for the chip area, and this is the SIM900. As I told you in the previous lecture, SIM900A is a quad-band GPS, GPRS engine which works on four frequencies, 850 megahertz, 900 megahertz, 1800 megahertz, and 1900 megahertz.

And it can be directly interfaced to the serial port of the PC. The baud rate can be configured from 9600 to 115200 through AT commands. The connection is fairly straightforward. The digital pins RXD and TXD, that is the receiver and transmitter, are connected to the digital port lines, we will see that how we have connected. An external power supply through an adapter is required. And of course, +5 volt and ground pins for other circuits is required.

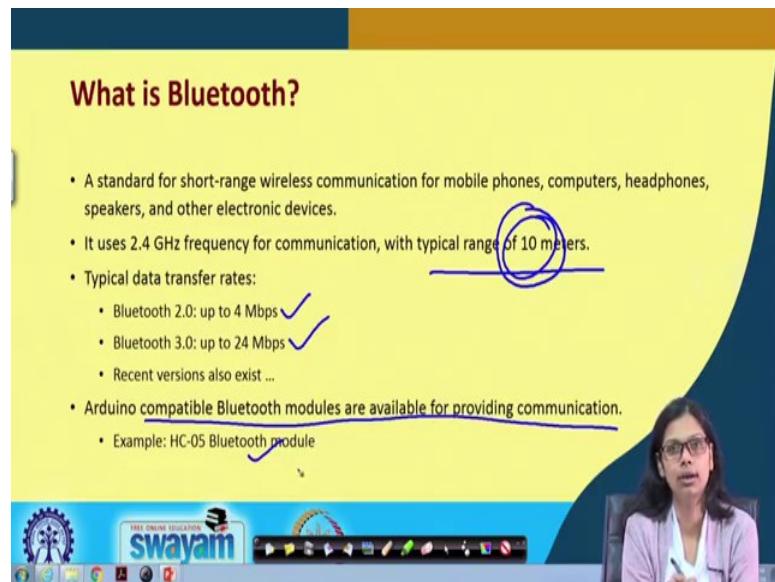
(Refer Slide Time: 13:03)



This is how the connection goes. This is the Arduino Uno board, this is the 5 volt pin, which is connected to the GSM 5 volt pin. And this is for the ground, which is connected to the ground of the GSM. And the receiver of this GSM must be connected to the transmitter of this pin, and the transmitter of this pin must be connected with the receiver pin of this Arduino board.

We must understand this particular thing. Whenever I say this will send some data, it will receive some data; this will send some data, and it will receive that data. So, one-way communication will be this to this, one-way communication will be this to this. So, accordingly the TX pin of this Arduino board must be connected to the RX pin of this SIM module. And the RX pin of Arduino must be connected with the TX pin of this particular model. So, we need to make sure that we are done with this proper connection.

(Refer Slide Time: 15:07)



What is Bluetooth?

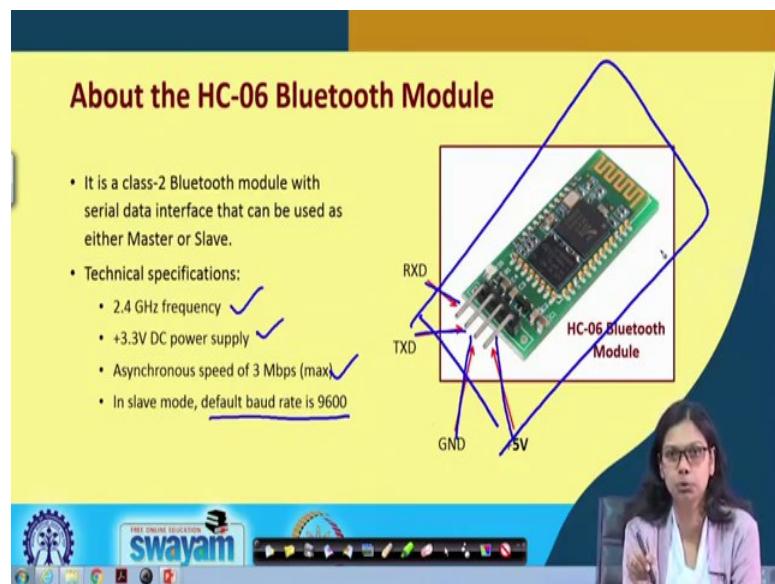
- A standard for short-range wireless communication for mobile phones, computers, headphones, speakers, and other electronic devices.
- It uses 2.4 GHz frequency for communication, with typical range of 10 meters.
- Typical data transfer rates:
 - Bluetooth 2.0: up to 4 Mbps ✓
 - Bluetooth 3.0: up to 24 Mbps ✓
 - Recent versions also exist ...
- Arduino compatible Bluetooth modules are available for providing communication.
 - Example: HC-05 Bluetooth module

Next I will talk about Bluetooth; we all might have used Bluetooth in our mobile phones for transferring photos or any other data. Let me very briefly talk about Bluetooth. This is a standard for short-range wireless communication for mobile phones, computers, headphones, speakers, and other electronic devices. These days you must have seen that you have wireless mouse and wireless keyboard. The communication between the PC and the device is done through Bluetooth communication that is how they communicate with each other.

It uses 2.4 GHz frequency for communication. And the typical range is 10 meter. So, the range for communication is not large. In previous GSM module, we were using a SIM module that was sending the data through SMS. If internet connection is there, you can send the message to any place, any mobile number and range is not limited. But, here you see that it is limited to 10 meters only.

The typical data transfer rates for Bluetooth 2.0 is up to 4 Mbps. And for Bluetooth 3.0, it is 24 Mbps, there are some recent version also. Arduino compatible Bluetooth modules are also there, and of course STM compatible. Bluetooth module are compatible with any board, you must connect it in the correct fashion, and you must use the correct command for using it.

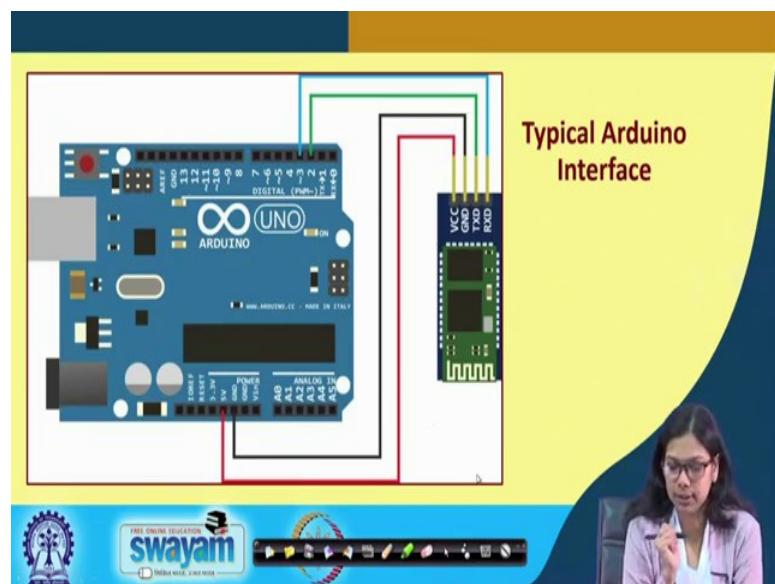
(Refer Slide Time: 17:35)



Let us see this HC-06 Bluetooth module. This is the Bluetooth module that we have also used it in our experiment ok. It has got this is 5 volt, ground, and this is the TX and this is the RX. It is a class-2 Bluetooth module with serial data interface that can be used as either master or slave.

Basically, these are some technical specification, which I have already discussed, like 2.4 GHz frequency, +3.3 volt DC power supply, maximum speed of around 3 Mbps, in sleep mode the default baud rate is 9600.

(Refer Slide Time: 18:49)



So, this is the typical Arduino interface if you see, where Vcc is connected to 5 volt, ground is connected, TX is connected with pin-2, which will be the RX. And RX is connected with pin-3, which was be TX.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science & Engineering
National Institute of Technology, Meghalaya

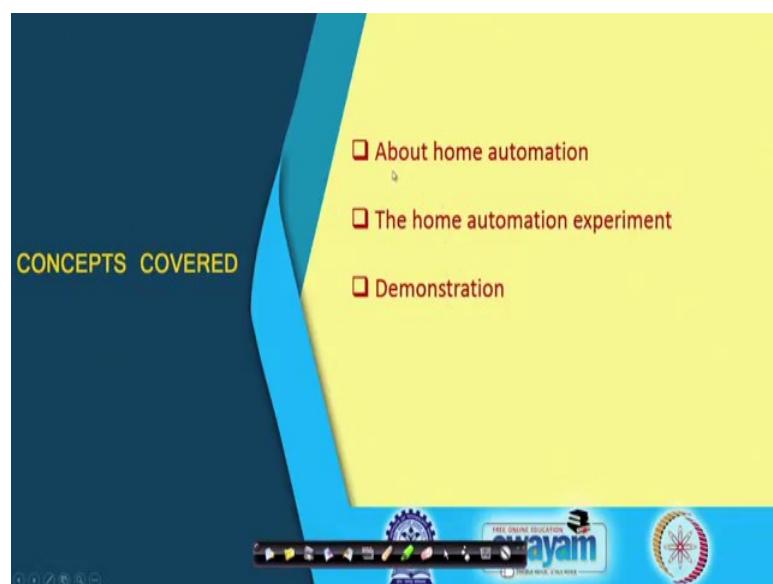
Lecture – 36
Design of Home Automation System

Welcome to lecture-36. In this lecture, I will be discussing about the design of a Home Automation System. The example that I will be taking here is a bulb, and that bulb I will switch on by sending an SMS.

Of course, we can do this using program control, but here we have done through SMS control such that we can give you a flavor of this IoT from anywhere in the world. If you can send an SMS “ON” to this particular device, then the device will be on. And if you are able to send “OFF”, it will be switched off.

First of all, I will be discussing two sets of code. One set where I can send SMS from any mobile, and it will work. And the other one is I will restrict sending SMS from some particular mobile number, because you will not want anyone to switch on of your home bulb. So, you want it to be done only by your family members. So, you can actually integrate those numbers of your family member in a secure code, where only it will be on or off if the SMS is received from a particular mobile number. So, I will be discussing these two things in this lecture.

(Refer Slide Time: 02:33)



I will talk about the home automation, the experiment, and then I will demonstrate.

(Refer Slide Time: 02:41)

About Home Automation

- Home automation or smart home is about building automation for a home.
 - Such a system can control lighting, temperature and humidity, entertainment system and appliances.
 - It can also involve home security like access control and alarm systems.
- Typically connects controlled systems through a suitable user interface.
 - Wall mounted terminals, mobile phone application, over the Internet, etc.
- When connected to the Internet, the home devices form an important constituent of the *Internet of Things* (IoTs).

FREE ONLINE EDUCATION
swayam
India Niye, Jee Niye

Home automation or smart home is about building automation for a home. Of course, this is not only switching on and off bulb, there could be many more things that could be done. Such a system can control lighting, temperature, humidity, and appliances. There could be many appliances in your home, which you can actually control. It can also involve home security like access control and alarm system as well.

Typically, it connects a controlled system through a suitable user interface. You can use a wall mounted terminal or a mobile phone application, over the Internet. When connected to Internet, the home devices form an important constituent of the so called internet of things. I have told you, the home automation that I will be discussing here is that I will be switching on and off a bulb through SMS control.

(Refer Slide Time: 04:09)

The Experiment

- An automated system in which an electric appliance can be turned on or off remotely by sending a SMS from a mobile phone.
 - A lamp is used for demonstration purpose.
 - The lamp is turned on when the message ON is sent through SMS.
 - The lamp is turned off when the message OFF is sent through SMS.
- Two versions:
 - A normal version where the message can come from any mobile phone.
 - A secured version where the message must come from a particular mobile phone.



The experiment goes like this; an automated system in which an electric appliance can be turned on or off remotely by sending a SMS from a mobile phone. A lamp (bulb) is used in this case for demonstration purpose. The lamp is turned on, when the message “ON” is send though SMS. And the lamp is turned off, when the message “OFF” is sent through SMS. And as I told you, there are two versions. One is a normal version, where the message can come from any mobile phone. And a secured version of course, where the message must come from a particular mobile phone.

(Refer Slide Time: 05:01)

Accessories Used in the Experiment

- Microcontroller (Arduino)
 - As the home controller.
- GSM Modem (SIM900A)
 - Used to communicate with the microcontroller from a mobile phone using SMS.
- Relay
 - Actuator device connected to microcontroller to turn ON/OFF an appliance.
- 15W Bulb
 - A home appliance used for demonstration.



What are the hardware components that we require for this experiment? We of course require a microcontroller; in this case we have used the Arduino UNO. A GSM modem is required; we have used SIM900A GSM modem, which is used to communicate with the microcontroller from a mobile phone using SMS.

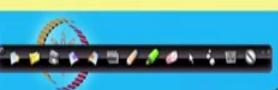
We need an electromechanical device here, that is relay, which is already discussed earlier. In this experiment it is acting as a actuated device, which is connected to microcontroller to turn ON and OFF the appliance. And we have used a 15 watt bulb for demonstration, you can use any other device.

(Refer Slide Time: 06:03)

Working of the System

- The system polls the GSM modem and waits for arrival of message from the user.
- The microcontroller reads the message, and compares it with the condition for controlling the equipment (viz. the lamp).
- The relay is activated to turn on the lamp when it receives the message ON.
- The relay is activated to turn off the lamp when it receives the message OFF.
- The process continues.

• For the second version, an additional check is made for the mobile number from where the message has been sent.

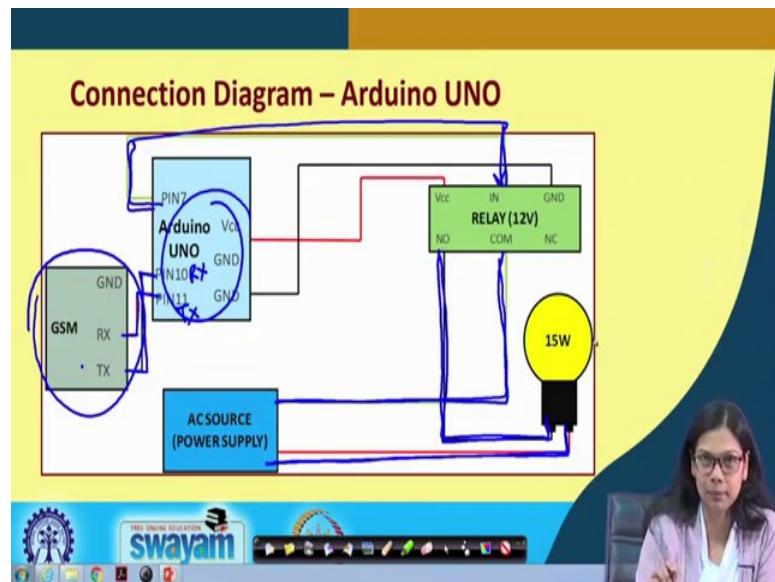
  



The working of the system goes like this. The system reads the GSM modem and waits for arrival of a message from the user. Then the microcontroller reads the message, compares it with the condition for controlling the equipment.

Accordingly, what it receives either ON or OFF. The relay is activated to turn on the lamp, when it receives the ON message. And the relay is activated to turn off the lamp, when it receives the OFF message and the whole process continues. So, this is all about working of the system. And as I said for the second version, an additional check is made for the mobile number from where the message has been set.

(Refer Slide Time: 07:01)

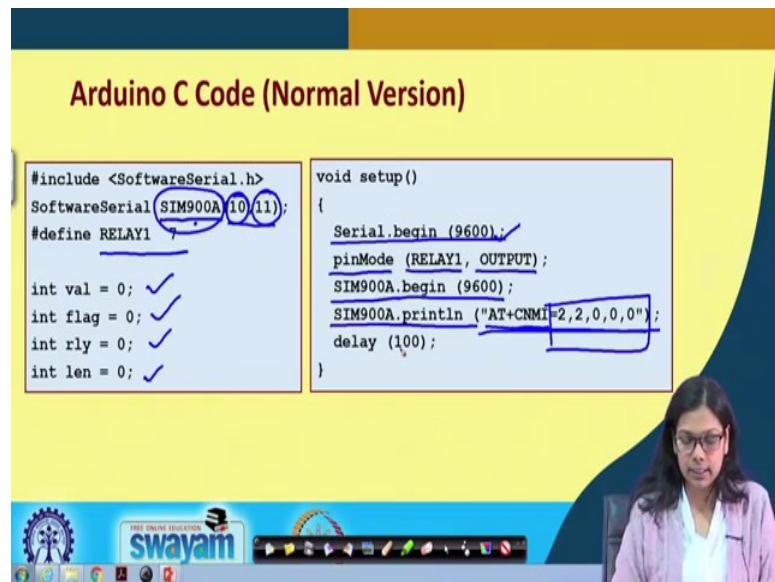


Let us look into the connection. This is the relay, and we know that for relay you have a Vcc, you have a ground, and a control input that is connected to pin-7 of Arduino.

And this is my bulb, and this is my AC source. One end of the bulb is directly connected to NO output pin of this relay. The other end of the bulb is connected to the AC source. And another end of the AC source is directly connected with the COM output pin of the relay. Pin 7 is connected to the input.

And with the GSM board, which will receive the SMS that is either ON or OFF, will be connected like this; RX pin of this is connected to pin number 11, which is the TX. And TX pin of this is connected to pin number 10, which is the RX. This is all about the connection, and of course ground and Vcc will be connected accordingly. GSM will receive an SMS, and depending on that it will do some processing, and send a value through this pin-7 to the relay. And according the relay will make this bulb glow or it will make this bulb off depending on the condition that we have given.

(Refer Slide Time: 09:43)



Arduino C Code (Normal Version)

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900A(10,11);
#define RELAY1 7
int val = 0; ✓
int flag = 0; ✓
int rly = 0; ✓
int len = 0; ✓

void setup()
{
  Serial.begin (9600);
  pinMode (RELAY1, OUTPUT);
  SIM900A.begin (9600);
  SIM900A.println ("AT+CNMI=2,2,0,0,0");
  delay (100);
}
```

Let us now see, how the code goes. So, you have to include this SoftwareSerial, you have to make this SIM900A this RX, TX pin which is 10 and 11. And we define this RELAY1 as 7. Then we define some variables val, flag, relay, len, we have initialized all these to 0. We will see one by one why we require these variables.

In the setup phase recall that for serial communication, we do not require explicitly any hyper terminal for Arduino, but for STM we require that. For Arduino it can be directly printed in the serial monitor.

So, in the setup phase we first define the baud rate for that serial printing that is Serial.begin(9600). pinMode sets RELAY1 as an output port. We are setting up the baud rate as 9600 again and one of the important thing that we are doing, now once we have made this particular object SIM900A.begin(9600).

We have to also specify that the GSM module that we are using, we will be using it for messaging purpose, so that particular thing we have to define it. For that particular reason, we need to use an AT command, which goes like as shown.

CNMI stand for Command Name Message Indication. It means we will be able to receive message here. And then we give a delay of 100 millisecond.

(Refer Slide Time: 12:57)

```
void loop()
{
    if (val == 0) ✓
    {
        off(); ✓
        val = 1; ✓
    }
    while(1)
    {
        readsms();
    }
}

void on()
{
    digitalWrite(RELAY1, 0); // Turn Relay ON
    Serial.println("Light ON");
    rly = 1;
}

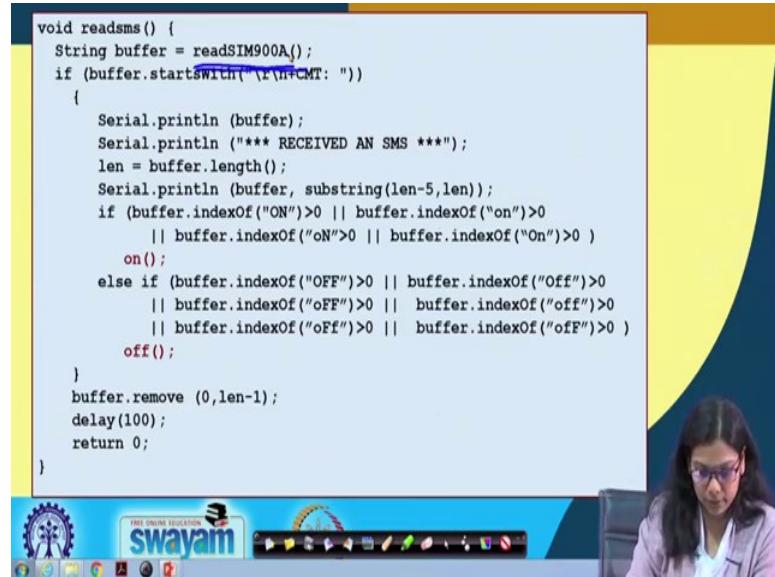
void off()
{
    digitalWrite(RELAY1, 1); // Turn Relay OFF
    Serial.println("Light OFF");
    rly = 0;
}
```

This is the set up phase. Now, in the loop phase, what we are doing. Initially, we are checking for the variable “val”. The value for “val” is 0 now. I make it off, and then I make “val” equals to 1. So, it will not go to this loop again, because “val” has now become 1. And then in the while(1) loop, I am reading SMS.

So, I will see what this readsms() function does. Let me tell you what this ON and OFF function will do; basically the ON and OFF function will make the relay ON or OFF. If you make relay ON or OFF, the bulb will be made ON and OFF. In the ON function, we do a digitalWrite (RELAY1,0) to turn the relay ON. And this Serial.println, it will print “Light ON” in the serial monitor.

And then we are making “rly” as 1. And in the off function, we are doing a digitalWrite (RELAY1, 1) that will turn off the relay. And it will print in the serial monitor “Light OFF”. And this will also make “rly” equals to 0.

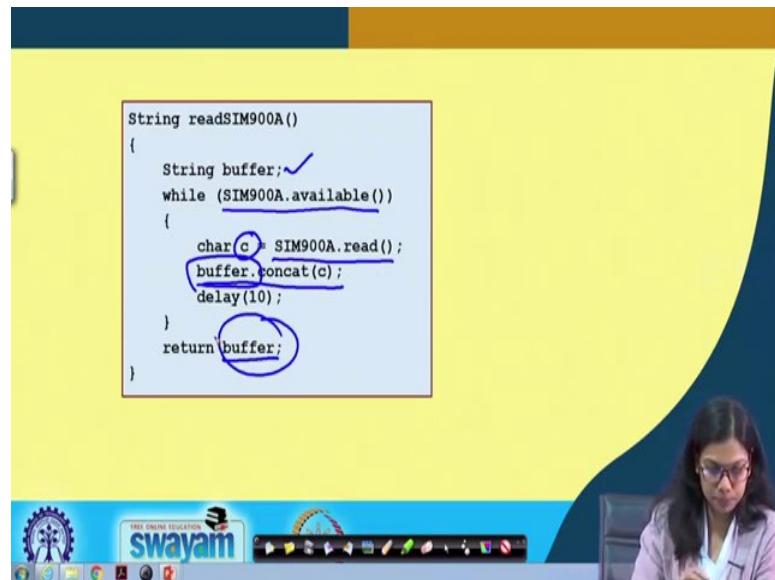
(Refer Slide Time: 15:45)



```
void readsms() {
    String buffer = readSIM900A();
    if (buffer.startsWith("R\n+CMT: "))
    {
        Serial.println (buffer);
        Serial.println ("*** RECEIVED AN SMS ***");
        len = buffer.length();
        Serial.println (buffer, substring(len-5,len));
        if (buffer.indexOf("ON")>0 || buffer.indexOf("on")>0
            || buffer.indexOf("oN")>0 || buffer.indexOf("On")>0 )
            on();
        else if (buffer.indexOf("OFF")>0 || buffer.indexOf("Off")>0
            || buffer.indexOf("oFF")>0 || buffer.indexOf("off")>0
            || buffer.indexOf("oFf")>0 || buffer.indexOf("ofF")>0 )
            off();
    }
    buffer.remove (0,len-1);
    delay(100);
    return 0;
}
```

Next the function readsms. There is one more function that we have written, readSIM900A(), let us understand this carefully. What we are doing in readsms, we are continuously running this in that loop.

(Refer Slide Time: 16:31)



```
String readSIM900A()
{
    String buffer;
    while (SIM900A.available())
    {
        char c = SIM900A.read();
        buffer.concat(c);
        delay(10);
    }
    return buffer;
}
```

Let us go to readsim900A(), what it is doing it is defining a string called buffer. And then it is checking in this while condition, whether the connection is available or not.

If it is available, it is reading SIM900A.read using this function and it is storing it in c. And this buffer.concat, we are taking each of this character one by one reading it, and we

are concatenating in this buffer.concat(c). And we have given a small delay that is 10 milliseconds. And finally, we are returning this buffer.

Now, we will go to the previous slide that is readsms. We are calling that function which I have shown just now, and we are storing it in buffer. And then we are checking if this buffer string starts with this or not.

Now, again we are printing, whatever we have received. And we see that once this is printed, SMS is received, we display this in the serial. And these things are only written for our checks, you can remove it in the final program if you want. Then we calculate the length of this buffer. After calculating the length, we take the substring, where we cut out that length minus 5, and we take that particular string.

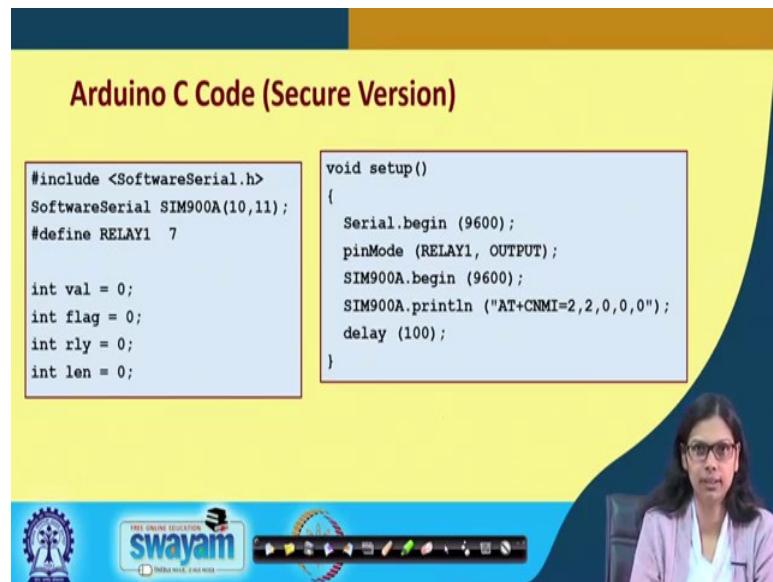
And now what we do, once I have the SMS with me, there are certain other characters also, that comes in. We are removing that part, and we are now concerned about the exact SMS. Now, if buffer.indexOf is ON, indexOf will search for this string from this buffer, and it will return that particular location where it has found out. So, in that case, it will always return a value greater than 0. And if it does not find that, it will return something less than 0, that is, minus 1.

So, in our case we will receive something greater than 0, so that is why what we are checking here buffer.indexOf is ON. It could be combination small and big, and then big and small. So, all the possible ways we have taken into consideration.

If the message contains any of these things, then it will call the on function, and it will switch on the bulb. Similarly, we check for OFF conditions, there are various possibilities. So, we receive an SMS, we cut out some portion, and then we see the exact SMS with us.

And in that particular SMS, I am checking for whether it has got “ON” in that message or not. And finally, from this buffer we are removing the entire string, we are giving a 100 milliseconds delay, and we return 0. This goes on repeatedly. This is a simple code; from any mobile number if the SMS comes, then it will do like this.

(Refer Slide Time: 21:59)



Arduino C Code (Secure Version)

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900A(10,11);
#define RELAY1 7

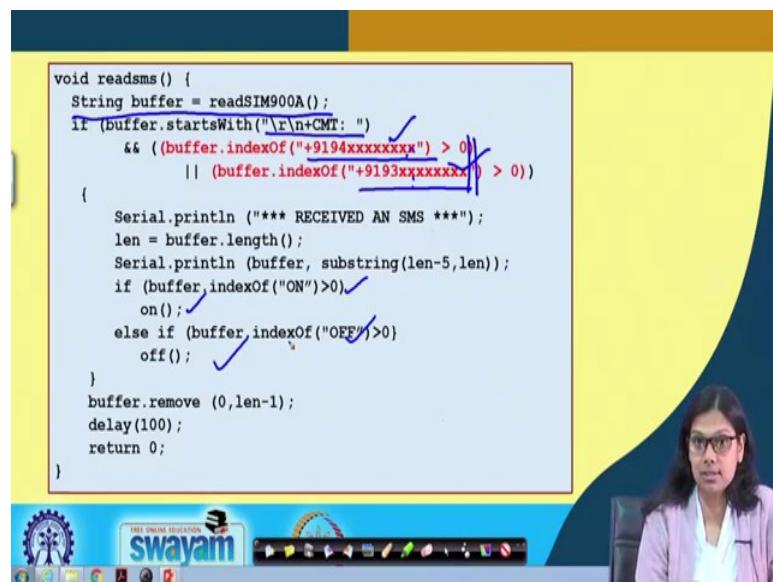
int val = 0;
int flag = 0;
int rly = 0;
int len = 0;

void setup()
{
  Serial.begin (9600);
  pinMode (RELAY1, OUTPUT);
  SIM900A.begin (9600);
  SIM900A.println ("AT+CNMI=2,2,0,0,0");
  delay (100);
}
```

The slide is from the Swayam platform, as indicated by the logo at the bottom.

Now, we will see the next code, which is the secure version. Secure version means, I have already told you, it must receive from some particular number. And then only I will go for it, I will make it on or off. If I get it from any other mobile number, I will not make it on or off. So, this part of the code will be fairly the straightforward, and this part as well will be the same, where the difference starts is here.

(Refer Slide Time: 22:23)



```
void readsms() {
  String buffer = readSIM900A();
  if (buffer.startsWith("\r\n+CMT: "))
    if ((buffer.indexOf("+9194xxxxxxxx") > 0) || (buffer.indexOf("+9193xxxxxxxx") > 0))
    {
      Serial.println ("*** RECEIVED AN SMS ***");
      len = buffer.length();
      Serial.println (buffer, substring(len-5,len));
      if (buffer.indexOf("ON")>0)
        on();
      else if (buffer.indexOf("OFF")>0)
        off();
    }
  buffer.remove (0,len-1);
  delay(100);
  return 0;
}
```

The slide is from the Swayam platform, as indicated by the logo at the bottom.

We are checking if the string starts with this. And we are also checking buffer dot index. Here I have checked with these two numbers, you can do this for many numbers.

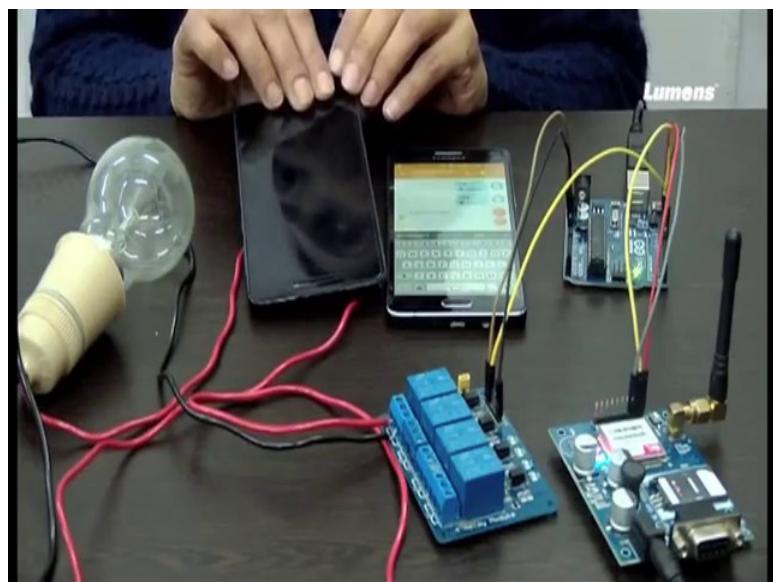
If it sends either from this number or from this number, then it should work. But, it should not work with any other numbers other than what is specified here. And then finally, you remove from the buffer. So, this is all about the two versions of the code for this making a bulb ON and OFF using SMS control.

Now, we will be demonstrating you the experiment that I have just now discussed.

So, now I will be showing you the experiments, where I will be switching ON and OFF a bulb using SMS control. I have already discussed the codes with you. I have already discussed how you will be making the circuit.

So, there are two variation of the code that we have done. One anybody can send, another you can put series of phone numbers from whom you want this particular switch ON and OFF to happen. So, these are the two things that I will be showing you. You already have come across how you will switch ON a bulb using relay just using program control. There we are not sending any SMS, rather we are just switching ON and OFF a bulb using relay control. You already know how it works. I will just integrate it along with the GSM module here.

(Refer Slide Time: 25:59)



So, this is the bulb. This bulb should be connected to this plug like this AC source, you must have a relay module with you, this is a four module relay, you can have a single module relay with you also. You should have a GSM module with you. This is SIM900

module, I will have already discussed in detail regarding this module. You need a SIM card for it, you can see the SIM card that is put in here. And you need a microcontroller board, which in this case is Arduino Uno.

Now, see how this connection goes. One point of the AC source will be directly connected to this bulb. And another point of the AC source, which is this one is going to be an input to the COM point of this relay. And another end of the bulb is going to this normally open point of the relay.

Now I will be showing you what connection you have to make with this GSM board, and with your microcontroller board. GSM board has got a ground. This ground is connected to the ground of Arduino. And this GSM board has got RX and TX pins.

The TX pin of the GSM must be connected with the RX pin of the Arduino. So, we have made two pins as RX and TX for Arduino, which is pin 10 and pin 11. So, the TX pin must be connected with the RX pin, here it is pin number 10. And the RX pin is connected with pin number 11 that is the TX pin of Arduino.

Now, finally I have to make a connection with the relay. Now, what is basically happening if you see, the bulb will glow when you send an SMS. So, you must have a connection between the GSM module and the Arduino board. The Arduino board must receive the SMS, depending on which it should control this relay to make it ON or OFF.

We have used this PIN7 of Arduino for connecting with the input of this relay through which we will be sending. And there is one GND, and one VCC. So, we are using this fourth input of this relay.

So, finally an SMS will be received through this GSM. And the microcontroller will instruct to do certain things, either ON or OFF. Now, I will be dumping the code one by one to show both the things.

How do I show you the secure one? I will be using these two mobile phones. First I will send SMS from this particular mobile phone, and you see that it will work.

Secondly, I will show a secure connection where I will first send SMS from this mobile phone, and you will see that it will not work. But, when I sent from this mobile phone, it will work. Let me show you. Let me dump the code.

Now, see I will be sending an ON message, it seems to have some issue. So, I will just ON it and I will just send. So, you can see that I have sent it from this mobile, so this is not the secure code basically. I will again send it OFF from my other mobile, which is this one let me do that. So, I will make it OFF ok. So, the bulb is switched off.

I will show once more. I am sending ON, it is working. And I will send again OFF, you see it is working, but this code is not the secure one. As you said from both the mobiles, it was working. Now, I will make sure that I will when I send it from this mobile, this will not run. And when I send with this particular mobile, it will only work.

Now I will try out in this mobile, I have already dumped the code. So, now I am sending ON message, let us wait for some time nothing happened right? Now, I will try with this mobile, and it worked. So, this is a secure communication, where you can restrict doing this kind of automatic ON, OFF switch through SMS control for certain phone numbers, and not for any phone numbers

So, today we discussed about a simple automation that we can do using relay, a GSM board, a microcontroller, and of course a bulb, it can be any electrical appliances, but here we have used a simple bulb to make it ON and OFF.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science & Engineering
National Institute of Technology, Meghalaya

Lecture – 37
Design of a Simple Alarm System Using Touch Sensor

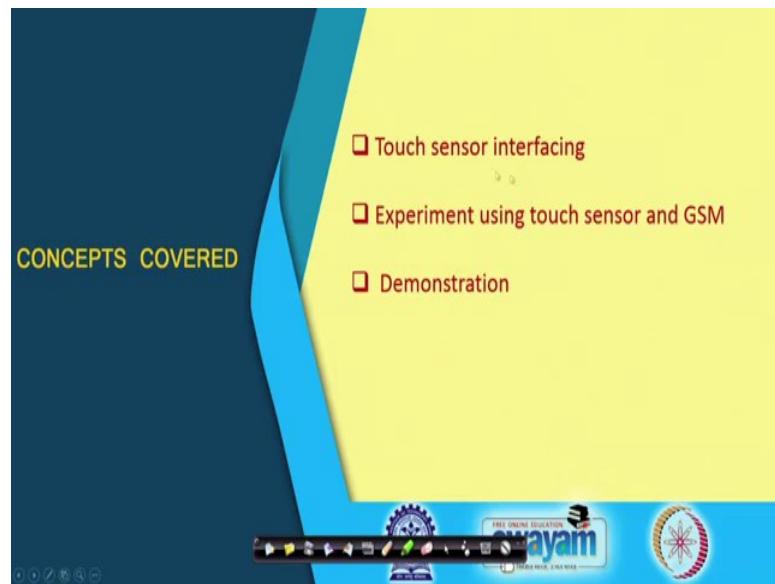
Welcome to lecture 37. In this lecture, I will be showing the design of a Simple Alarm System using Touch Sensor. Let us think of a scenario, where you are out of your house during summer vacation for a long time, of course securities are there in places, but still theft may occur. Any way if you know that there are certain points in your house through which a person can enter; the door there is a lock and someone has to break that lock and have to enter, or it can be from some windows anyway you have to break that window.

So, any vulnerable point in your house, you can consider that for using this particular device which is nothing but a touch sensor., You can put a touch sensor in those places and if somebody touches there, an alarm will go off.

So, accordingly you can have a system where you can put this touch sensor along with the application where whenever somebody touches the sensor an SMS alert will be sent to your mobile. And then you can make an app, where if an SMS is received from that particular number, an alarm system will get activated.

I will be discussing about this particular thing here, whenever an unwanted person touches your door when you are not there, an SMS alert will be sent. In the previous example we were sending an SMS from a mobile to your system that was making the bulb glow on and off. But here what we are doing? Here if the system analyzes that there is a theft attempt or something like that then the system will send a SMS to some particular number.

(Refer Slide Time: 03:55)



We will talk about the touch sensor. The sensor interface is straightforward. Finally, I will demonstrate.

(Refer Slide Time: 04:17)

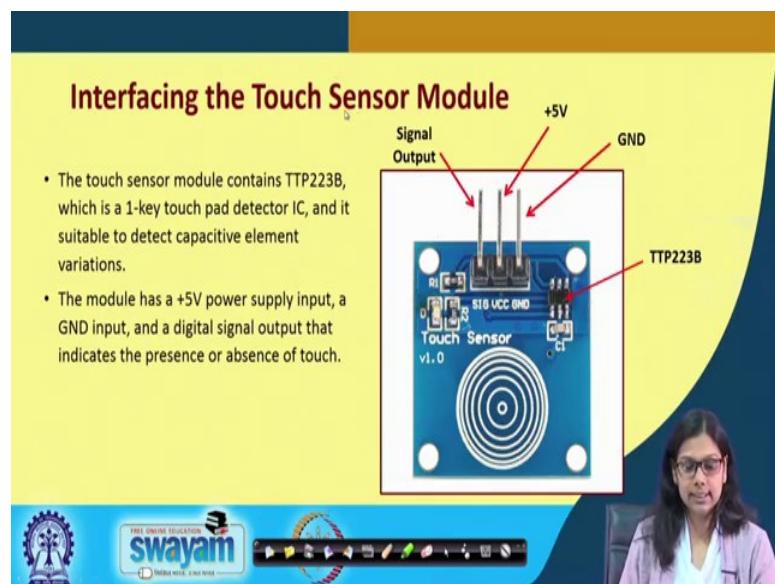
As I said, this experiment demonstrates another application of home automation system; it can be integrated for the security aspect. So, this is related to home security where an automated alarm generation system warns the owner of the house whenever an unauthorized access takes place.

Whenever an unauthorized access takes place, it will send SMS.

How is unauthorized access detected? Here the unauthorized access is detected through a touch interface circuit that is similar to a car intrusion alarm system, where somebody touches your car and an alarm will activate.

What happens after the detection? Often the alarm system can be activated and the owner can be warned by sending an SMS from his or her mobile phone.

(Refer Slide Time: 05:41)



This is the touch sensor that we have used. There is a Vcc, GND and this is the signal output that is coming here. And this is TTP223B; this is the area if you touch either this part or if you touch from below also it will work. The touch sensor module contains this TTP223B which is a 1-key touch pad detector IC and is suitable to detect capacitive element variations.

The module has a 5 volt power supply and a ground, and a digital signal output that indicates the presence or absence of the touch. The digital signal output indicates the presence of this or absence of this touch.

(Refer Slide Time: 06:45)

The Experiment

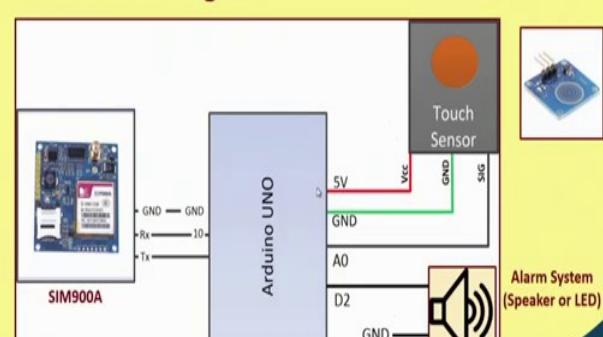
- An Arduino UNO microcontroller system connects to the touch sensor, and a GSM module (SIM900A).
 - The microcontroller continuously polls the touch sensor to find out whether an intrusion (i.e. a touch) has been detected.
 - If so, a suitable message is sent to the person concerned over SMS.
 - An alarm system is also activated.



The experiment we have done using the Arduino board where we connect this touch sensor, the GSM module and the microcontroller. The microcontroller continuously polls the touch sensor to find out whether there is an intrusion or not. And if so, a suitable message is sent to the concerned person over SMS, an alarm system is also activated. We have simulated the alarm system using the speaker that we have already discussed earlier; a siren like sound will come and we also sent an SMS.

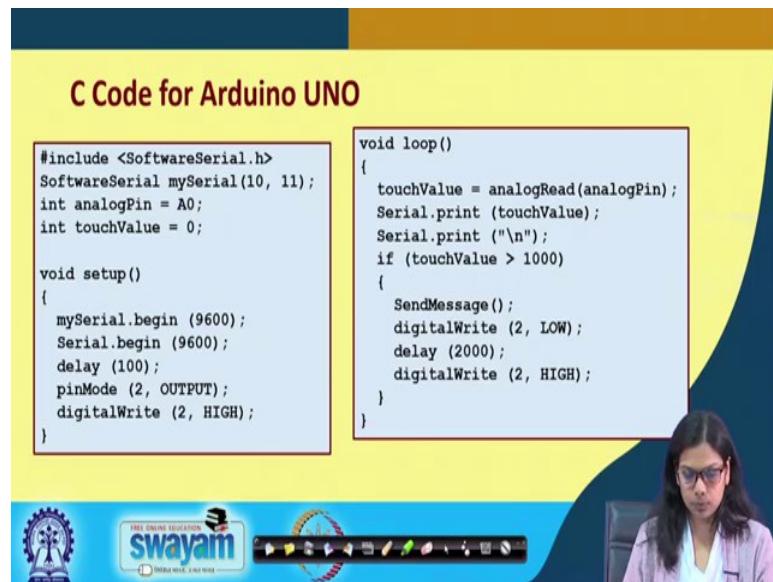
(Refer Slide Time: 07:37)

Connection Diagram



This is the circuit diagram. So, this is ground, this is Rx, this is Tx that are connected to Arduino pins 10 and 11, and we have connected the signal to A0 pin, and this Vcc and ground. This is the touch sensor and we have also connected an alarm system; to simulate the alarm system we have used a speaker, the speaker is connected to pin D2 and this is connected to ground.

(Refer Slide Time: 08:15)



```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11);
int analogPin = A0;
int touchValue = 0;

void setup()
{
    mySerial.begin (9600);
    Serial.begin (9600);
    delay (100);
    pinMode (2, OUTPUT);
    digitalWrite (2, HIGH);
}

void loop()
{
    touchValue = analogRead(analogPin);
    Serial.print (touchValue);
    Serial.print ("\n");
    if (touchValue > 1000)
    {
        SendMessage();
        digitalWrite (2, LOW);
        delay (2000);
        digitalWrite (2, HIGH);
    }
}
```

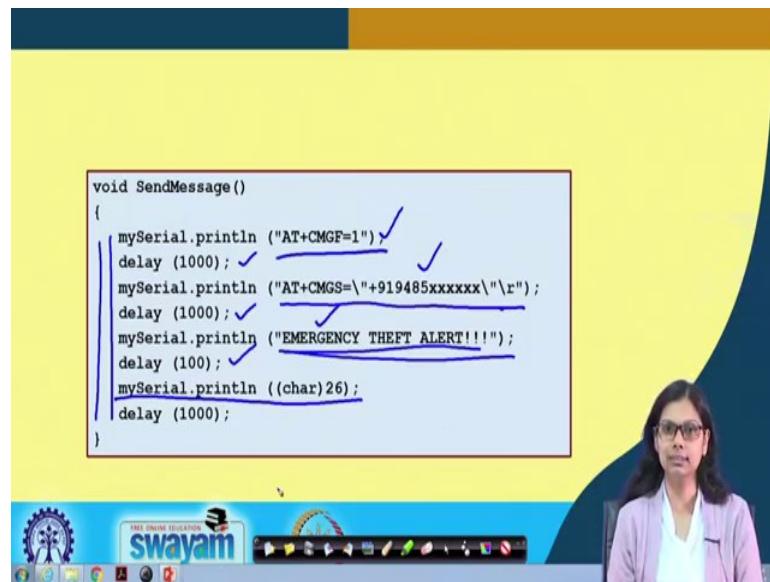
FREE ONLINE EDUCATION
swayam
INDIA WIDE, 24x7x365

Now, here goes the code for Arduino UNO, we already know how to make the software serial connection. We are making the serial connection with pin number 10 and 11, and the analog port we are using A0. Initially the touch value we are assigning it to 0. In the setup phase we use mySerial.begin for this particular serial communication with the GSM, and this is for serial communication with the screen. Then there is a delay, and we are making the pinMode 2 to output, we are making pin number 2 as an output port and we are writing high digital value here in port 2 in the setup phase.

Now in the loop phase what we have to do? We will read the value from the analog pin through which that signal is connected to the touch sensor, and then we are printing the touch value we are getting, you can see that in the serial monitor. Now if the touch value is greater than some value, that is 1000 here, then we send the message, and then we are writing digital LOW to pin 2, and we provide a delay of 2000 milliseconds or 2 seconds and then again we digital write pin 2 to LOW.

What does send SMS function do we will see in the next part.

(Refer Slide Time: 10:29)



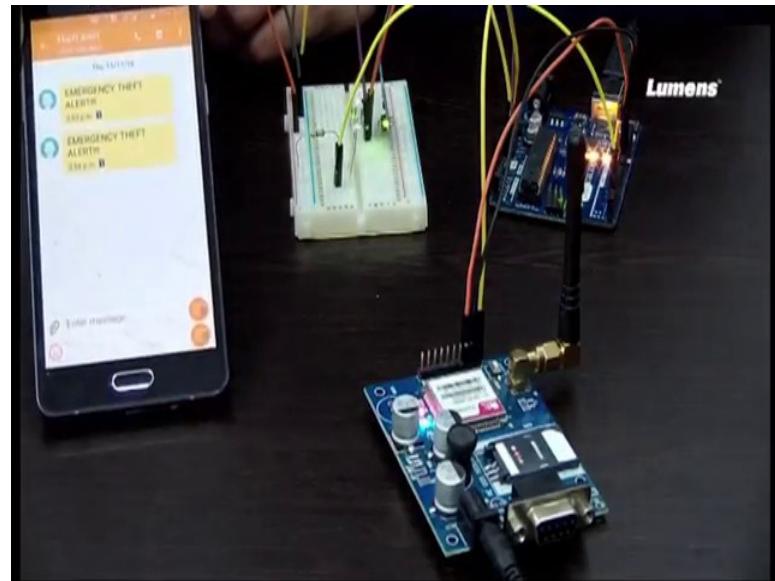
In the previous example what we did we received the SMS. We received the SMS, then we cut out some part and then we did the needful, now here we have to send the SMS to a particular number. In that case you have to use this particular AT command that is necessary, “AT+CMGF=1”.

Then we give a delay of 1 second and then again we are sending another AT command where we are providing the mobile number to which the SMS will be sent. We again give a delay of 100 millisecond and send the SMS. The SMS alert will be sent to your mobile, and again with a delay we are providing. The total characters if you count is 26.

In today's experiment we will be interfacing a touch sensor, I have already discussed about the feature of this touch sensor that I will be interfacing today.

Let us see how I can interface this sensor.

(Refer Slide Time: 14:41)



This is the touch sensor. Similar to other sensors, this also has got 3 pins, one is GND, next one is Vcc, and the next one is the output signal that is an analog signal.

The touch sensor works from both sides, if you touch from this side as well if you touch from this side. I will be first connecting this touch sensor. One I will be connecting to Vcc, one to GND, and one to the analog port that I will be using is A0.

Now as I said I have also connected an LED. This is the anode and from this point I will connect to digital pin 2. This is just for the checking purpose that the message has been sent or not.

For the GSM module, this is connected to GND and the other 2 pins, the first one is Tx that will be connected to pin number 10, and Rx is connected to pin number 11. You may recall that in that bulb on-off through SMS control, we were sending an SMS from our mobile to this GSM and then whatever we received based on that this microcontroller was operating.

But now it is just the opposite, when I touch this an SMS alert from this GSM will be sent to my mobile number. I will just dump the code first and this is the mobile where I will be receiving an SMS if I touch this touch sensor.

The code has been dumped. Now let us see ... I will touch this sensor. The green light is on meaning that the SMS has been sent from this particular GSM module through this

SIM to my mobile, and let me see what I receive; I have received an SMS. I have saved this GSM mobile number that is being displayed in this. I will do once more. So, the LED glows, and I will receive an SMS. You can see I have received another SMS.

The entire process is fairly straightforward, but the point is we can have variety of applications with this touch sensor.

Thank you.

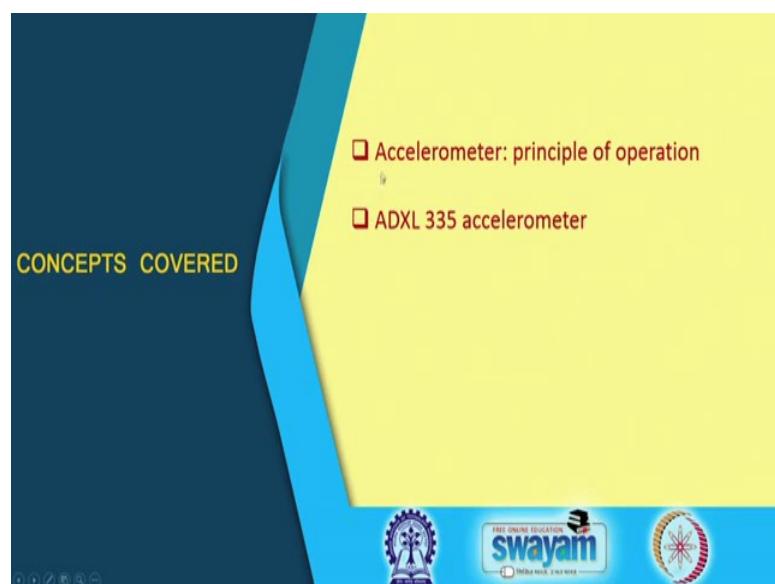
Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 38
Accelerometer

Welcome to week-8. This is the final week for the course Embedded System Design with ARM. In this week, we will be discussing about Accelerometer and some of the experiments that we can do with this device. So, in the first lecture, I will be discussing about accelerometer what it is and what is the principle operation.

Then in the subsequent next two lectures, I will first discuss that how we can interface an accelerometer with STM board. And how we can receive the data, this is the first thing that I will show. And then I will show you some experiment where the orientation of the device we will find out, whether the device is lying flat or it is vertically up or it is vertically down etc. So, these are few things we will be looking into with respect to accelerometer. And there is one more thing that we will look into in this week is an experiment with a gas sensor.

(Refer Slide Time: 01:50)



Let me talk about accelerometer. In this particular lecture, I will be discussing the principle of operation of accelerometer. And the accelerometer that I will be discussing is ADXL 335.

(Refer Slide Time: 02:11)

What is an accelerometer?

- A dynamic sensor that can measure acceleration in one, two, or three orthogonal axes.
- Typically used in one of three modes:
 - As an inertial measurement of velocity and position.
 - As a sensor of inclination, tilt or orientation in 2/3 dimensions.
 - As a vibration or impact (shock) sensor.
- Most accelerometers are based on Micro Electro-Mechanical Sensors (MEMS).
 - Based on the displacement of a small mass etched into the silicon surface of the IC, and suspended by small beams.
 - As an acceleration is applied, a force develops ($P = m f$) that displaces the mass.





 FREE ONLINE EDUCATION
swayam
India Meets Knowledge

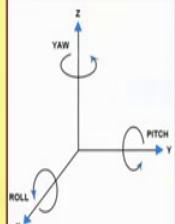


So, what is an accelerometer? It is a dynamic sensor that can measure acceleration in one, two, or three orthogonal axes X, Y and Z. And typically it is used in one of three modes. It can be used as an inertial measurement of velocity and position, it can be also used as a sensor of inclination, tilt, or orientation in 2 to 3 dimensions, or it can also be used as a vibration or impact sensor, or shock sensor.

Most of the accelerometers that are developed are based on Micro Electro Mechanical Sensors (MEMS). And this is based on displacement of a small mass that is etched into the silicon surface of the IC, and suspended by small beams. We will look further into this when we discuss the working principle. So, as an acceleration is applied, a force is developed. We know that $P = m \times f$.

(Refer Slide Time: 04:01)

- The displacement of the mass can be measured using capacitive sensing or piezoelectric effect sensing.
 - Change in capacitance, or generation of a voltage.
- The basic structure of an accelerometer consists of fixed plates and moving plates (called *mass*).
 - Acceleration deflects the moving mass and unbalances the differential capacitor that results in a sensor output voltage amplitude which is proportional to the acceleration.
 - By measuring the acceleration along the x, y and z directions, it is possible to calculate the inclination or tilt.
 - It is also possible to calculate the angles of rotation along x, y and z axes (called *roll*, *pitch* and *yaw* respectively).

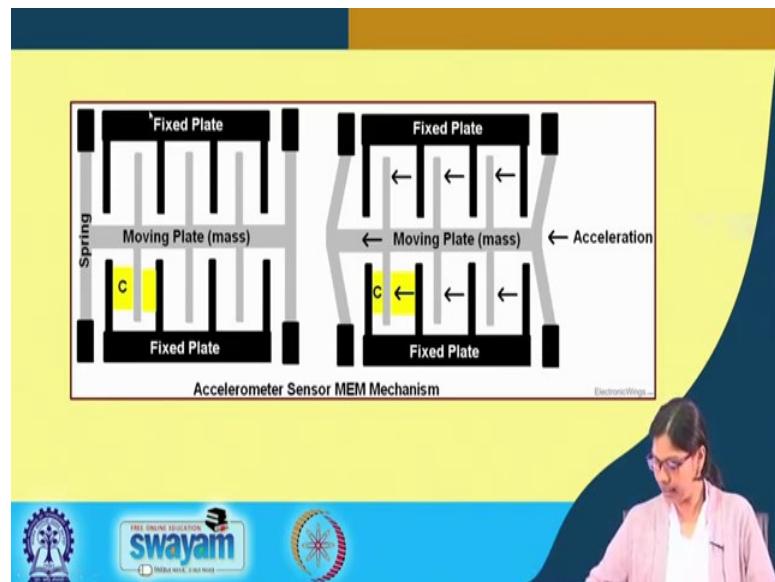


The displacement of the mass can be measured using capacitive sensing or piezoelectric effect sensing. So, the displacement of this mass when it is moved from one point to another, can be measured using either capacitive sensing or through piezoelectric effect sensing. What happens basically is there is a change in capacitance or there is a generation of voltage through which it is measured. The basic structure of the accelerometer consists of a fixed plate and a moving plate. And the acceleration deflects the moving mass and unbalances the differential capacitor, this results in a sensor output as voltage that is proportional to the acceleration.

What it means basically is there is a moving mass. When some acceleration is made this deflects the moving mass, and unbalances the differential capacitor. So, there is an imbalance in the mass. By measuring the acceleration along the x, y and z direction, it is possible to calculate the inclination or the tilt.

How much it is tilted can be accurately found out using the accelerometer. This can be done by measuring the acceleration along the x, y and z directions. It is also possible to calculate the angles of rotation along x, y and z axes that are called roll if it is along x axis; pitch if it is along y axis; and yaw if it is along z axis.

(Refer Slide Time: 06:21)



Here we have these fixed plates, and here we have the moving mass. And you see that whenever there is some acceleration, this particular yellow thing that is there it changes. Based on that we can accurately find out whether it is tilted more towards x, or y, or z direction. This is the accelerometer sensor with MEMS mechanism.

(Refer Slide Time: 07:16)



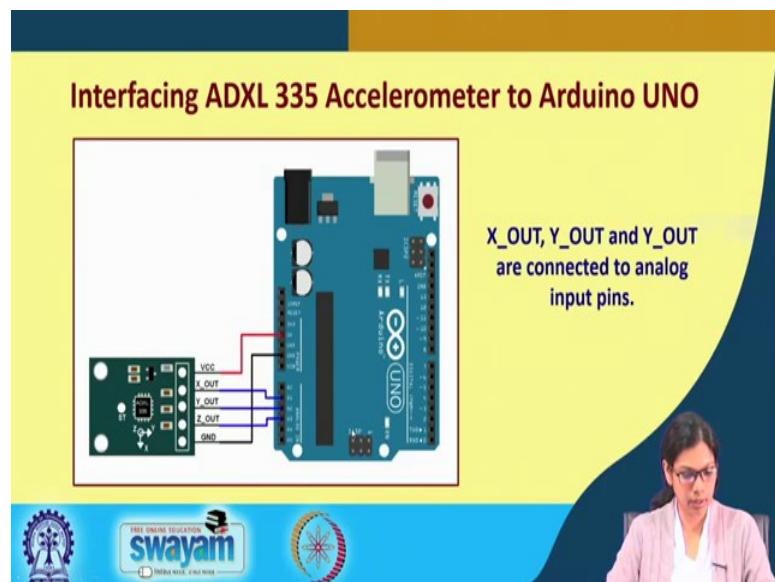
The accelerometer that we will be using in this experiment is ADXL-335, where you can see that these are the pins of this accelerometer. You have this Vcc, we have this GND,

and we have x, y and z coordinates. So, this is from the other side and this is from the other one.

The ADXL 355 is a small, thin, and low-power 3-axis accelerometer with signal condition voltage output. What it can do basically it can measure the static acceleration due to gravity in tilt sensing applications as well as dynamic acceleration resulting from motion, shock or vibration. This can measure acceleration within the range of $\pm 3g$ in the x, y and z axes.

When we connect this with STM, we will be seeing that we are mostly concerned about this X_OUT, Y_OUT, and Z_OUT. So, we will connect we will be connecting these signals to analog ports. The output signals of these are nothing but some analog signals that are proportional to the acceleration.

(Refer Slide Time: 09:40)



Now, the interfacing is fairly very straightforward. We have shown it using Arduino. We have done the experiment using STM. This is where it is connected to Vcc, this is connected to GND. And this x, y, and z is connected to pin A1, A2, and A3.

So, this is all about this device accelerometer. Next we will look into how we can connect this accelerometer with STM board. We have already shown you the connection. We will look into that in the next lecture.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 39
Experiment Using Accelerometer

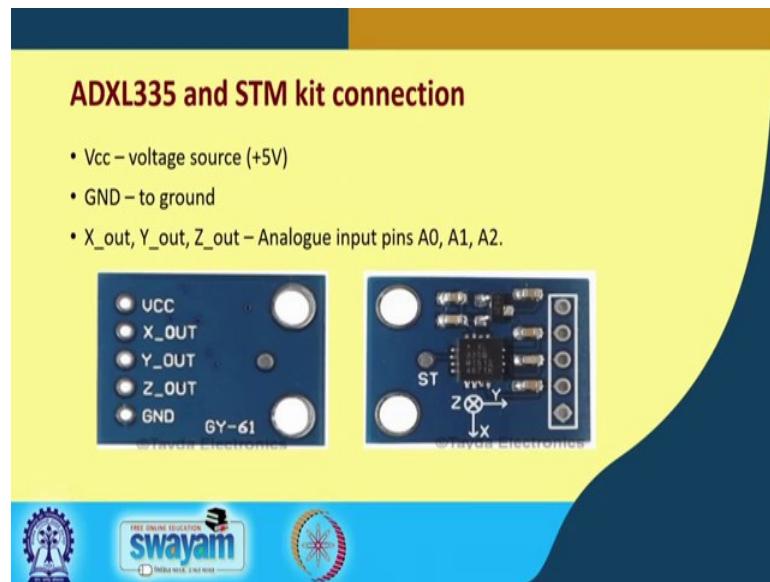
Welcome to lecture-39. In this experiment, I will be showing how we can interface accelerometer with STM board, and what value we will get for the different axis like x, y and z axis in CoolTerm. We already know how we have to use CoolTerm with STM board. For Arduino, it is straightforward it can be printed in the serial monitor.

(Refer Slide Time: 00:59)



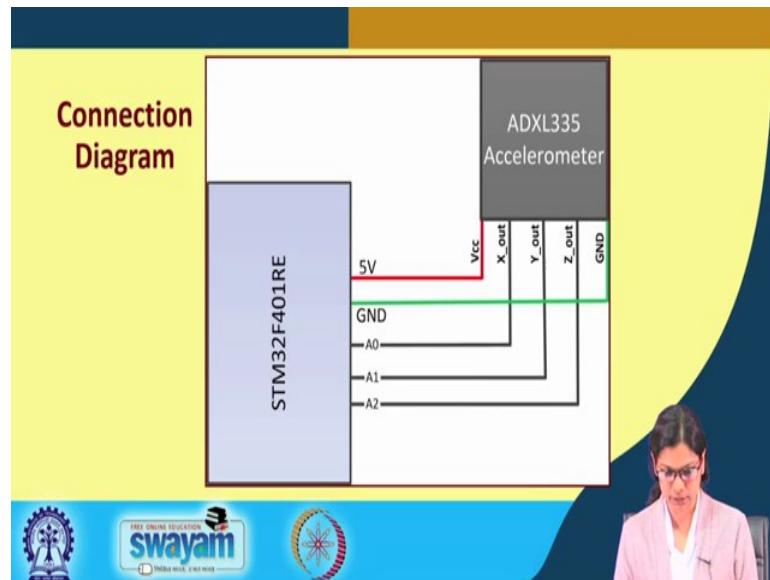
Firstly, I will show you the experiment with accelerometer, and then I will do the demonstration.

(Refer Slide Time: 00:07)

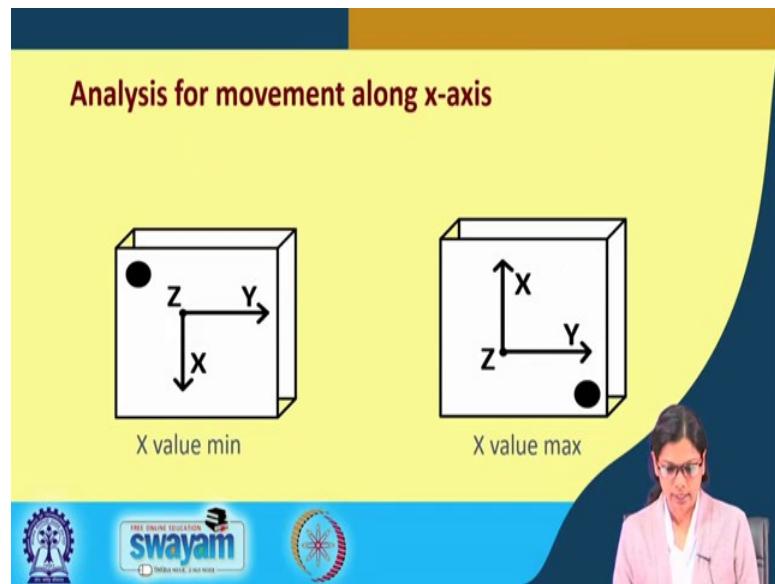


So, this is the ADXL-335 accelerometer, and I will be doing the connection with STM kit. So, the voltage source Vcc will be connected to +5 volt. The GND will be connected to the ground pin of the STM board. And the X_OUT, Y_OUT and Z_OUT analog pin outputs will be connected to the analog input pins A0, A1 and A2.

(Refer Slide Time: 01:46)

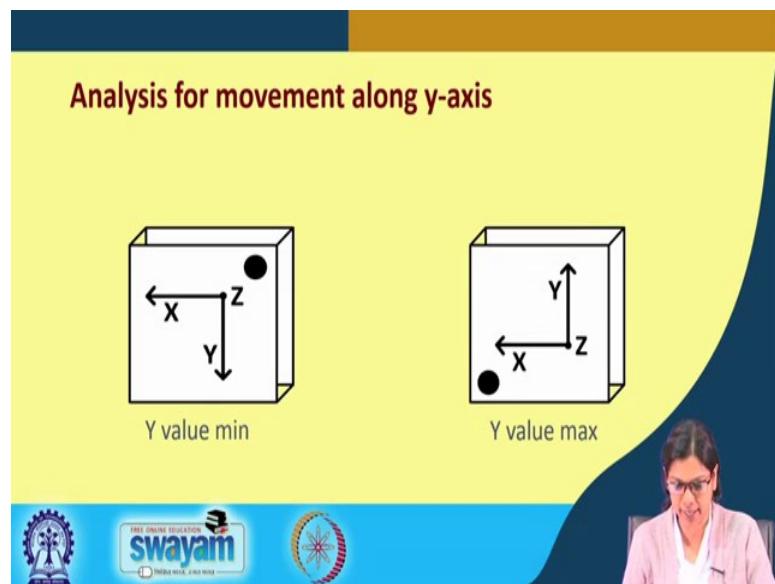


(Refer Slide Time: 02:09)



And then let us understand this analysis for the movement along x axis. When you hold the accelerometer, you will find out that that accelerometer is having certain axes, one is along X, another is Y and this one is Z. When you hold the accelerometer along this X, then you will see the highest value for this X. Similarly if you hold it like this, value of X should be minimum.

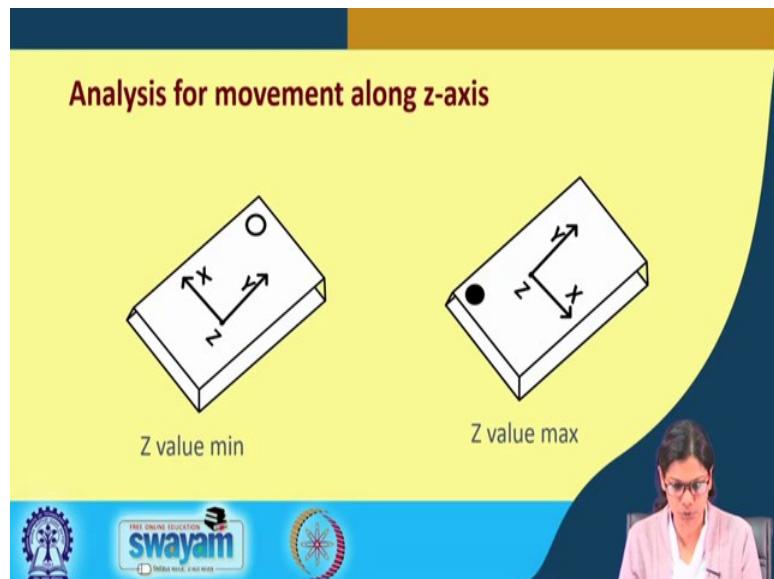
(Refer Slide Time: 02:54)



Similarly, for Y if you see, if you hold it this way, the Y value will be maximum; and if you hold it in this way, the Y value will be minimum. Depending on this you will be able

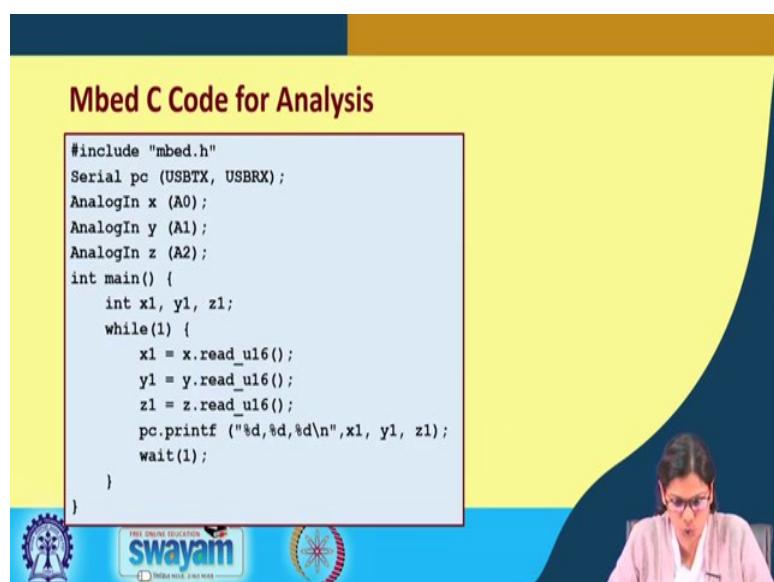
to actually understand how you are holding the device, whether the device is straight or it is tilted, or it is vertically tilted to the left, or it is vertically tilted to the right. So, different orientation of a device could be figured out from this value.

(Refer Slide Time: 03:27)



Similarly, if you hold the accelerometer in this direction, then the Z value will be minimum here, and the Z value will be maximum here. We will try this out with the STM kit.

(Refer Slide Time: 03:56)



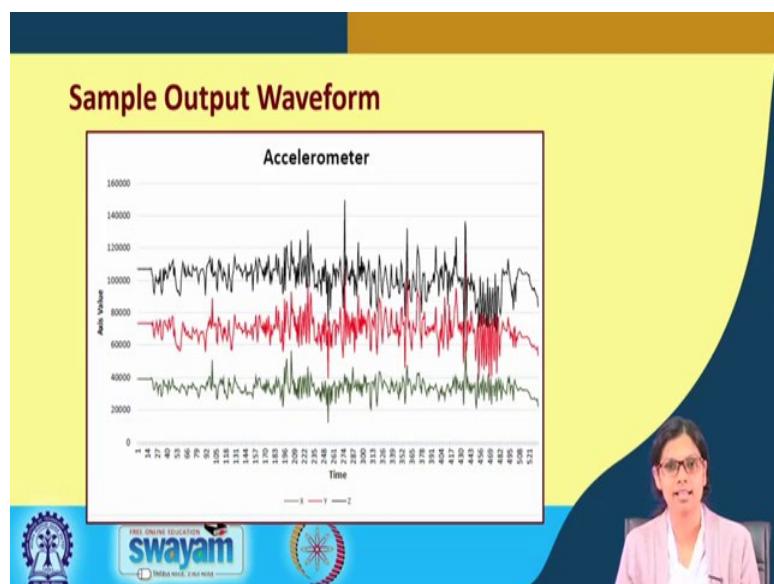
So, let us understand the mbed C code. This says that we have to include mbed library mbed.h. First we have to make the serial communication with USBTX and USBRX with the communication named as “pc”. We defined three analog input signals x, y, z and connected them to analog port numbers A0, A1 and A2 respectively. X OUT, Y OUT and Z OUT will be connected to the STM pins A0, A1 and A2.

Then we come to the main function. In the main function we define three variables x1, y1 and z1, where we are reading the analog value x1 using the function x.read_u16, for y1 it will be y.read_u16, for and z1 it will be z.read_u16. And with the serial communication with the name “pc” that we have already made, we will print the values of the x, y and z coordinate, which is stored in variables x1, y1 and z1.

And this is going in a while loop. After it gets printed it will wait for 1 second, and again it will do the same process.

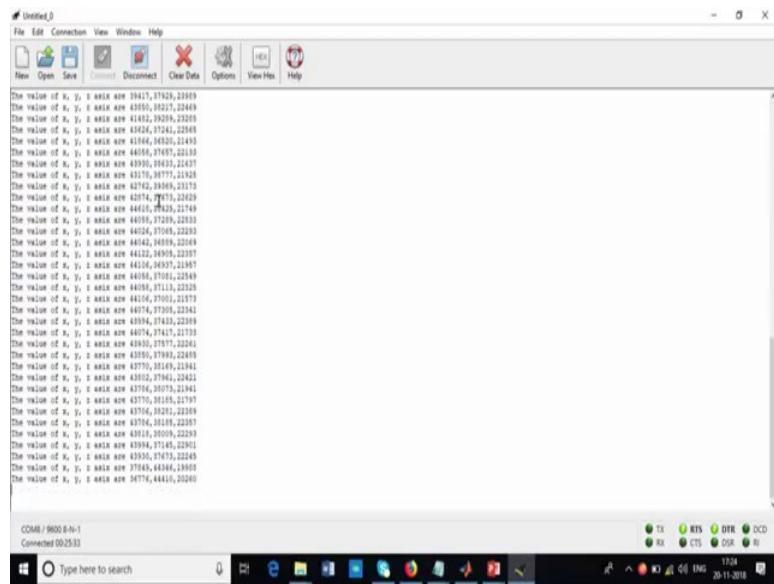
What you can do basically if you wanted to see the how the x, y, z coordinate values change, you can also use some kind of MATLAB code to display it that we are not showing here. But if you are interested, we can share those codes with you, you can try out at your end by plotting the x, y, and z coordinates. Or, you can do another thing, you can read this value, store it in some online database as we have shown you earlier using the SMS control. And whatever value is stored in the database, you can actually analyze it based on that you can plot that as well. So, there are variety of ways you can do it.

(Refer Slide Time: 07:38)



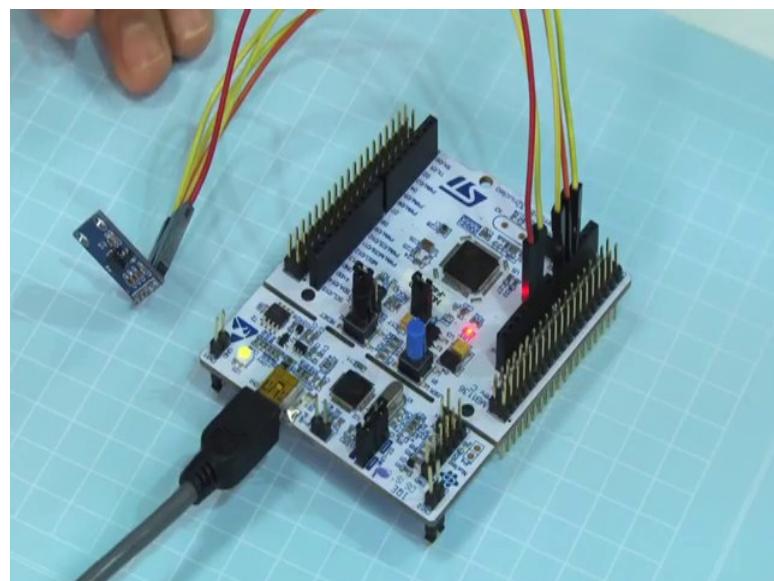
Finally, the plot looks somewhat like this. This is the sample output waveform from the accelerometer. This is the x-axis, y-axis and z-axis values, this is how they are changing. There are certain spikes as well we can see here and here. Now I will be showing you the demonstration of this using CoolTerm.

(Refer Slide Time: 08:27)



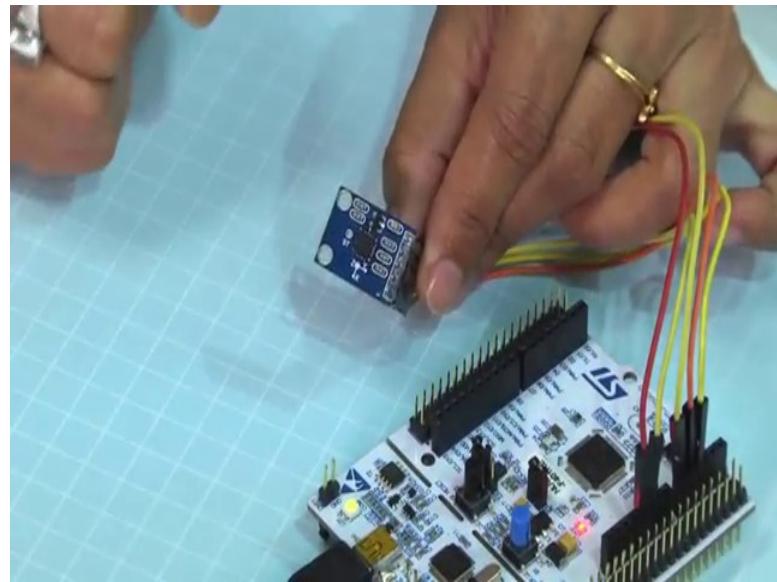
You have to use this CoolTerm; where I am clearing the data. So, some data are coming. First we will see the connection.

(Refer Slide Time: 08:42)



This one is the Vcc, which is connected to 5 volt. This is x axis that is connected to A0. This is y axis this one is connected to A1. And this one is z that is connected to A2. And finally, this one is connected to the ground pin. So, the connection is fairly straightforward, this is all required here for the connection.

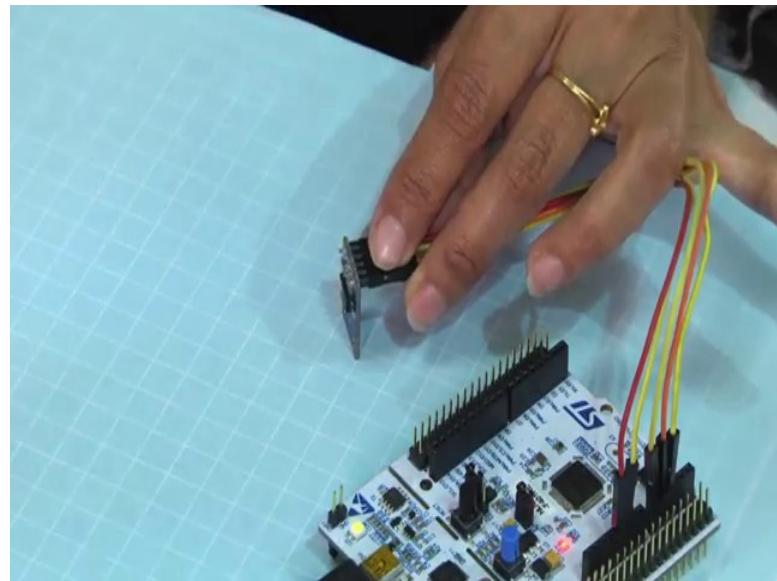
(Refer Slide Time: 09:40)



Now, if you see this accelerometer properly, you can see there is x axis here, here is the y axis, and this is the z axis. Whenever you hold this accelerometer with the arrow towards this x axis in this fashion, then the x coordinate value will be the highest.

Let us say I have hold the accelerometer in this fashion and will come and see the value of x in the hyper terminal. So now, see the x, y and z axis values, you can see the x axis value is 44058, the y axis is 36873 and of course, it varies. It is not the same although I have held it very tightly, but still there are some small variations.

(Refer Slide Time: 11:16)



Now, I will make sure that the y axis value will be maximum. I have put up the accelerometer with y axis at the top. So, now you can see in the hyper terminal what values you are getting. We are getting the x axis value as 36728. The y axis value is 44426, and the z axis value is 18628.

Let us say you have a small box in place where you wanted to see that the orientation of that particular thing should always be like ... the head of that particular thing should be at the top. So, you can always make sure that the x axis value should be high such that you have put up this accelerometer in that application in such a way that x axis is like this. So, if it is like this, the x axis value you will receive the highest one. And if it is not the highest, then the other two values are highest, then you know that your device is not properly placed. Some kind of alert may be given to indicate that the device is not properly placed.

From this experiment, what we have understood is that how we can interface the accelerometer to the STM kit. And if you want to find out the orientation of a device how we can find it out using this. And if we want a particular device to be placed in a particular way, we can ensure that using this. In most mobile device we have this accelerometer in place. If you have used some kind of apps like health monitoring, what it shows, it shows you that how many steps you have walked in a day.

These are measured using this device. Inside our mobile phone, we have an accelerometer or a gyroscope, which gives us the acceleration movement that we make in a day. Of course, there are some issues with these devices as well, because if you are just having a jerk, then also it will assume that you are walking. So, those things we need to look upon when we build an application.

Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institutes of Technology, Meghalaya

Lecture – 40
Experiment Using Bluetooth

Welcome to lecture-40. In this lecture, firstly I will be interfacing with the STM board a Bluetooth module. Along with that in previous two lectures, we have introduced you to accelerometer; with that device also we will be doing a couple of experiment.

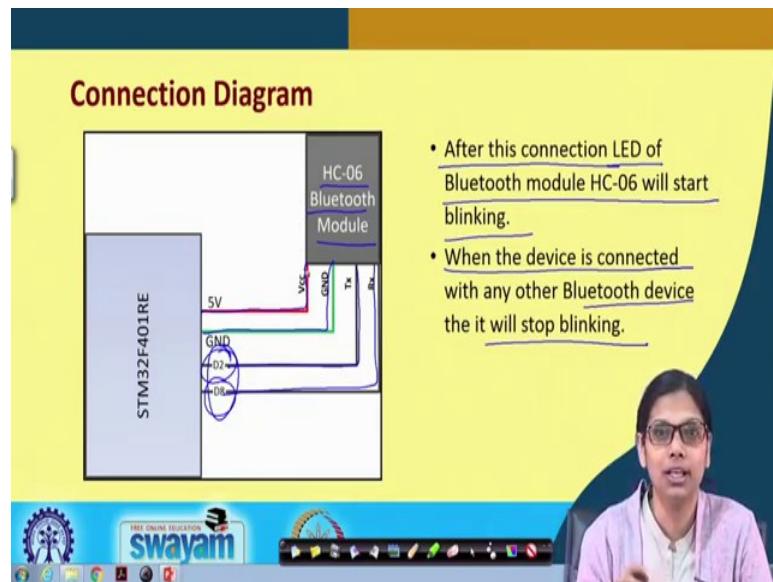
We will be showing how we can find out the orientation of a device, whether the device is lying flat or is vertically up or it is vertically down or it is horizontally left or it is horizontally right. These are the few orientations that we will find out.

(Refer Slide Time: 01:31)



We will first show how we can connect a Bluetooth module with the microcontroller board. And we already know how we can connect accelerometer, then we will figure out the orientation of any particular device, and we will determine that. And that will be displayed in a mobile phone, which I will be connecting through the Bluetooth module of with STM that is established between these two. And finally, we will show you the whole demonstration.

(Refer Slide Time: 02:16)



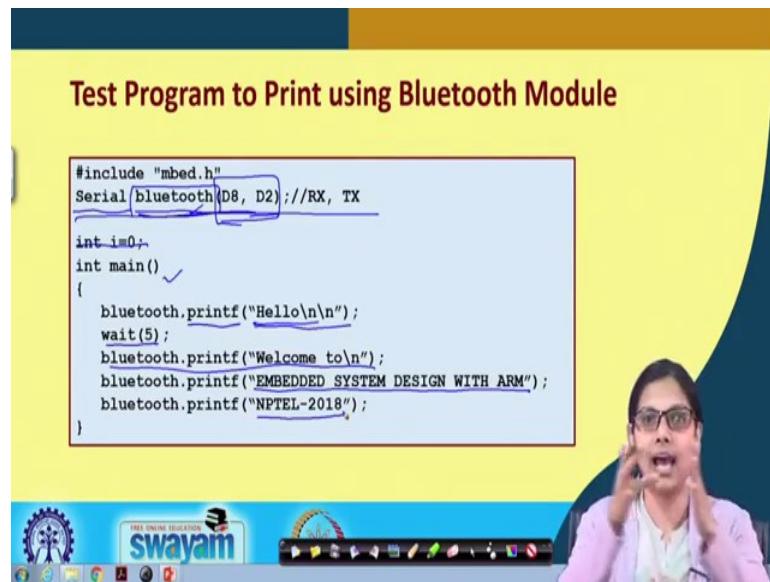
Now, the Bluetooth module that we are using here is HC-06. The connection is fairly straightforward. This is the Vcc, which is connected to 5 volt. This is the GND, which is connected to ground pin of STM. The TX is connected to D2, and RX is connected to D8. Similarly, we have to build a serial connection between these two pins. We have to make one pin as RX, and another pin as TX.

Now, I will be showing you in the demonstration how exactly it works when you connect a Bluetooth module. First of all the Bluetooth module have to be interfaced with the STM. Once the connection is made, the LED of the Bluetooth module will start blinking. This means that the connection has been established.

Next when the device is connected, so the Bluetooth module will be used for communicating between two device. So, when you will be connecting the Bluetooth module connected with the STM board to your mobile phone Bluetooth device, these two will communicate with each other. So, when you do that, the LED in the Bluetooth device will stop blinking, that means the connection between the two device has been established.

These are the two things from which you can find out that in the first phase when you just connect the Bluetooth module with microcontroller, when it starts blinking that mean the connection is built. But, when it is connected with another device, then it will stop blinking.

(Refer Slide Time: 05:12)



```
#include "mbed.h"
Serial bluetooth(D8, D2); //RX, TX

int i=0;
int main()
{
    bluetooth.printf("Hello\n\n");
    wait(5);
    bluetooth.printf("Welcome to\n");
    bluetooth.printf("EMBEDDED SYSTEM DESIGN WITH ARM");
    bluetooth.printf("NFTEL-2018");
}
```

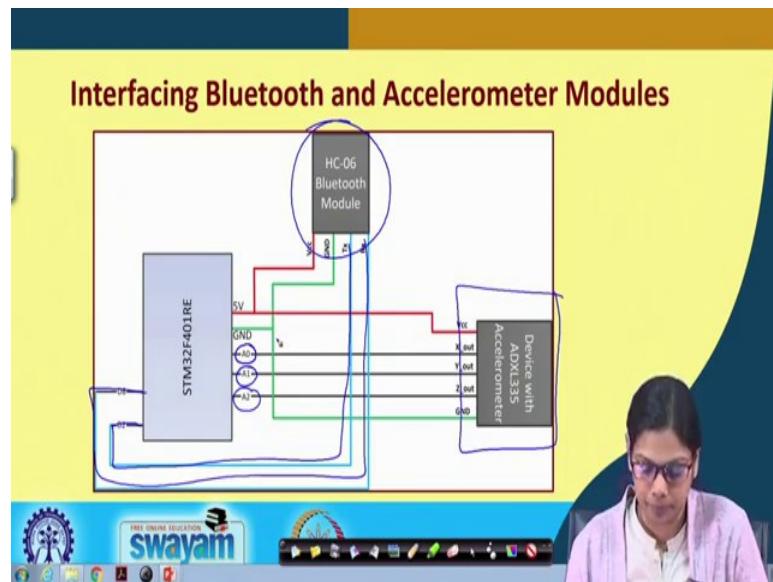
First I will show you a test program, where I will be connecting the Bluetooth module with STM board along with my mobile phone Bluetooth module, and it will display some text. Some text from the microcontroller board will be sent to my mobile phone through Bluetooth.

So, first we have to create the serial communication; the name that I have given is “Bluetooth”, with these two pins D8 and D2. This is the first thing, we have to initialize when we write the code. And then in the main what we are doing, actually we do not require this. In main what we are doing, the connection which we have made with the name Bluetooth between D8 and D2, we will use printf to print “Hello”.

So, once the connection is built with the Bluetooth module of STM along with the Bluetooth module of my mobile phone, we will transfer the data. The transfer of data takes place in this way, the object that we have created bluetooth.printf will print Hello. Then we will wait for 5 seconds. And then we will print these few lines of message.

Your microcontroller along with that Bluetooth becomes one device, and your mobile phone is another device through which you will be making the connection. This is a straightforward code, where basically we have made this connection making one TX and another RX.

(Refer Slide Time: 08:14)



Now, we will go ahead with the interfacing. This is fairly straightforward, because we have already discussed about Bluetooth.

I have already discussed about the accelerometer connection. We can make this connection, and connect to analog ports A0, A1, and A2. And with the Bluetooth module, it is connected to D2 with the TX, and D8 with the RX. And Vcc, GND for the Bluetooth module, and Vcc, GND for the accelerometer are connected.

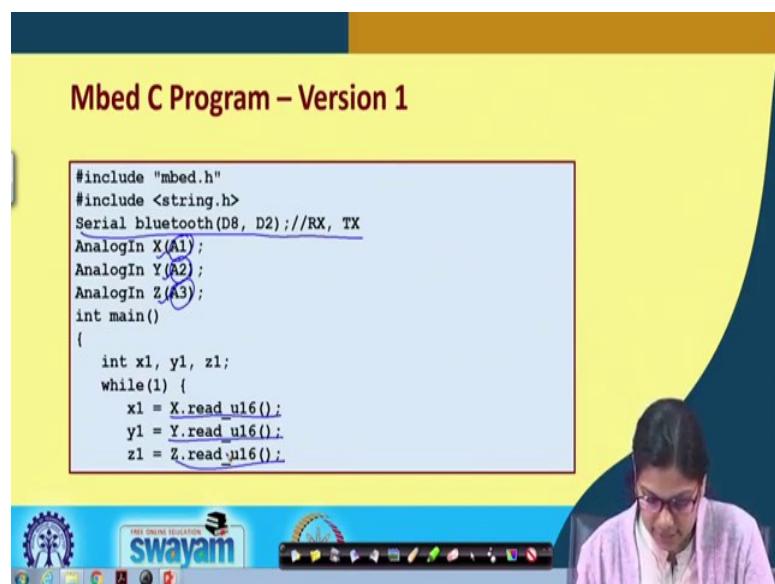
(Refer Slide Time: 09:13)



We come to the first experiment. I will be discussing the code first, and then I will demonstrate. In the first experiment will determine the orientation of the device. In the previous lecture, we have shown you that we are receiving some values corresponding to x, y, z coordinates. Based on that coordinate values, we have to test. When I put accelerometer along with a device, and we make it vertically straight, then what changes happen, and so on.

And accordingly we will write the program. If that x coordinate value is in between this and this or the y coordinate value is in between this and this or to z coordinate value is in between this and this, then it is vertically up or vertically down. In this experiment firstly will look into the orientation of the device. The connection diagram with Bluetooth module, and with accelerometer will be the same.

(Refer Slide Time: 10:44)



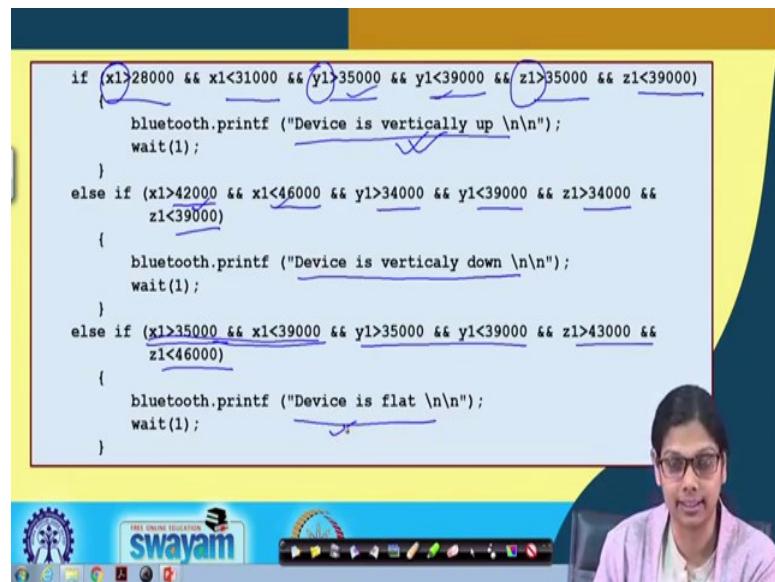
Mbed C Program – Version 1

```
#include "mbed.h"
#include <string.h>
Serial bluetooth(D8, D2); //RX, TX
AnalogIn X(A1);
AnalogIn Y(A2);
AnalogIn Z(A3);
int main()
{
    int x1, y1, z1;
    while(1) {
        x1 = X.read_u16();
        y1 = Y.read_u16();
        z1 = Z.read_u16();
```

This is the Mbed C code. In the Bluetooth module, it will be displayed whether it is vertically up or it is vertically right. Then we will be connecting X, Y and Z outputs to A1, A2, and A3, you can connect to any analog port.

Then we are reading the value. We are reading the three values that is X.read_u16, Y.read_u16, and Z.read_u16. After reading the values, we need to check.

(Refer Slide Time: 11:38)



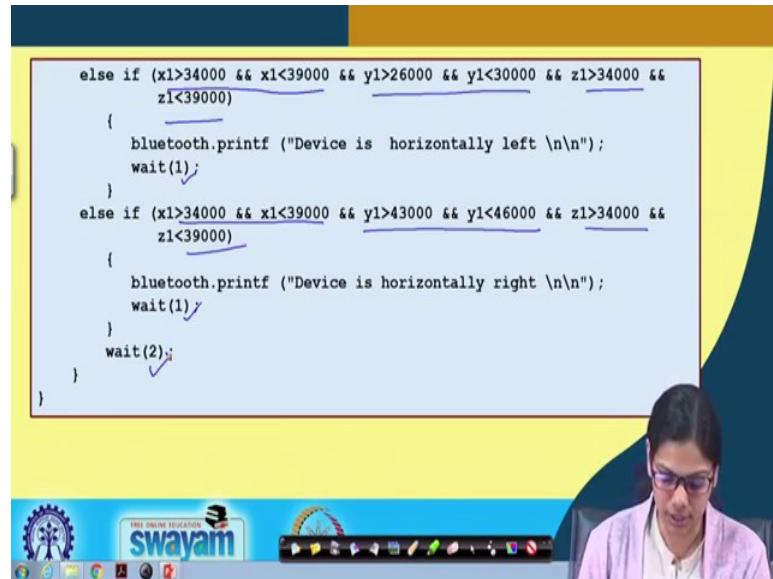
Here x1 is the X_OUT value, y1 is the Y_OUT value, and z1 is the Z_OUT value. We have seen that when we are getting x1 between this and this, y1 between this and this, and z1 between this and this, then the device is considered to be vertically up.

After doing this, the bluetooth.printf function will print the appropriate message. Similarly, for the other possible orientations..

This is just a representative thing that we have done. When you do it, I have already shown you that how you will be looking into the values, if you use Arduino, you can see it in the serial monitor; else you use CoolTerm where all these values will get displayed. You hold the device vertically up, and see what values of x, y, z coordinate you are getting.

Accordingly you can have the code written. Similarly, there are few more conditions like the device lying flat, when it is lying flat you see x coordinate value, and y coordinate value are to some extent same, but the z coordinate value is the highest.

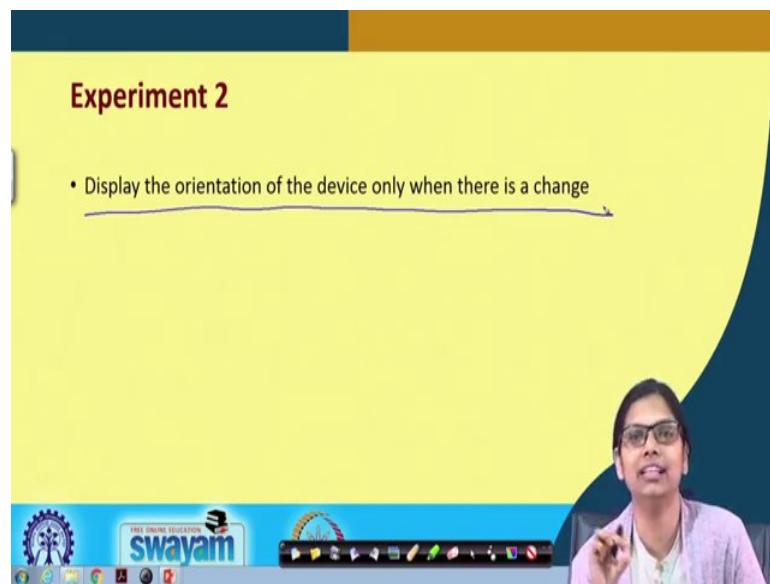
(Refer Slide Time: 14:14)



We have few more conditions, when the device is horizontally left. When it is horizontally left, when we have the value of x1, and of course z1 also we can see in the range of 34000 to 39000, and y1 value is less here. Similarly, here it will be the opposite; this value is in this range, but y1 value is highest, then it is horizontally right. So, it depends on how you are putting the accelerometer in the device also. You have to see that and write your program accordingly.

Then after every display there is a wait of 1 second, and after this there is a wait of 2 seconds, and everything repeats.

(Refer Slide Time: 15:25)



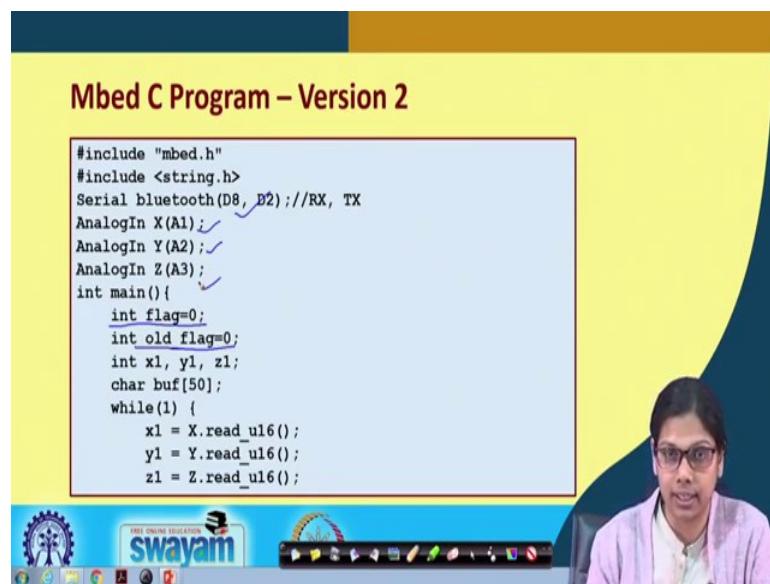
Experiment 2

- Display the orientation of the device only when there is a change

Let us go to another experiment. In the previous experiment, we have done it in a fashion that whenever the device orientation changes, after every 2 seconds it is displaying the status.

Here we have just made one change, where it will display the orientation of the device only when there is a change. That means, if the device is flat, it will remain flat. If there is a change from flat to vertically up or vertically down, then only it will get displayed in the Bluetooth module. Let me let me show you, what change I have done here.

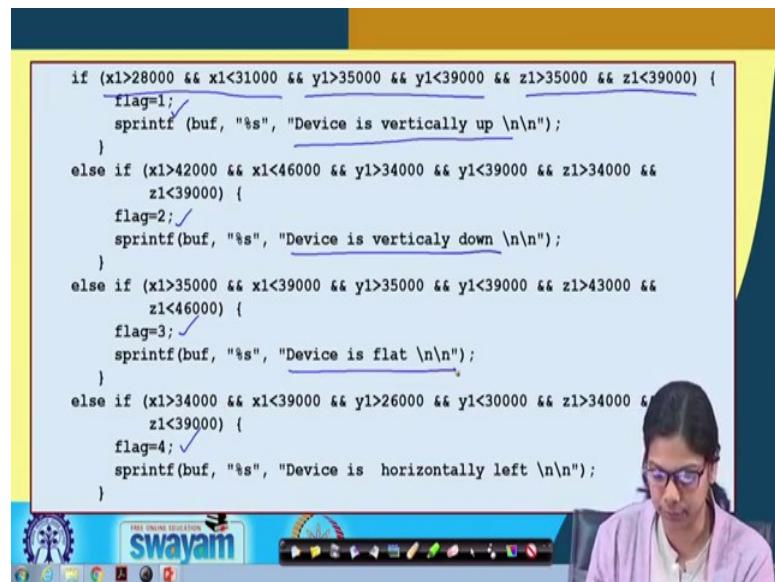
(Refer Slide Time: 16:27)



```
#include "mbed.h"
#include <string.h>
Serial bluetooth(D8, D2); //RX, TX
AnalogIn X(A1);
AnalogIn Y(A2);
AnalogIn Z(A3);
int main(){
    int flag=0;
    int old_flag=0;
    int x1, y1, z1;
    char buf[50];
    while(1) {
        x1 = X.read_u16();
        y1 = Y.read_u16();
        z1 = Z.read_u16();
```

This part of the code is the same basically, but we have taken two variables. One is flag, we have initialized it to 0. Another is old_flag, which is also initialized to 0. So, these are the two things that we have done. For all other things, it is pretty similar. We have built a serial connection, then A1, A2, A3 analog ports are connected with X, Y, Z out of the accelerometer.

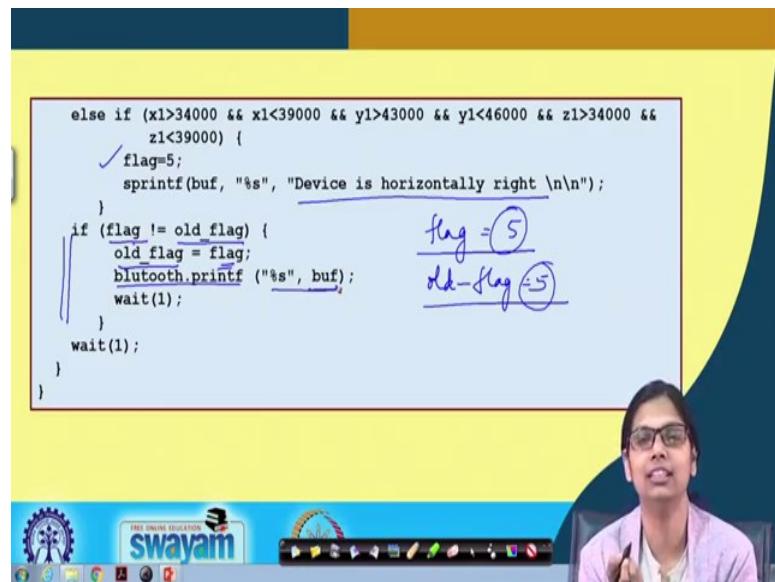
(Refer Slide Time: 17:01)



```
if (x1>28000 && x1<31000 && y1>35000 && y1<39000 && z1>35000 && z1<39000) {
    flag=1;
    sprintf (buf, "%s", "Device is vertically up \n\n");
}
else if (x1>42000 && x1<46000 && y1>34000 && y1<39000 && z1>34000 &&
z1<39000) {
    flag=2;
    sprintf(buf, "%s", "Device is vertically down \n\n");
}
else if (x1>35000 && x1<39000 && y1>35000 && y1<39000 && z1>43000 &&
z1<46000) {
    flag=3;
    sprintf(buf, "%s", "Device is flat \n\n");
}
else if (x1>34000 && x1<39000 && y1>26000 && y1<30000 && z1>34000 &&
z1<39000) {
    flag=4;
    sprintf(buf, "%s", "Device is horizontally left \n\n");
}
```

And then let us see the program here, what we are doing basically that the conditions would be pretty same, if you see one. Whenever we have either it is vertically up or it is vertically down, what is basically done is that we are initializing a flag. In the first case the flag is 1. In the next case, the flag is 2. In the third case, the flag is 3. In the next, flag is 4 depending on the orientation.

(Refer Slide Time: 17:44)



```
else if (x1>34000 && x1<39000 && y1>43000 && y1<46000 && z1>34000 && z1<39000) {
    ✓ flag=5;
    sprintf(buf, "%s", "Device is horizontally right \n\n");
}
if (flag != old_flag) {
    old_flag = flag;
    bluetooth.printf ("%s", buf);
    wait(1);
}
wait(1);
}
```

flag = 5

old_flag = 5

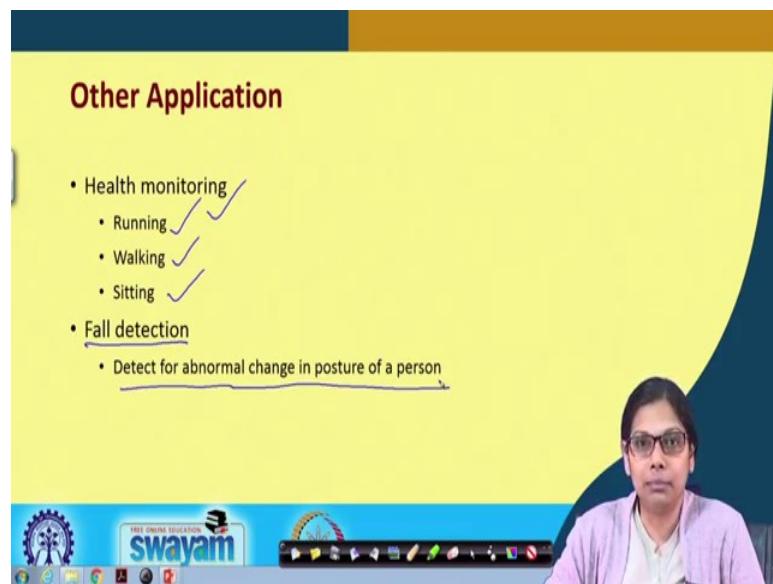
And then we see that it is again horizontally right with flag 5. And what we are doing in this check, we are seeing flag (initially the flag value was 0), and then in any one of these conditions the flag value will change to either 1, 2, 3, 4 or 5.

If flag not equals to old_flag, there is a change. For the first time the condition will be true, because flag is not equal to old_flag. The old_flag value is replaced by flag.

Suppose the device was horizontally right. Then flag value will be 5. For the first time this old_flag value was not equal to this value. So, old_flag value will become 5 here, and then we print the “Device is horizontally right”. We wait for one minute, and again check flag not equal to old_flag value, but no both are same. So, whatever is displayed will remain.

Let us say from horizontally right, we change it to horizontally left. In that case, what will happen is that this flag value will change. So, Bluetooth will display a text like the device is flat, etc.

(Refer Slide Time: 19:37)



Other Application

- Health monitoring
 - Running ✓
 - Walking ✓
 - Sitting ✓
- Fall detection
 - Detect for abnormal change in posture of a person

These are the two experiments that we will be showing you in the demonstration. There are other applications of accelerometer as well. One of the burning applications is in health monitoring that we have seen in our mobile phones.

Basically it will tell you that in a day how much time you have run or how much time you have walked, how much time you are sitting and so on. You can keep a track of your activities that you are doing throughout the day. And also there is a very vital application like fall detection mainly for elderly people.

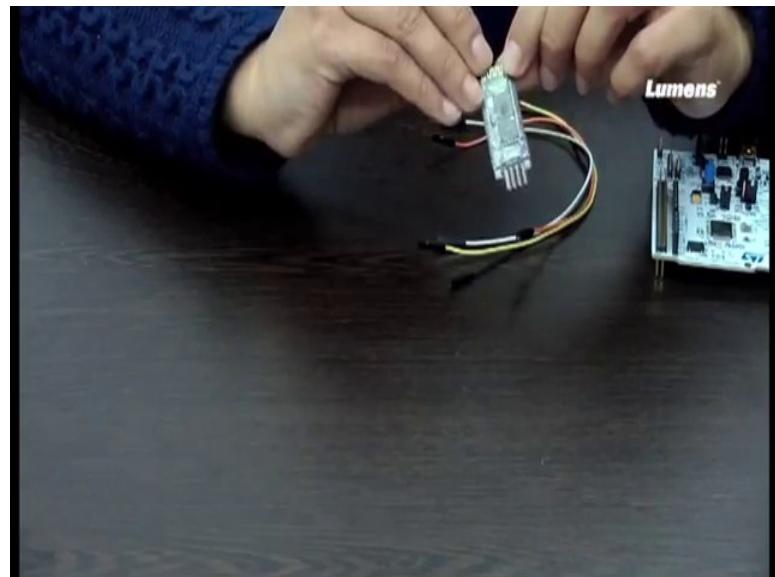
You see when old people generally have to stay back in their house alone most of the time. Even if there are people to look after them, but might be in certain situations, we find them all alone. And under such condition, if nobody is around them and there is some serious issue like they fall down in bathroom or in house only, then how do their near ones will come to know.

This fault detection application can be very nicely done using this accelerometer, but we need to keep various false positive conditions, it should not generate some false alarm. So, we have come to the end of this lecture. Now, we will be showing you the demonstration.

In the final week, we have been working with one of the device that is accelerometer. And what it does I have already explained. Now, what I want to do is that the coordinates that we received from this accelerometer, I want to send them to my mobile phone. We need to have some communication through which my mobile phone and the microcontroller with which it is connected with the accelerometer should communicate.

So, in this process we need a Bluetooth module. We will be connecting a popular Bluetooth module that is HC-06, where we will be sending some data from the microcontroller to my mobile phone. So, why we are requiring this? Because later what we will do is that we will connect accelerometer, and will transmit the x, y, z coordinates through Bluetooth to the mobile device.

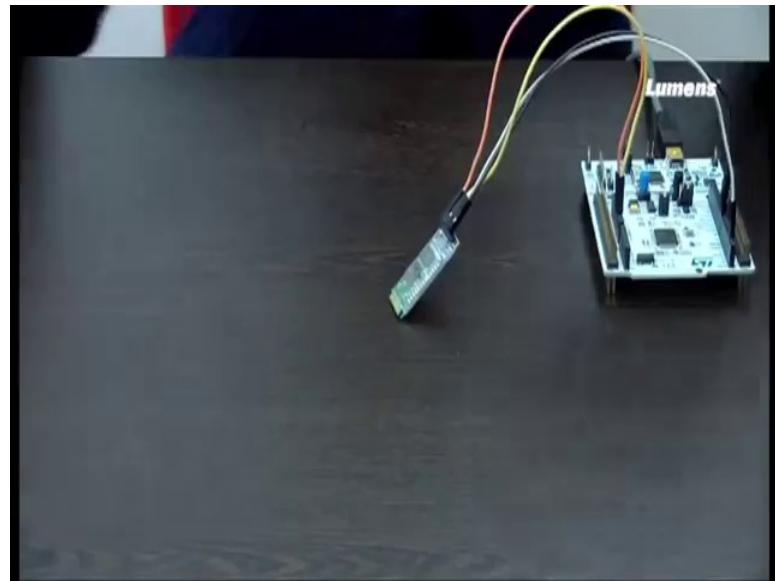
(Refer Slide Time: 23:50)



Let us see the connection that we have to do for this Bluetooth with STM board. This is the Bluetooth module that is HC-06, you can see it has got four pins, so the four pins goes like this. It has got an RX, it has got a TX, it has got a GND, and it has got a Vcc.

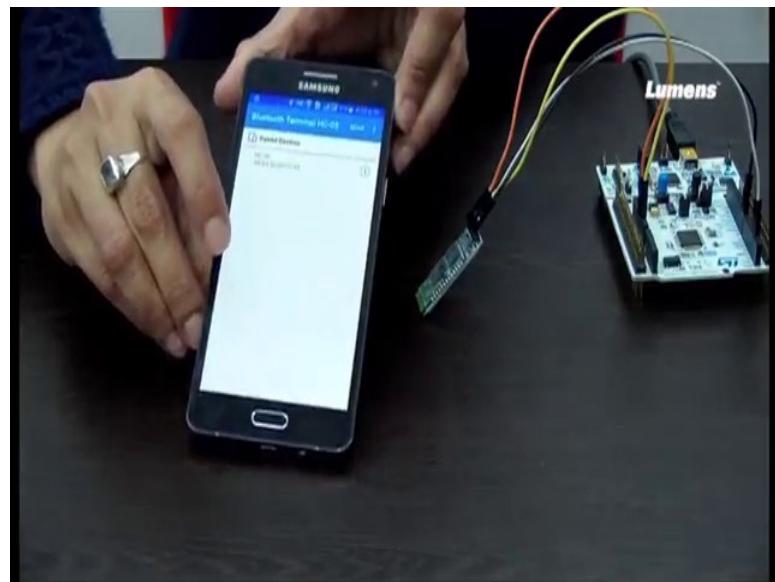
I will be connecting with the GND, Vcc, TX, and RX pins of the STM microcontroller with which we were working. First we will just connect these two, and we will send a data from this microcontroller to my device. For doing this also you need one more thing that you have to install a Bluetooth terminal, I am coming to that a little later.

(Refer Slide Time: 25:05)



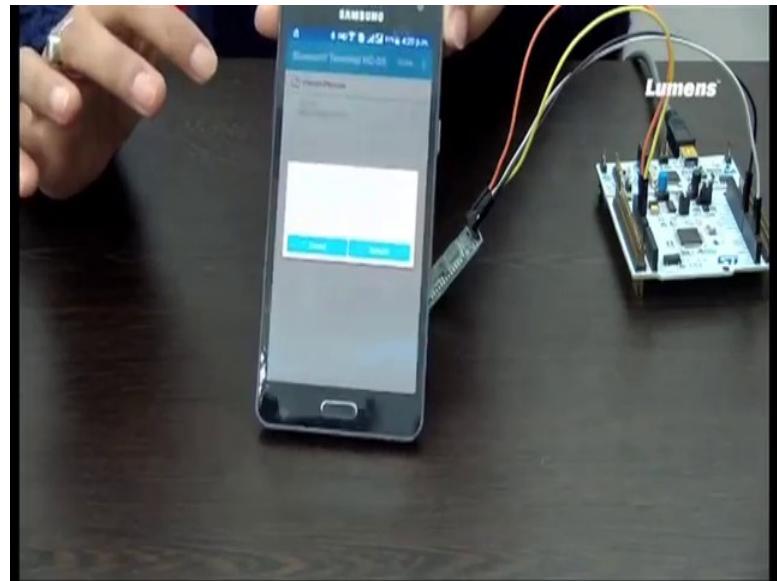
First let me show the connection that we have to do. As I said this is Vcc, which is connected to this one, this is GND, which is connected to the ground port of the STM board. And then we have TX and RX. The TX is connected to port D2. And the RX is connected to D8.

(Refer Slide Time: 26:40)



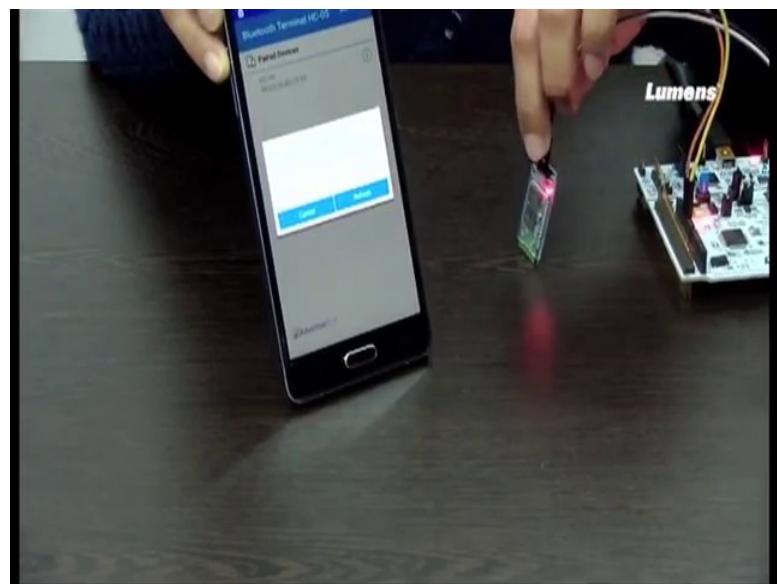
Once you have this Bluetooth module connected, you also need a Bluetooth terminal. So, I have already installed a Bluetooth terminal, this is the Bluetooth terminal that I have already installed.

(Refer Slide Time: 27:03)



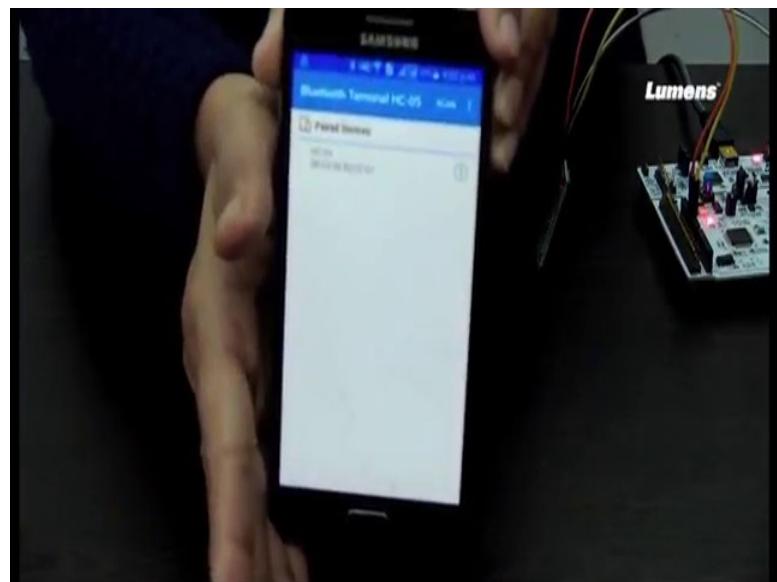
So, here you can scan, and then it will scan and will actually display what all devices are nearby. You see when I connect this LED is blinking, basically this means that it is ready for connection, it can connect to any nearby devices.

(Refer Slide Time: 28:07)



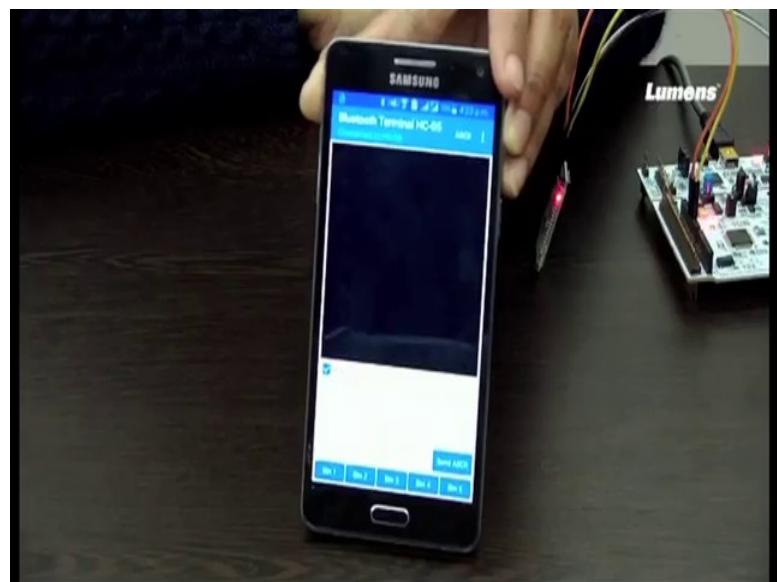
Now, this is my device, this is my mobile device, which I will be scanning first. So, we are scanning, and you just notice one thing that the LED is blinking in this Bluetooth module. No device found, it says.

(Refer Slide Time: 29:09)



Now, it is showing that there are some devices, as I have already connected previously. You can see that the pair device, it is showing up here. This one is the pair device, so I will just select HC-06.

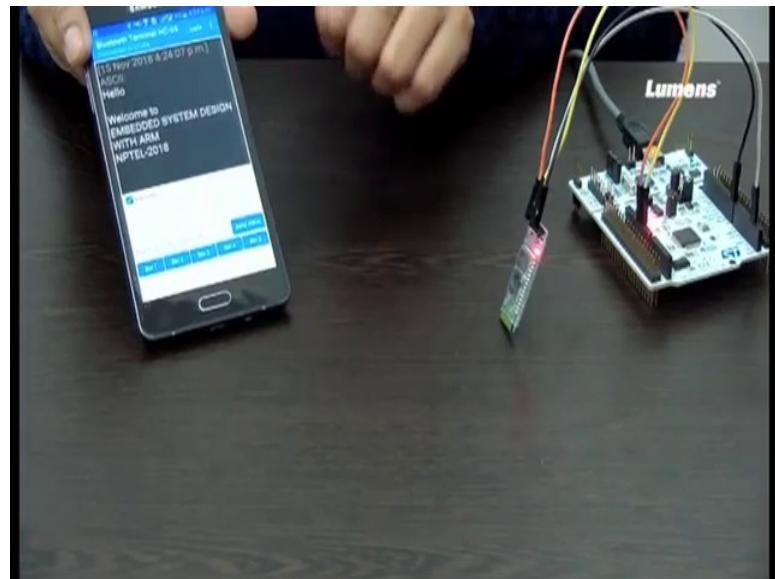
(Refer Slide Time: 29:29)



I will just click on that and you can see a terminal is coming. Now, see what has happened. Once the connection is established with this mobile, the LED has stopped blinking. Now, I will show you once more by switching off the connection. And you can see that the LED has started to blink again, so it is ready for connection. And when I

again connect, it has stopped that means it has connected. Now, I will send two messages through Bluetooth from this microcontroller to this mobile.

(Refer Slide Time: 30:45)

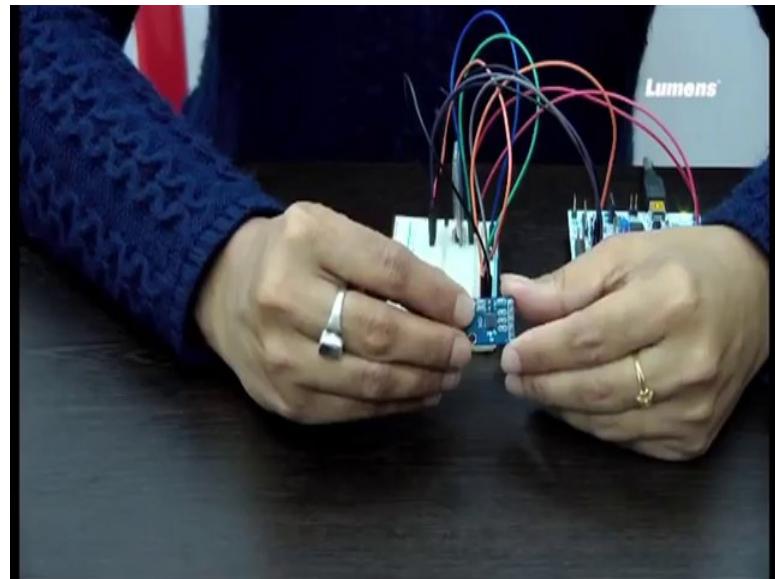


So, you can see two messages have come. This is how you can actually connect a Bluetooth module with any other device, and then you can send the data through your microcontroller and through this communicating device to any other mobile device.

Today, I will show you another experiment, where I will integrate accelerometer along with the Bluetooth connection that I have already shown. And then we will analyze the orientation of any object. So, what I will be doing, the accelerometer will be getting the x, y, z coordinates.

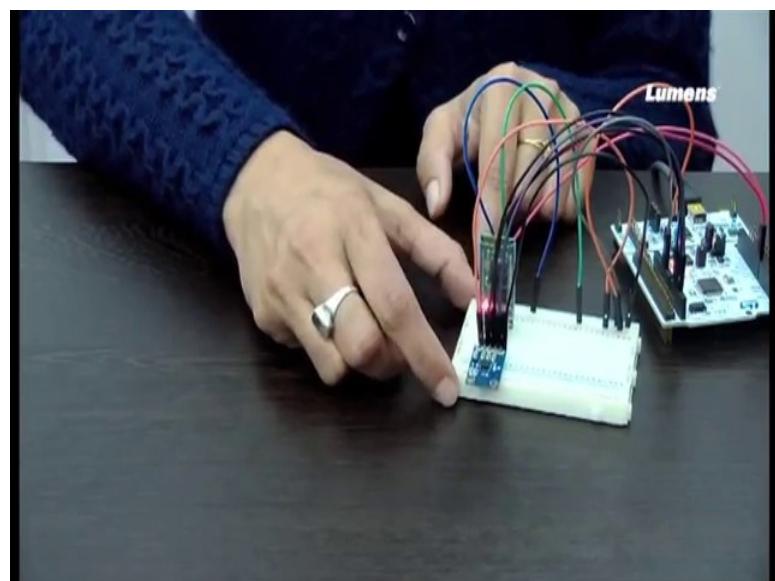
Then the microcontroller will do the needful, it will analyze to check whether the coordinates received signifies that the device is flat or the device is vertically up or it is vertically down or it is horizontally left or it is horizontally right.

(Refer Slide Time: 33:56)



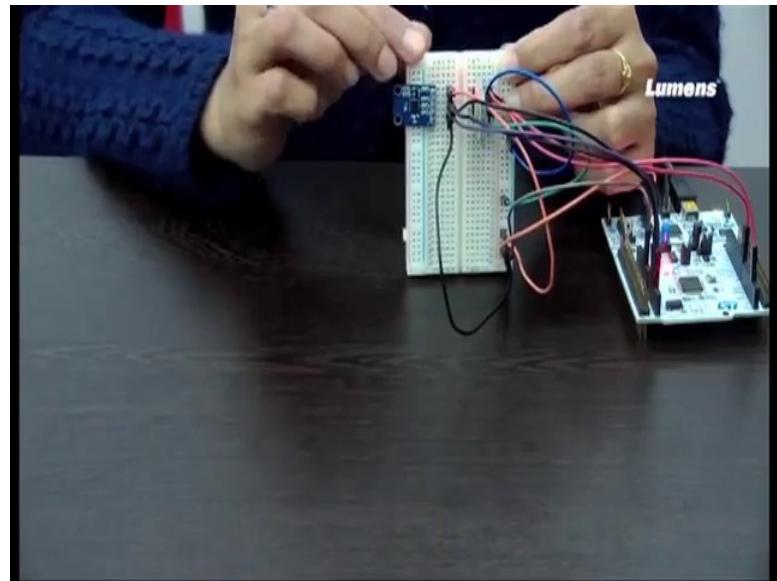
So, you see this is the accelerometer, it has got 5 pins. So, these pins are Vcc, x, y, z and GND.

(Refer Slide Time: 34:30)



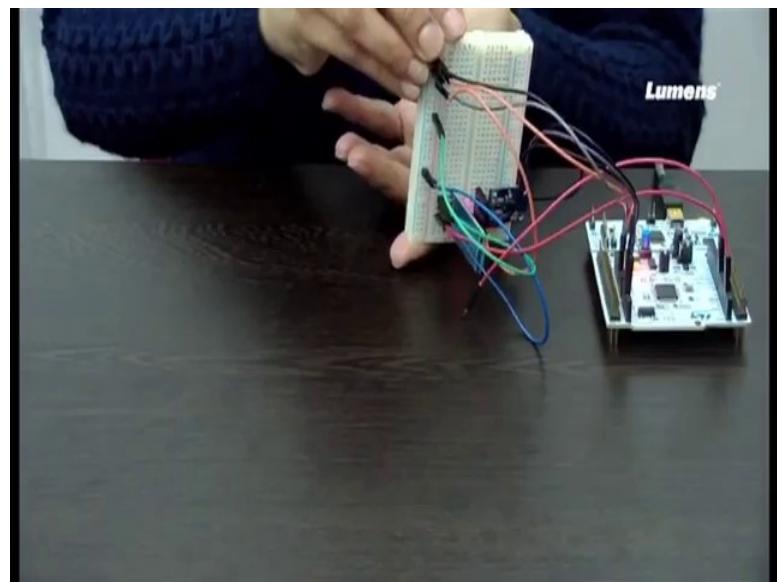
I will be connecting this to the breadboard, where I have already made the connection. You can see this I have connected with Vcc, this I have connected with GND, and then x, y, z.. And the Bluetooth connection we have already seen. And you can see that in the Bluetooth this red LED is blinking, meaning that it is ready for connection, but the device has not connected yet.

(Refer Slide Time: 35:19)



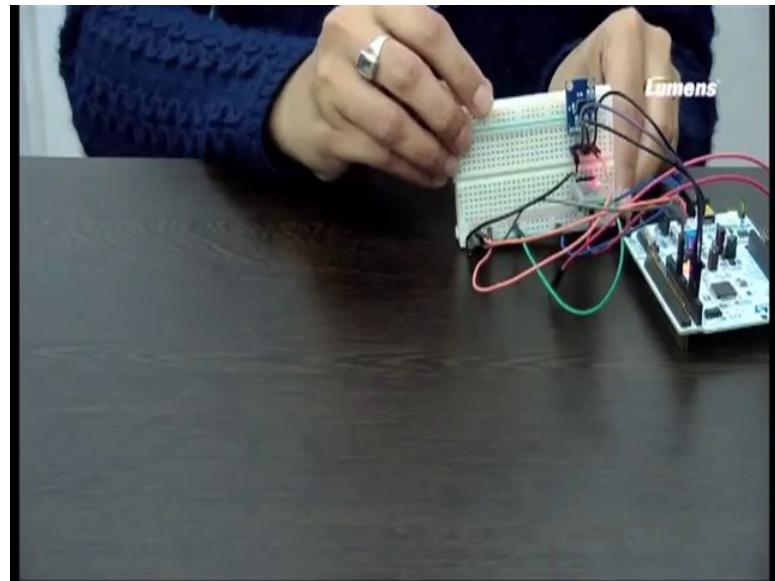
Now, the way we have made the program, when we place the device in this fashion meaning it is vertically up.

(Refer Slide Time: 35:27)



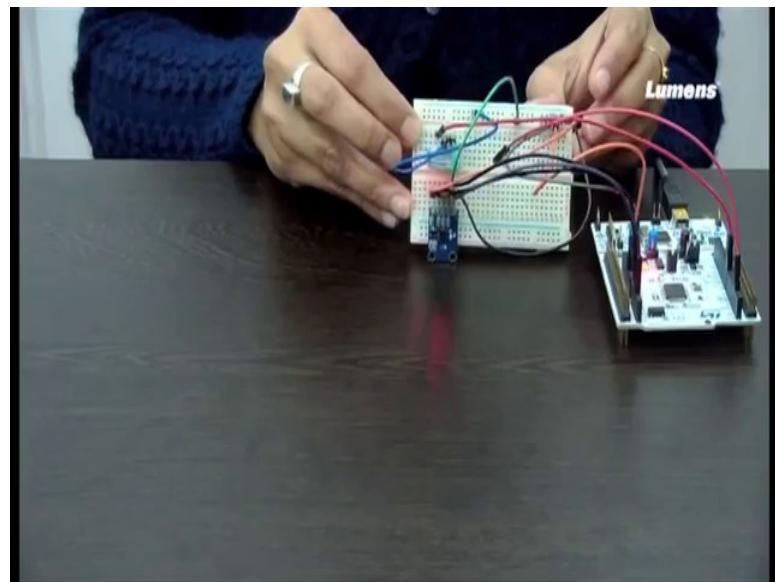
If you make it like this, it should say vertically down.

(Refer Slide Time: 35:36)



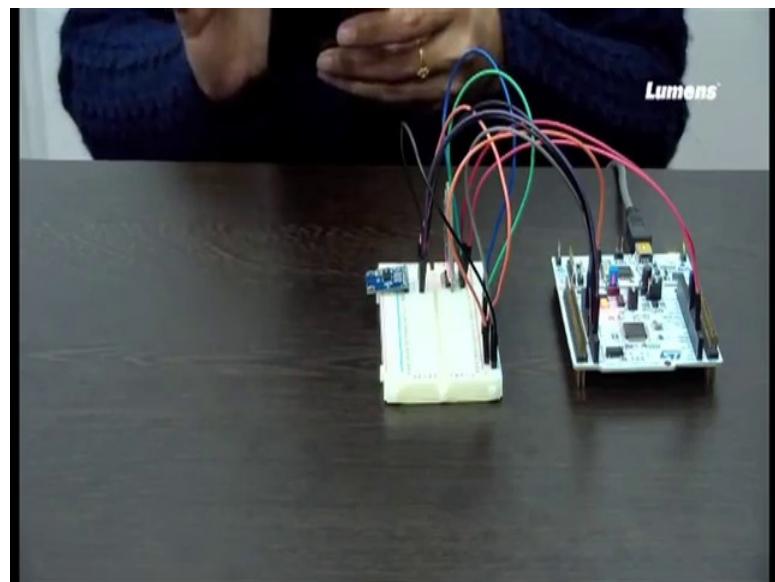
And then if you turn left, it will say horizontally left.

(Refer Slide Time: 35:46)



And when you do this, it will be horizontally right.

(Refer Slide Time: 36:08)

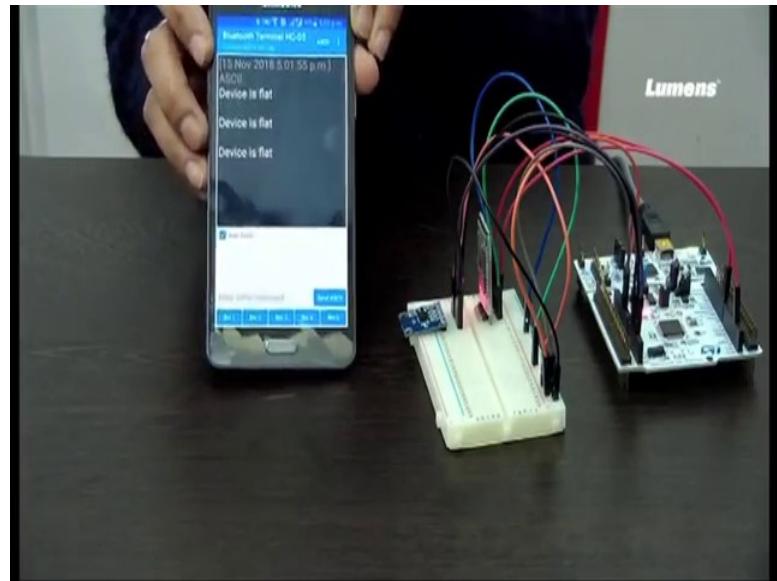


So, let me first do the needful. This is the connection with Bluetooth, we have already seen. And this is the connection I have made with this accelerometer.

And I said there are two set of codes that we have put; for one code it will continuously send what is the orientation of this particular device.

And in the next program whenever the orientation changes, then only it will send the data through this Bluetooth module. So, let me first connect to the Bluetooth module. It is still blinking you can see that, and now I will do the connection. And you can see that it is now stable.

(Refer Slide Time: 37:24)

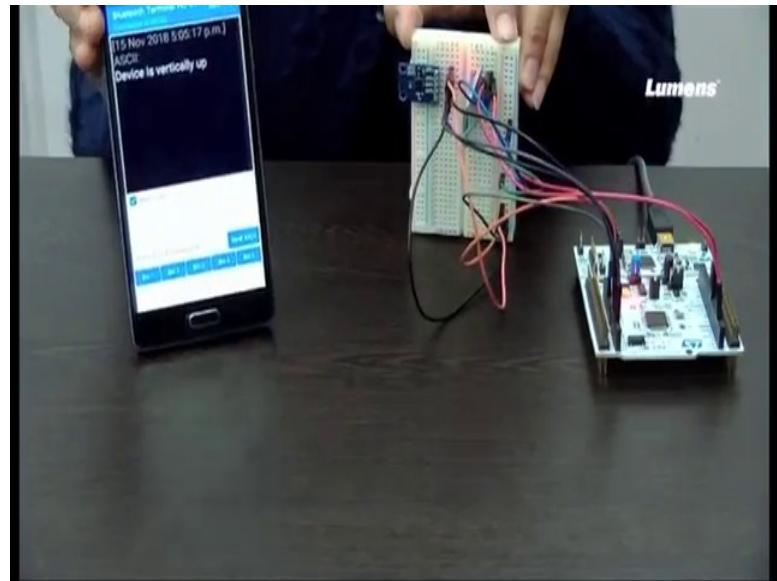


And now see what message is getting displayed, the device is flat, so it is lying flat. Now, I will make this device vertically up, so it is vertically up now. Now, I will make this device vertically down, now this is showing that the device is vertically down. And the device is flat now, every two second it is sending this. Now, again I have made it vertically up, which is displayed in this terminal. And then I will make horizontally left, I have done to the left side. And now I will make it horizontally right, so you can see that it is now horizontally right.

So, these are the few orientation, which we have made with respect to x, y, z axis. We have already seen that what value will receive on x, y, z axis. And depending on how we have put the accelerometer here, we have written the code accordingly.

So, these are the few things, we have to take into consideration when we do this. Now, I will again disconnect. And you can see that it has started to blink again,. Now, I again dump another code, where only it will change, if the orientation of this particular device changes, it will not change or it will not send anything if the position does not change.

(Refer Slide Time: 40:42)



Now, I am not changing the device. Now, let us say I have changed devices vertically. Now, if you recall in the previous experiment, it was continuously displaying that it is vertically up in 2 seconds, but now I am not displaying that. I am only displaying, when there is a change. Now, you see message has been sent that the device is flat.

So, whenever there is a change in orientation or change in position of this device that is the orientation, then only there is a message coming up, otherwise no. So, continuously we are not sending something, we are only sending something whenever there is a change in this particular device orientation. Now, it is horizontally right, now it is up. And it will not send any message unless there is a change in the position of this device.

The code for the same was already discussed, I have just shown the demonstration. There are various applications of this accelerometer, I have already discussed previously. Like you can track the number of steps a person is walking in a day. So, we have already seen that when we move the accelerometer in various position, in various ways, the x, y, z value changes and when we are sitting or we are moving in a very slow position, how the value changes.

You can actually take all these things into consideration while tracking human activity. And there are many other applications as well.

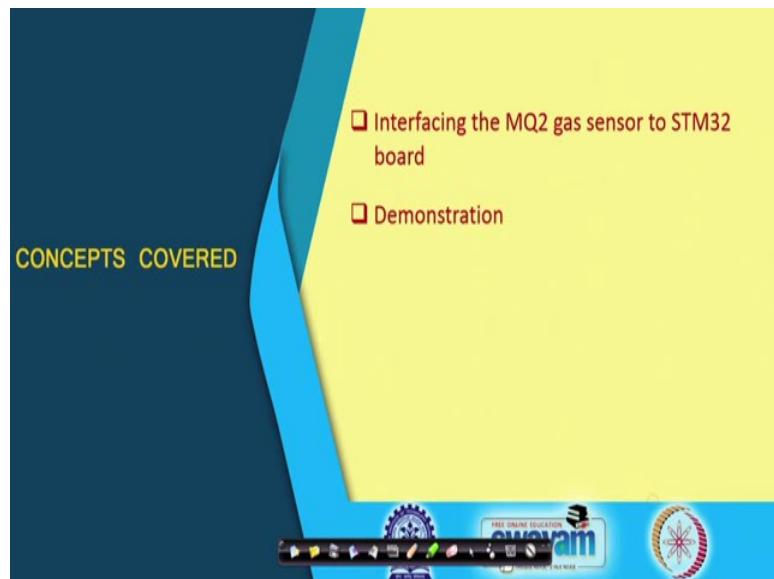
Thank you.

Embedded System Design with ARM
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 41
Experiment with Gas Sensor

Welcome to lecture 41. In this lecture, we will be showing you an experiment with a sensor which is a gas sensor. In this experiment, we will read the sensor values and depending on that an alarm system will be activated.

(Refer Slide Time: 01:05)



Firstly, I will show you how I will be interfacing the MQ2 gas sensor to the STM32 board, and then I will demonstrate the experiment.

(Refer Slide Time: 01:22)

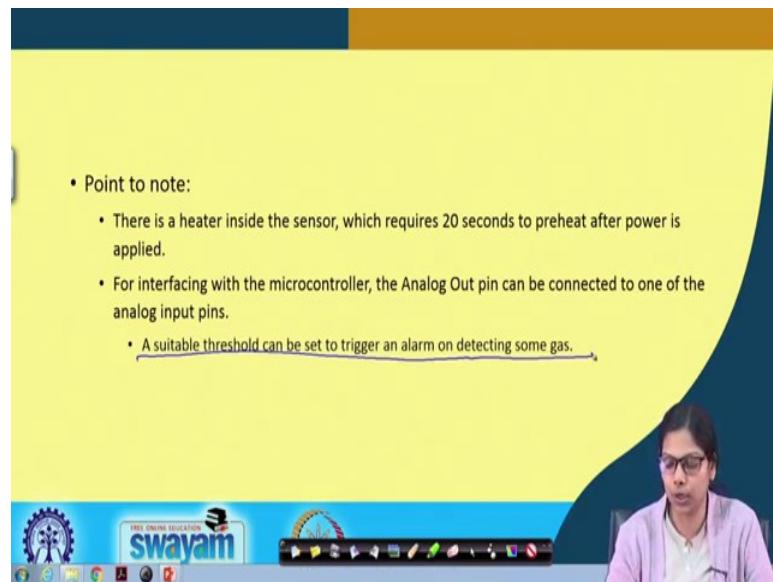
About the MQ2 Gas Sensor

- The MQ-2 gas sensor is most suited to measure the presence of the following gases:
 - Methane, Butane, LPG, Smoke
- Pin configuration:
 - Pin 1: Power supply voltage (typically +5V)
 - Pin 2: Ground connection
 - Pin 3: Digital Out – We can get a digital output from this pin, by setting a threshold value using the potentiometer.
 - Pin 4: Analog Out – This pin outputs 0-5V analog voltage based on the intensity of the gas.

Firstly about the MQ2 gas sensor it looks like somewhat like this. You can see there is a Vcc, GND, analog out, and a digital out. MQ2 gas sensor is most suited to measure the presence of the following gases. It can detect gases like methane, butane, LPG or smoke.

The pin configuration goes like this. The first pin is the power supply voltage Vcc which is typically 5 volt, then the GND connection. Then there is a digital out pin. In this pin, we can get a digital value by setting the threshold value using the potentiometer. We have already seen the use of potentiometer in LCD display. Here also if you want to use the this digital out value, then we have to use the potentiometer to adjust this threshold value. The analog out pin outputs 0 to 5 volt analog voltage based on the intensity of the gas, which will be connected to an analog port.

(Refer Slide Time: 03:35)



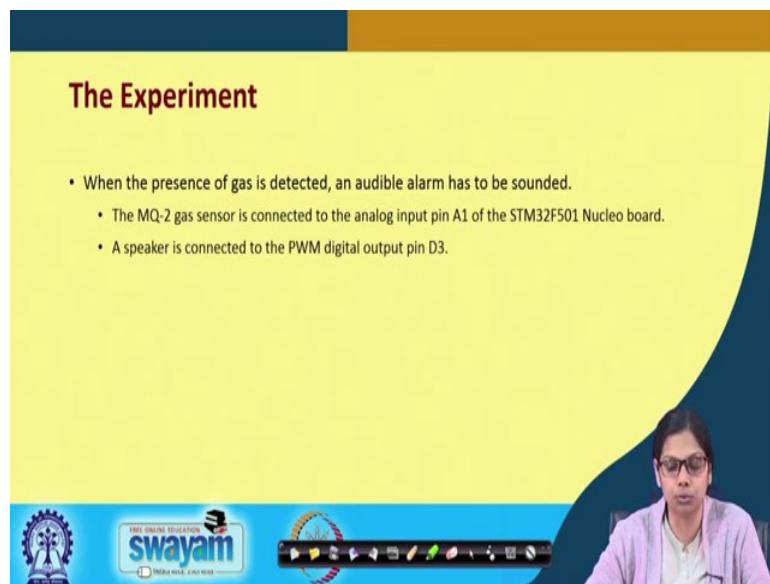
There are certain points to be noted that in that MQ2 gas sensor there is a heater inside the sensor. And it requires some time to preheat after the power is applied. So, when we start the system, it might require some time to initiate or to preheat the particular sensor, so that it can work properly. For interfacing with the microcontroller, the analog out pin can be connected to one of the analog input pins, so that we have already done for most of the experiments. The same way we will be doing that.

And a suitable threshold can be set to trigger the alarm on detecting some gas. Now, this is something you have to think upon like at what level you want your system to get activated, either it can be an alarm system or you can connect a GSM that will directly send a message to fire brigade. But generally the level of the smoke you can understand, you have to again calibrate, you have to do something such that smoke gets generated and then you have to see what value you are receiving through CoolTerm and accordingly you write your program.

(Refer Slide Time: 05:38)

The Experiment

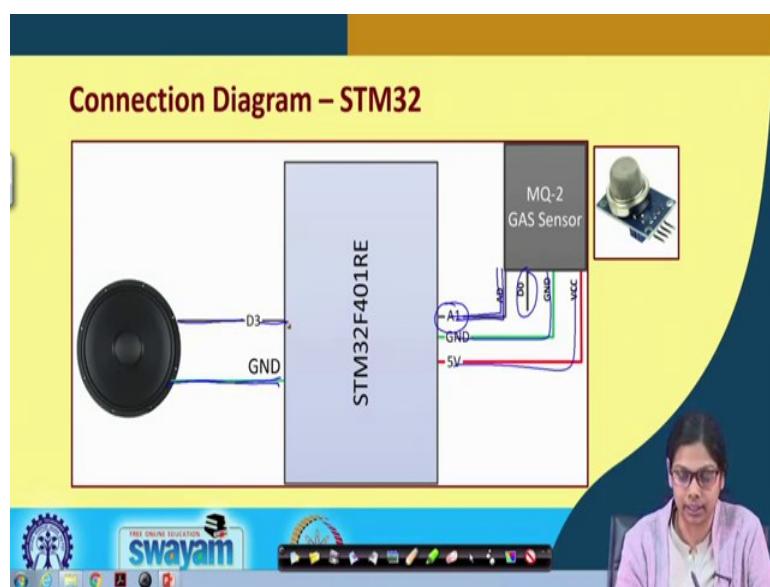
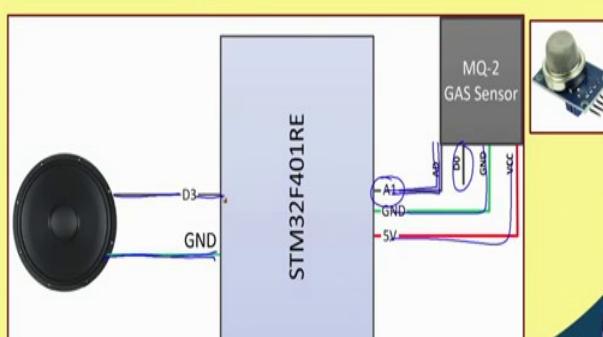
- When the presence of gas is detected, an audible alarm has to be sounded.
 - The MQ-2 gas sensor is connected to the analog input pin A1 of the STM32F501 Nucleo board.
 - A speaker is connected to the PWM digital output pin D3.



In this experiment what we will be doing is that when there is presence of gas, then an audible alarm will be activated. The MQ2 gas sensor is connected to the analog input pin A1 of STM board and a speaker is connected to the PWM output pin D3.

(Refer Slide Time: 06:13)

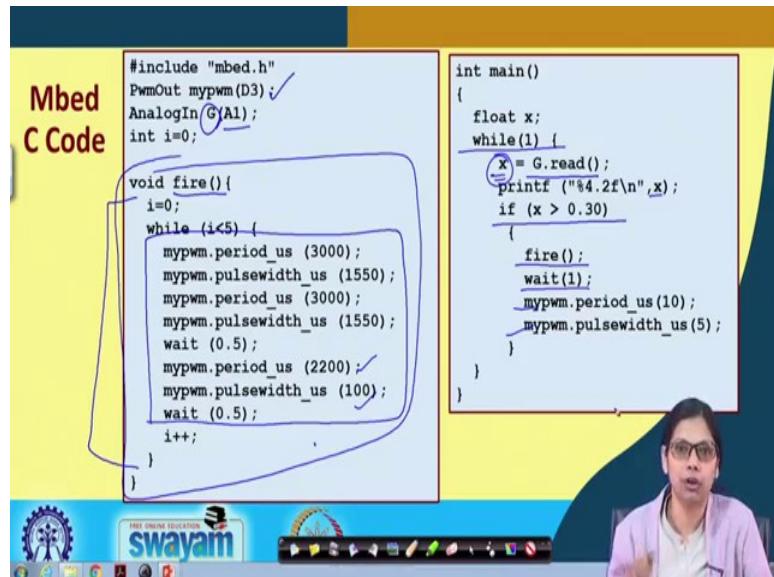
Connection Diagram – STM32



This is the connection diagram which is fairly straightforward. We are not concerned about the digital pin, we are connecting the analog out to A1. And then with the speaker the alarm system when it will get activated depending on the percentage of gas present in

the in a room; we use this D3 pin. So, this is the simple connection diagram that is required.

(Refer Slide Time: 07:00)



Mbed C Code

```
#include "mbed.h"
PwmOut mypwm(D3);
AnalogIn G(A1);
int i=0;

void fire(){
    i=0;
    while (i<5) {
        mypwm.period_us (3000);
        mypwm.pulsewidth_us (1550);
        mypwm.period_us (3000);
        mypwm.pulsewidth_us (1550);
        wait (0.5);
        mypwm.period_us (2200);
        mypwm.pulsewidth_us (100);
        wait (0.5);
        i++;
    }
}

int main()
{
    float x;
    while(1) {
        x = G.read();
        printf ("%4.2f\n",x);
        if (x > 0.30)
        {
            fire();
            wait(1);
            mypwm.period_us(10);
            mypwm.pulsewidth_us(5);
        }
    }
}
```

Let us look into the Mbed C code. `mypwm.period_us` is 3000 in microsecond, and the pulse width is 1550, and wait for some time and again we are doing 2200 and pulse width of 100. So, basically if you see this void fire, fire is a function that we have written and this function is very similar to what we were doing for the speaker generating a siren like sound.

Prior to that these are again straightforward things. This is PWMOut `mypwm` is for connecting with the speaker and `AnalogIn G (A1)` is for the gas sensor connecting to analog port A1. This is all about the connection and this part of the code is for the fire alarm. In the main function, `while(1)` means this will be repeated.

`x = G.read()` reads the gas sensor value. If the value is greater than 0.30, then we are calling this alarm system. We wait for 1 second and we again set some period and this goes on. If it is again greater than 0.30, again the alarm system will be put on. So, this is a sample code that we have written for interfacing gas sensor along with the alarm system using the speaker.

So, we have come to the end of lecture 41. And this actually ends formally the experiments that we have shown you in this particular course. So, now I will be showing

you how I will actually demonstrate you the interfacing part with the gas sensor which I have talked about just now.

In this experiment, I will be connecting a smoke sensor along with STM board. And what the smoke sensor will do, whenever it senses smoke in the surrounding, it will make an alarm. But you can also do something like this; whenever smoke is sensed an SMS should go to the fire brigade. In that interfacing what you have to do is apart from connecting this gas sensor, you also will be requiring a GSM board.

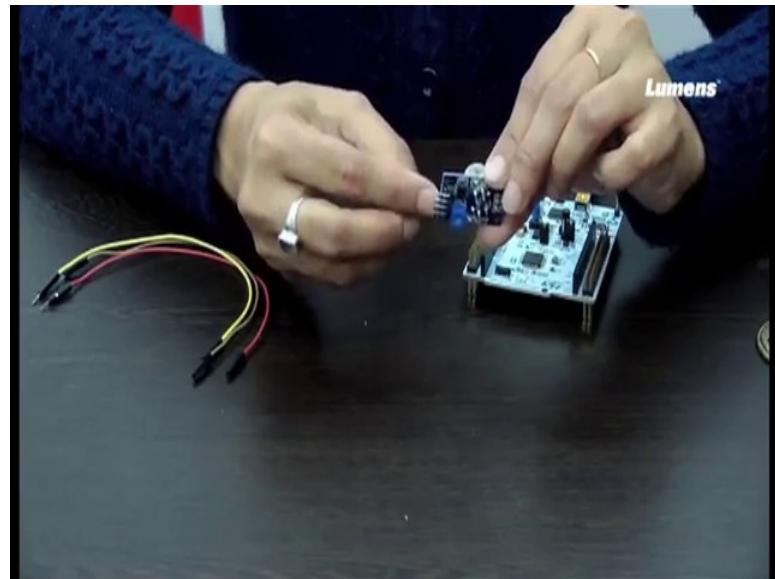
So, we can do many possible things, but here what we are doing essentially is will be interfacing this smoke sensor. And whenever there is a smoke inside this room, the speaker will make a siren kind of sound. Let us see how do I interface this gas sensor along with the speaker and with a STM board.

(Refer Slide Time: 12:25)



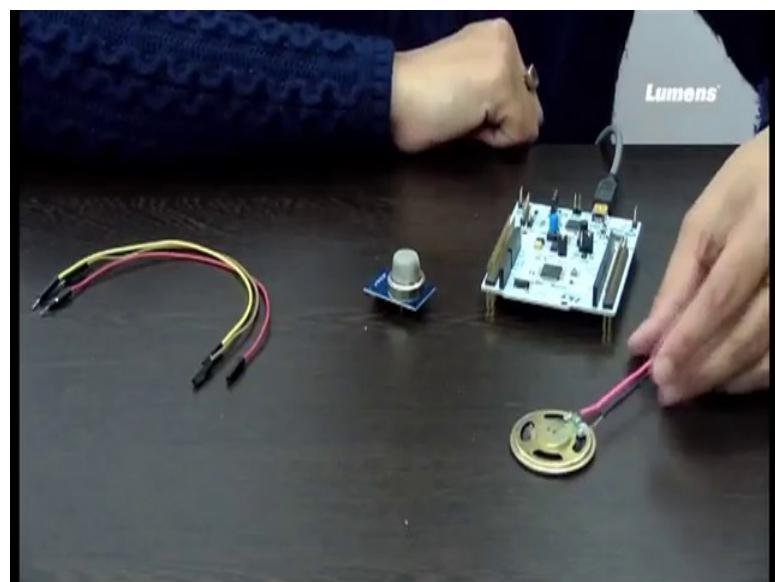
So, this is the gas sensor.

(Refer Slide Time: 12:34)



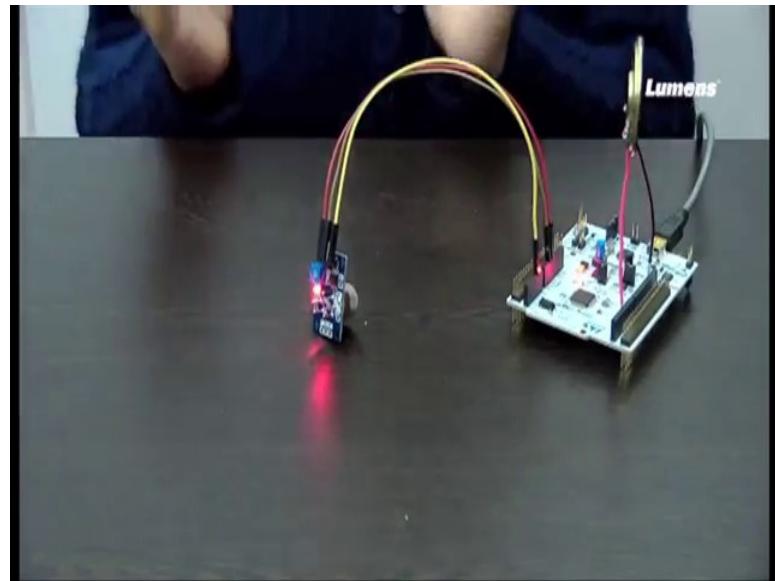
This is the gas sensor that we will be using. You can see there are four pins that are connected. So, what are those pins? One is AD, another is DD, another is GND and another is Vcc. I will be connecting the GND and Vcc, and I will be taking the analog output. So, I am not concerned about the digital output. I will be taking the analog output and I will be connecting it to pin A1 of my STM board.

(Refer Slide Time: 13:23)



The speaker I will be connecting one to ground and another to one of the PWM port. In this case we have used D3 port as the PWM port.

(Refer Slide Time: 13:41)



So, first one is Vcc. The next one is GND, and then the last one is the analog output that I will be connecting with the A1 pin of STM. This is a simple connection with the smoke sensor that I have done. Now, I will be connecting this speaker with D3 and one with GND which is the fourth pin from this side. So, this is all about the connection that we have made ok. The code I have already discussed with you. I will now dump the code.

Now, I will use a matchstick. I will just blow this matchstick and use the smoke of this. Whenever it senses the smoke, the siren will be on for sometime. I will just repeat this once more. It should receive a good amount of smoke, then only it will sense.

In an home automation system, you can always have for security purpose, this smoke sensor integrated. So, this is all about the experiment with the smoke sensor. We will move on and we will see that what all experiments we can do next.

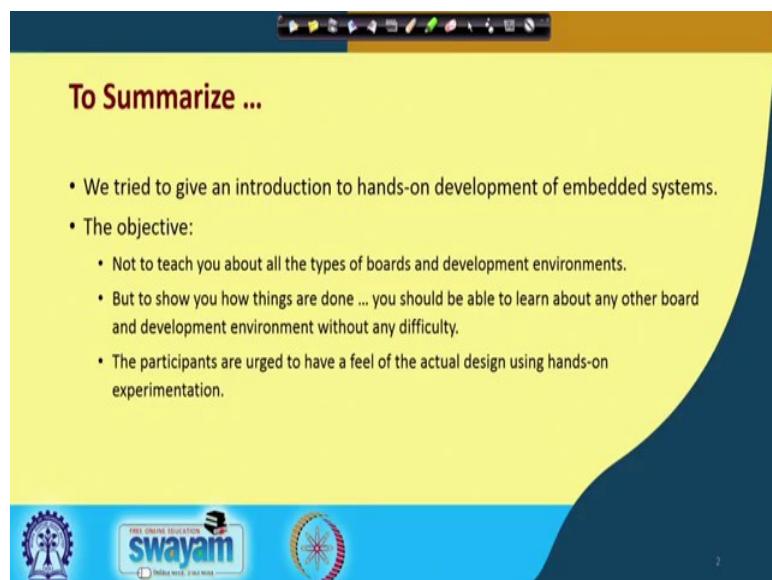
Thank you.

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

Lecture – 42
Summarization of the Course

So, we have come to the end of this course on Embedded System Design with ARM. We sincerely hope you have enjoyed this course. Let me tell a few things about this course before we say goodbye to you.

(Refer Slide Time: 00:41)



In this course, we have tried to give you a broad introduction to hands-on development of embedded system design. As we have seen we have talked very little about theoretical aspects on the subject. There are other courses, there are very nice textbooks on the subject, which dwell with the underlying theory behind the design of embedded systems. How do carry out various kinds of trade off, how to design software, how to design hardware, how do you carry out something called hardware-software code design, etc. but we have tried to give you an introduction to hands-on developmental aspects.

The objective of this course was not to teach you about all the kinds of boards that are available in the market, to tell you about all the kinds of software development systems which are available, but rather we have taken a couple of examples only. We have talked about the STM32 board, we have talked about the Arduino UNO board that are very

popular development boards. And with the help of those we tried to demonstrate that it is quite a simple task to interface devices and build systems that work according to some requirements.

Designing such an embedded system is not at all a difficult task, it is very simple in fact. We hope that the participants were with us, they were also involved in actual hands-on demonstration with some of the boards that were available to them, because without hands-on sessions, you really cannot appreciate the beauty and simplicity of this process. There was a time when hardware design was really very cumbersome, but today people are talking about open-source hardware design. You buy these modules from the market, these are really trivial to interface with the boards as I have already shown. And it is no longer necessary to write programs in assembly language or machine language, you can write in a high level language like C that we have seen.

(Refer Slide Time: 03:17)

Why learn about embedded systems?

- They are coming in a big way.
 - Shall be influencing the way we live, we communicate, etc.
 - Internet of Things is going to be the next technological revolution.
- We should not remain silent spectators.
 - Everyone should be equipped to contribute to this revolution.
 - Knowing the technology is important.
 - Tomorrow's gadgets are likely to be more intelligent, and programmable.

Now the question is that why do we need to learn about embedded systems? The first and foremost motivation is that embedded systems are coming to us in a really big way. We have already talked about internet of things, they are spreading like wildfire across every aspect of a life, there are places where they are really spreading very fast. These developments shall be influencing the way we live, we communicate in our daily life.

The point is that we should not remain silent spectators in this revolution, that is why the objective of this course was to try and give you a flavour of the developmental aspects of

embedded system. Like, what is the state of technology with which you can develop such a system today, because tomorrow's gadgets and devices will be quite different from what we see today. They will mostly be programmable and for that some knowledge about technology is very important.

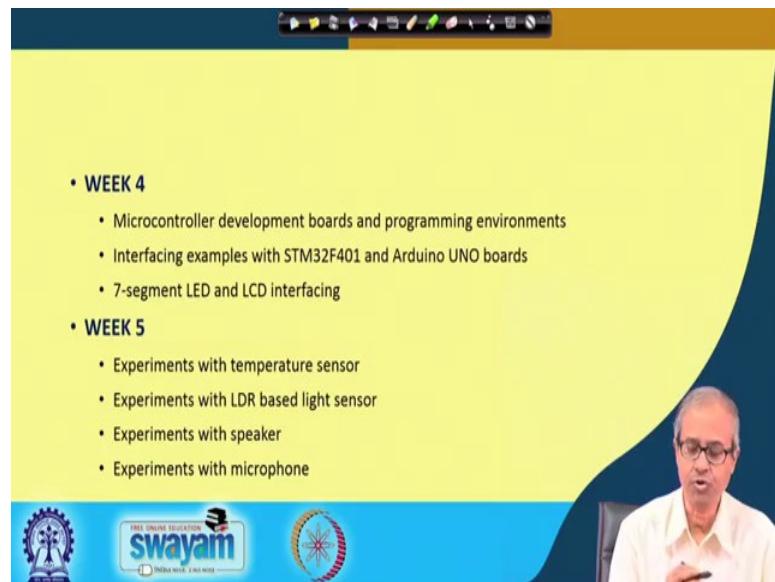
(Refer Slide Time: 04:45)

Course Coverage – Just to Recall

- **WEEK 1**
 - Introduction to embedded systems
 - Introduction to ARM architecture
- **WEEK 2**
 - Introduction to ARM instruction set
 - A case study with the STM32F401 Nucleo board
- **WEEK 3**
 - Data conversion techniques (digital and analog)
 - Output devices, sensors and actuators

Very quickly let us go through what we have covered over the weeks. During the 1st week, we had talked about some introductory aspects about embedded system design, various tradeoffs and also we talked about the architecture of the ARM microcontrollers. During the 2nd week, we talked about some interesting aspects about the ARM instruction set, some innovations that were incorporated therein, and we took as a case study the STM32 board. And in the 3rd week, we talked about the A/D converter and D/A converter designs and various output devices, sensors and actuators. Most of them we actually showed through hands-on demonstration sessions subsequently.

(Refer Slide Time: 05:49)



• **WEEK 4**

- Microcontroller development boards and programming environments
- Interfacing examples with STM32F401 and Arduino UNO boards
- 7-segment LED and LCD interfacing

• **WEEK 5**

- Experiments with temperature sensor
- Experiments with LDR based light sensor
- Experiments with speaker
- Experiments with microphone

FREE ONLINE EDUCATION
swayam
India Meets Knowledge

A man with glasses and a white shirt is speaking on the right side of the screen.

In the 4th week, we talked about microcontroller development boards and the software development or programming environments. Specifically, we saw the interfacing examples with two boards, the STM32 and the Arduino UNO. And specifically we had detailed discussions on 7-segment LED display and LCD display interfacing. During the 5th week, we talked about experiments with various kinds of sensors like temperature sensors LM35, light dependent resistors (LDR), speakers and microphones.

(Refer Slide Time: 06:31)



• **WEEK 6**

- Introduction to control system design
- Experiments with relay control
- Experiments with motor interfacing

• **WEEK 7**

- Introduction to IoT systems
- Experiments on home automation using GSM communication
- Experiments on simple alarm system based on touch sensing

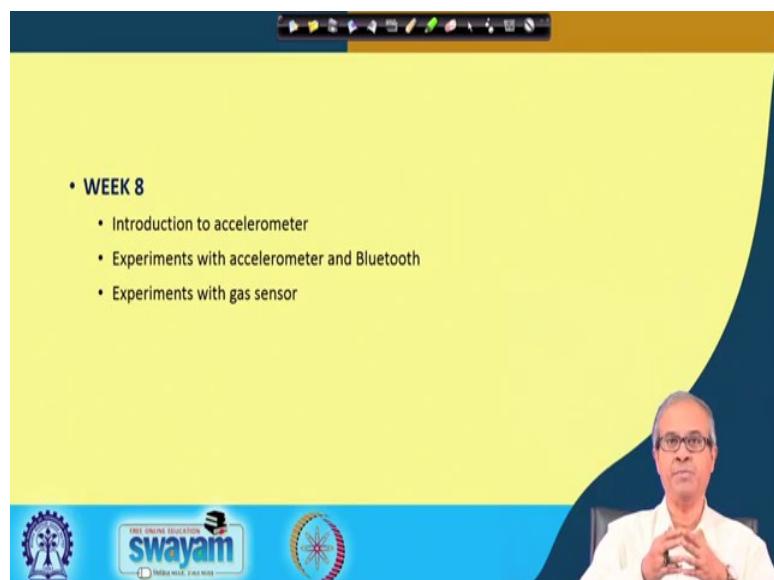
FREE ONLINE EDUCATION
swayam
India Meets Knowledge

A man with glasses and a white shirt is speaking on the right side of the screen.

In the 6th week, we very briefly talked about the design of control systems, various kinds of control algorithms – proportional, integral, on-off control etc. Then we saw some experiments using relays and also how we can interface DC motor to the microcontroller, speed sensing of the motor, and so on. In the process, we also learnt how to use the interrupt system of the STM32 board. We saw that it was really a very simple task for developing interrupt based software in this kind of environment.

And in week 7, there was a very brief discussion about internet of things (IoTs), then there were some experiments on home automation. Here we also learnt about various communication techniques, GSM for instance. And also we talked about some simple alarm systems using some kind of touch sensors, and so on.

(Refer Slide Time: 07:41)



And in the last week, we saw how an accelerometer works, what are its various applications. In particular we looked at some experiments where you also interfaced such an accelerometer device with Bluetooth technology, where we paired with a mobile phone and some messages were displayed on the mobile phone screens. And lastly we had some experiments with gas sensors, which is also very important kind of a sensor for many applications.

(Refer Slide Time: 08:19)

- We sincerely hope that the course was useful to the participants.
- We would like to acknowledge the support of:
 - The teaching assistants (TAs) associated with the course.
 - The participants who interacted with us through the discussion forum, making it an enlightening experience for all of us.
 - The entire NPTEL team at Centre of Educational Technology, IIT Kharagpur, and IIT Madras.

Before we end this course and say goodbye to all of you, myself and also Dr. Kamalika Datta who was my co instructor in this course, sincerely hope that the course was useful to you. Also it would be very proper to acknowledge the support of various people without which this course recording, the regular day-to-day running, etc. would not have been possible.

Firstly, we acknowledge the support of the teaching assistants (TAs), who were associated with the course throughout. The way you interacted with us through the discussion forum, it was really a very rewarding and enlightening experience for all of us. And lastly the entire NPTEL team at the Center of Education Technology here at IIT Kharagpur and also at IIT Madras, who were actually running this whole show. They have been actually enabling this technology to reach all of you, helping in sharing the knowledge, and making the experience really beautiful and rewarding. So, with these few words, we thank you once again, and we hope to see you again in the future.

Thank you.



**THIS BOOK IS NOT FOR SALE
NOR COMMERCIAL USE**



(044) 2257 5905/08

|



nptel.ac.in

|



swayam.gov.in