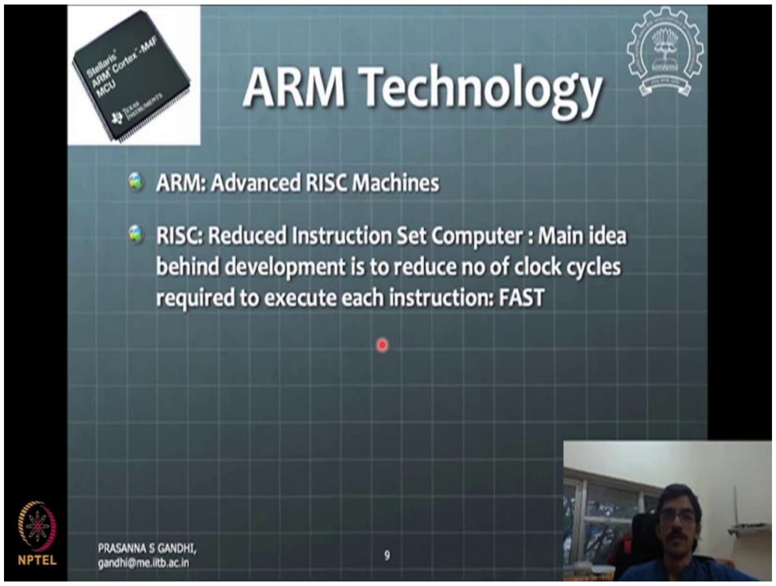
Design of Mechatronic Systems
Professor Prasanna S. Gandhi
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Lecture 18
Microcontroller Architecture ll

(Refer Slide Time: 00:11)

Lecture No. 18

Microcontroller Architecture II



ARM Technology

- ARM: Advanced RISC Machines
- RISC: Reduced Instruction Set Computer : Main idea behind development is to reduce no of clock cycles required to execute each instruction: FAST

PRASANNA S GANDHI, gandhi@me.iitb.ac.in

9

So, let us understand a little bit more about this ARM technology now. So, this Texas Instruments has developed this ARM Cortex M4 series MCUs which are quite popular for lot of industrial usage and applications. Some of the usages if you go to the datasheet, you will find they are gaming interfaces and lot of mobile chips, we may have additional chips as these controller chips and things like that.

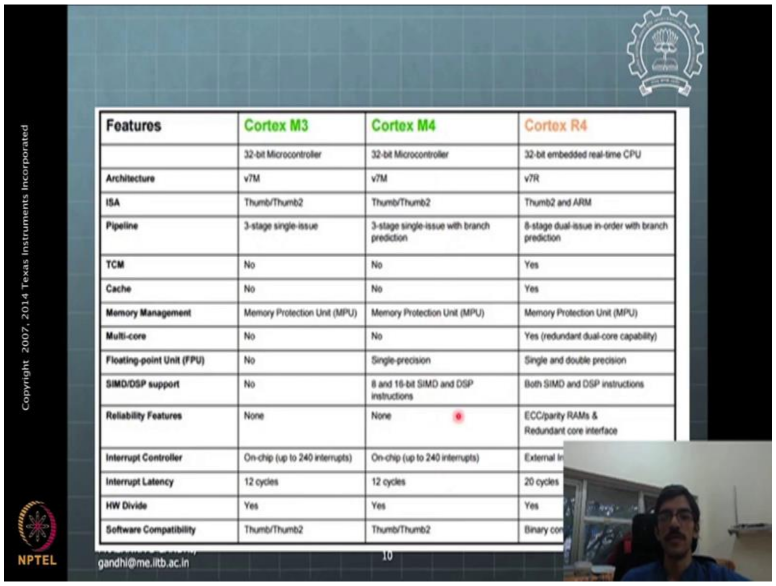
So, there are many-many devices in which you will find this and in the application note it will give this more details, maybe, or it maybe not now, but (maybe) we can go through some of these details in the datasheet, but maybe we can get first overview of this technology itself and then maybe we can go into little more details of the datasheets of this, let us do that first not jump into, so this ARM technology has lot of these different you can just appreciate for now that this technology has lot of families of microcontrollers, which are used in many-many different industrial applications and devices.

And the main crux of this technology is this ARM stands for this advanced-RISC-machine, RISC stands for reduced-instruction-set-computer. So, this is reduced instruction set computer is having the main idea or main philosophy, here, to develop this kind of a technology is to reduce the number of clock cycles required to execute whatever instruction needs to be executed, each of the instructions that you execute, say you want to move some contents of one register to other register, it takes some finite number of clock cycles.

Now, these clock cycles determine, how much is time taken for execution of the command and clock frequency would determine how fast that will happen. So, if you have 80-megahertz clock which is there in Tiva, and say you are taking 15 clock cycles to execute some command, so you can find out what is the time taken to execute that competitive command. So, as compared to the normal microcontrollers where this amplifier is not there and when this ARM technology is there each of these commands whatever are designed in this ARM technology, they will have the reduced number of clock cycles to execute.

So, that is why this, they can do really fast execution of your hardware programs. So, that is a main advantage of this ARM technology, and it is some industry standard now for a lot of industrial machines, mechatronic systems, people are using this ARM based controller. And the other thing is that the programs that you write here, even if now the chip is upgraded here, you will execute those programs with not much of a modification. So, here again the programming is also based on the base C-level programming. So, we will see what ways, how do you go ahead and program these.

(Refer Slide Time: 04:39)



Features	Cortex M3	Cortex M4	Cortex R4
	32-bit Microcontroller	32-bit Microcontroller	32-bit embedded real-time CPU
Architecture	v7M	v7M	v7R
ISA	Thumb/Thumb2	Thumb/Thumb2	Thumb2 and ARM
Pipeline	3-stage single-issue	3-stage single-issue with branch prediction	8-stage dual-issue in-order with branch prediction
TCM	No	No	Yes
Cache	No	No	Yes
Memory Management	Memory Protection Unit (MPU)	Memory Protection Unit (MPU)	Memory Protection Unit (MPU)
Multi-core	No	No	Yes (redundant dual-core capability)
Floating-point Unit (FPU)	No	Single-precision	Single and double precision
SIMD/DSP support	No	8 and 16-bit SIMD and DSP instructions	Both SIMD and DSP instructions
Reliability Features	None	None	ECC/parity RAMs & Redundant core interface
Interrupt Controller	On-chip (up to 240 interrupts)	On-chip (up to 240 interrupts)	External In
Interrupt Latency	12 cycles	12 cycles	20 cycles
HW Divide	Yes	Yes	Yes
Software Compatibility	Thumb/Thumb2	Thumb/Thumb2	Binary co

Copyright 2007, 2014 Texas Instruments Incorporated

gandhi@me.iitb.ac.in

10

So, let us move on to these, these are different, different technologies and corresponding features of the technology. So, this is a Cortex M4 technology and now more/upgraded model is Cortex R4 technology. So, there is some more details are given, so this is what this architecture, so this is Version 7, v7M architecture, so v7R some differences are given, we may not understand everything at these stages, it is okay, but one can see there are some things that are there in the advanced, say for example, cache memory is there in this Cortex R4 series, but not during this earlier series.

So, as people use this and then they demand the manufacture, we want some kind of this feature, people start developing, the other companies start developing the next versions of the microcontroller and that is how the evolution of these different microcontrollers takes place. The way you see the evolution that has happened in your computer systems.

Similarly, there are this kind of evolution that happened keep on, some developments keep on happening. So, the microcontroller that we will be using or we will be looking at is this Cortex M4 series microcontroller. And it has this 32-bit kind of architecture, 32-bit data processing will be there.

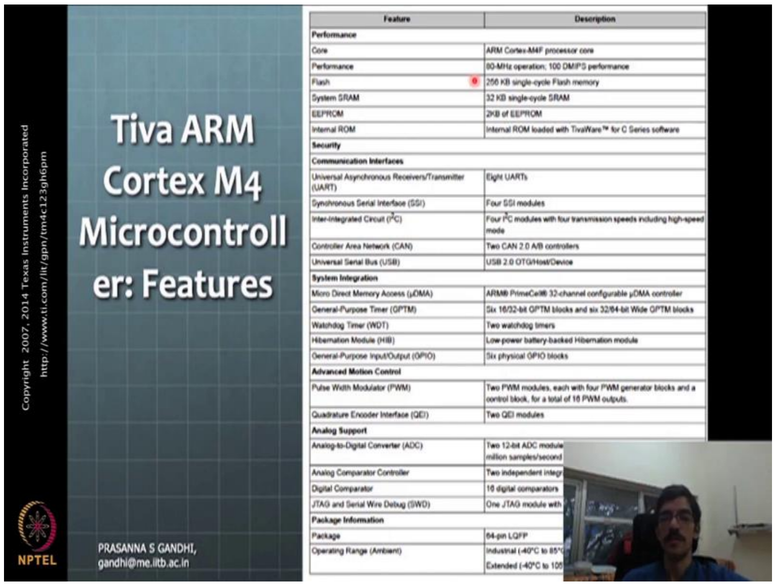
So, can you look at some of these details, some details I may also not be able to tell you, details I will read more about it and I will be able to tell you, but you can try to find out and see if we can understand some of the parts at least.

Say, for example interrupt latency here is 12 cycles, here it is 20 cycles, it will have some delay in the interrupt coming in execution. So, this is increased some advanced processing, something like that maybe there, which is I do not think this is a good thing to happen, I mean we want interrupts to get executed very fast. But anyway, so as I just said, these are the features that are given and some advanced features will be there in the higher series. From mechatronics perspective we may or may not need to get into too many details about these anyway.

So, unless very specific application comes, we should be able to dig into that and see, otherwise we probably from mechatronics engineers, we do not need to really bother too much about. See, this floating-point unit is there inside this Cortex. So, this is one can appreciate that you have this floating-point built-in unit and single and double precisions are also there.

So, these are more precise calculations can be done with Cortex R4 series than M series, fundamentally. So, even if you do not know in the specific datasheet of the microprocessor, microcontroller, as long as you know that it belongs to this Cortex M4 series is fine, you know that it cannot do the double precision as a built-in in the future in the microcontroller, one has to write a separate program, that means one has to expand more number of clock cycles, to execute double precision in the single precision inherent microcontroller platform. So, those are things one can see from mechatronics perspective.

(Refer Slide Time: 08:52)



Copyright 2007, 2014 Texas Instruments Incorporated
http://www.ti.com/lit/gpn/tm4c123gh6pm

Tiva ARM Cortex M4 Microcontroller: Features

Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I^2C)	Four I^2C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	Two CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 OTG/Host/Device
System Integration	
Micro Direct Memory Access (μDMA)	ARM® PrimeCell® 32-channel configurable μDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Advanced Motion Control	
Pulse Width Modulator (PWM)	Two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs.
Quadrature Encoder Interface (QEI)	Two QEI modules
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC module million samples/second
Analog Comparator Controller	Two independent integr
Digital Comparator	16 digital comparators
JTAG and Serial Wire Debug (SWD)	One JTAG module with
Package Information	
Package	64-pin LQFP
Operating Range (Ambient)	Industrial (-40°C to 85°C Extended (-40°C to 105

NPTEL

PRASANNA S GANDHI,
gandhi@me.iitb.ac.in

So, now if you come to this Tiva Cortex microcontroller, its features, if you see in the datasheet are given in this format. And then there will be some block diagram to explain these more in detail. So, you have this 80-megahertz operation, so these 80-megahertz mean your clock is running at that frequency and then depending upon what is the length of your program or number of clock cycles that your program may take you can have a sampling time which is more or less based on this, this can determine your sampling time actually.

Then, some memory details are there, and there are communication details, what for mechatronics system perspective is more important for us is this advanced motion control. So, they have these PWM interfaces, 2 modules which with 4 PWM generators where there are 8 PWMs in this microcontroller and for total 16 PWM outputs. And so, 2 PWM modules each with 4 PWM generator blocks, so actually it should be 8, I do not know why they calling these 16 PWM outputs, so there may be something more we need to see in more details here.

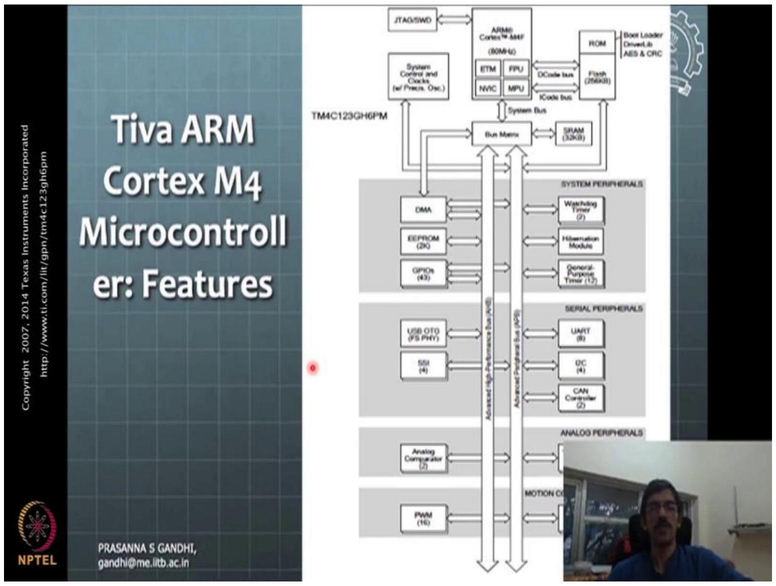
Then, there are these two quadrature encoder interfaces modules, the encoder which is having A and B pulses, so that encoder can be now directly connected to this quadrature encoder interface, you remember why it is called quadrature, because it gives this A and B pulses which are phase shifted by 90 degree and the combination of that gives over one cycle four different pulses can be possible.

So, 4 first quad, so that is why it is called quadrature, 4 different pulses you get over one cycle, because of this cascade of two A and B signals shifted by 90 degrees. So, that phase difference is what gives you the direction also. So, that is how this entire thing, how to read these pulses and convert them into count and increment the count if its direction positive and decrement the count if its direction negative, everything is done by this dedicated interface.

So, you do not need to do any of the operations of this kind to get the number for the encoder count into your microcontroller system that will be directly given by this, if you configure in this interface appropriately, you will be directly getting the output ready. And that is what we will do, we will be sending you this motor and along with its encoder, so we will interface this encoder with Tiva and see how it can give these counts, like that all we will do as a part of the lab.

So, we will focus on this motion control part and then some part of analog inputs. And so, these other things are useful package and other things will be useful from, if you want to now build a dedicated embedded system board for your application, which we will not have opportunity to do in this course. We will have to do a PCB manufacturing for that and it is not part of this course.

(Refer Slide Time: 12:51)



Tiva ARM Cortex M4 Microcontroller: Features

Copyright 2007, 2014 Texas Instruments Incorporated
http://www.ti.com/lit/gpn/tm4c123gh6pm

PRASANNA S GANDHI, gandhi@me.iitb.ac.in

So, we will move on, from here, this is more details about the diagram. So, this JTAG refers to a communication interface, which is typically used for downloading the C level program compiled version to the microcontroller memory and exchange the variables from the memory to this software program in which you will write the course, which in this case will be Code Composer Studio to view.

So, there will be some interface that is needed, which will share some memory locations to real time, what is happening to different registers, what is happening to different variables in what we, what you have used in program. They all will be exchanged to this interface, to your computer system to be displayed on your desktop.

So, that you will see, you run some program in microcontroller and you see that, when I execute this command, this happens to this register all the seen you can see on the on the thing. And also, you may be able to register the, say if you want to record some signal in time, you will be able to do that to such a kind of interface, so this is a JTAG or SWD those kinds of interface are there.

And then these are different, you can see this bus system is there and then these are memories that are interface, so some part it will have a unidirectional some part it has bi-directional memory. Then, there is some, so this is the interrupt interface, then you just read some part may make sense some part may not make sense, it is okay, does not matter. So, as, so part which then makes sense you will go through and read little more and things start making sense.

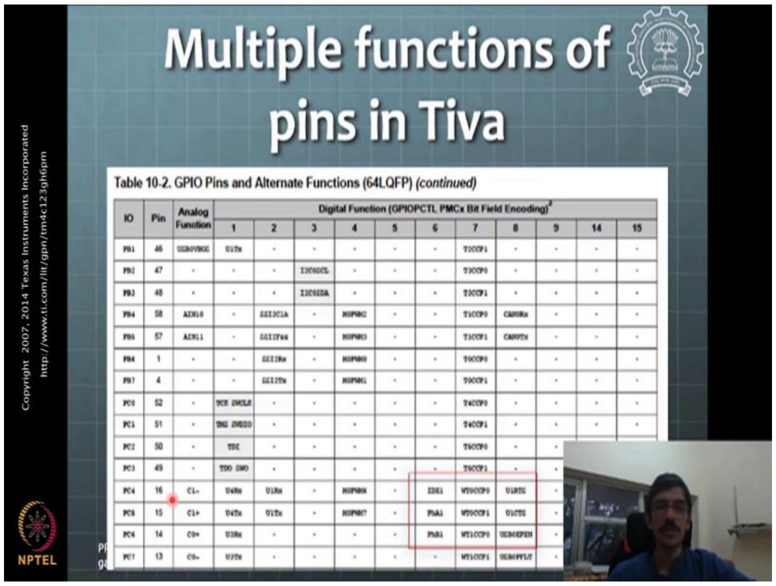
There is a ROM in which read only memory if you know, I am sure you know, it has some driver libraries. So, it has, see this ROM has boot loader, that means the system boots for this microcontroller through this ROM. When you start your computer system also you will have some small part of the program that get executed, that is written in the ROM, so that your system is at the, for the first time it starts booting from there.

And then it calls of other routines to execute and get the computer ready for you to work. The same way it happens in microcontroller, it starts off with execution of this commands in the ROM, and then the microcontroller is ready for your programming. The important part is that the driver library is already recorded into this ROM, so those libraries are not fetched and burned separately through these JTAG interface, they are taken and burned here every time the program execution takes place.

So, that saves you a lot of what you say programming time partly and also the execution time. So, the memory of your application will not have any of the library files there, because they are already coded into this ROM, and there will be call from the ROM to be used. And one of the other reasons things are fast in some sense in these microcontrollers. Especially the programming is fast, you will be finding very quickly this program gets loaded onto this microcontroller and now you are ready to execute.

When you start programming, you notice this difference as compared to what you are doing with XEP 100 or your Arduino or anything the previous thing, we will find that the program will get very fast downloaded onto this microcontroller platform Tiva platform.

(Refer Slide Time: 17:24)


Copyright 2007, 2014 Texas Instruments Incorporated
http://www.ti.com/lit/gpn/tm4c123gh6pm
NPTEL

Multiple functions of pins in Tiva

Table 10-2. GPIO Pins and Alternate Functions (64LQFP) *(continued)*

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PB1	46	USB0VBUS	U1Tx	-	-	-	-	-	T2CCP1	-	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-
PB4	58	AIN10	-	SSI2Clk	-	M0PWM2	-	-	T1CCP0	CAN0Rx	-	-	-
PB5	57	AIN11	-	SSI2Fss	-	M0PWM3	-	-	T1CCP1	CAN0Tx	-	-	-
PB6	1	-	-	SSI2Rx	-	M0PWM0	-	-	T0CCP0	-	-	-	-
PB7	4	-	-	SSI2Tx	-	M0PWM1	-	-	T0CCP1	-	-	-	-
PC0	52	-	TCK SWCLK	-	-	-	-	-	T4CCP0	-	-	-	-
PC1	51	-	TMS SWDIO	-	-	-	-	-	T4CCP1	-	-	-	-
PC2	50	-	TDI	-	-	-	-	-	T5CCP0	-	-	[illegible]	[illegible]
PC3	49	-	TDO SWO	-	-	-	-	-	T5CCP1	-	-	[illegible]	[illegible]
PC4	16	C1-	U4Rx	U1Rx	-	M0PWM6	-	IDX1	WT0CCP0	U1RTS	-	[illegible]	[illegible]
PC5	15	C1+	U4Tx	U1Tx	-	M0PWM7	-	PhA1	WT0CCP1	U1CTS	-	[illegible]	[illegible]
PC6	14	C0+	U3Rx	-	-	-	-	PhB1	WT1CCP0	USB0EPEN	-	[illegible]	[illegible]
PC7	13	C0-	U3Tx	-	-	-	-	-	WT1CCP1	USB0PFLT	-	[illegible]	[illegible]

Again, here also as I said for the way it was there for XEP 100, if you see the pins here, this is our GPIO pins, now this is general purpose input output or one can understand this as a digital input output interface. So, they have alternate functions, so this is important to know that okay pins can have alternate functions.

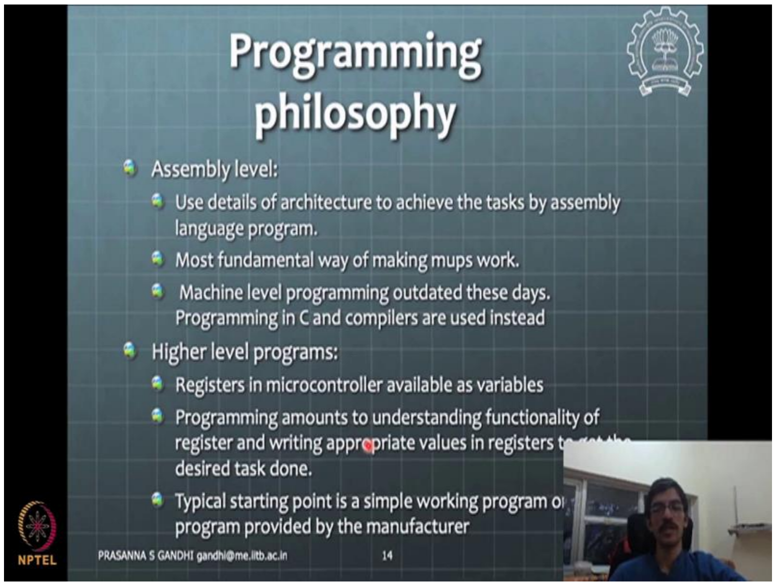
So, for example here if you see this PC4, PC5, PC6 these are the pins here, they have, they are shared between this Phase A1, Phase B1 and Index 1. So, these are the pins for the encoder interface, you remember that we had A phase, B phase and you had index pulse also. So, these pins, so when you interface your encoder, you will use these three pins or there are two encoder interfaces one of the interfaces is this some other interface you will find for two IDX2, PhA2, PhB2, like that you may find some other pins.

So, in this case they fall as PC, so these are the names of general-purpose input output, PC6, PC5 and PC4. These pins you will connect to your encoders and then you hope that okay you configure this appropriately. So, naturally then there will be some where you need to say that I am using these pins not as general-purpose input output, but I am using these pins as encoder outputs.

So, that configuration, reconfiguration of the pins needs to be done somewhere, there will be some commands to do that. So, this is how one can understand by looking at these pin diagrams or some other details, what it means when I start programming? In programming also, it will be given there, but if we know what we are looking for then it is much easier to find out what we are looking for and then figure out and go ahead with programming.

So, it is a skill, I would say to develop and that comes through your previous background of whatever microcontroller programming and you start programming a new microcontroller use some philosophical idea and then some of these cues from somewhere or some previous examples of some programming and you need to finally make sure that your application is running fine. So, that is exercise we all have to go through we will understand little more about this ARM Cortex technology.

(Refer Slide Time: 20:29)



Programming philosophy

- Assembly level:
  - Use details of architecture to achieve the tasks by assembly language program.
  - Most fundamental way of making mups work.
  - Machine level programming outdated these days. Programming in C and compilers are used instead
- Higher level programs:
  - Registers in microcontroller available as variables
  - Programming amounts to understanding functionality of register and writing appropriate values in registers t
  - desired task done.
  - Typical starting point is a simple working program o program provided by the manufacturer

PRASANNA S GANDHI gandhi@me.iitb.ac.in 14

So, this we will talk a little bit at a philosophical level about the programming, this is important and see this, see once you have this philosophy firm up in your mind, then actually what to expect what not to expect is very clear, so we do not need to teach learn a syntax, syntax is not a big deal to learn, so you might have, we might appreciate that in by now, you might have learned two, three computer languages C or some other thing.

And the syntax is not but ideas and philosophy or the way of doing things, or way of thinking, how do I create a logic around this that is what is more important. So, we need to shift that for microcontroller programming, the way you think in the domain of normal computer programming, C language programming, although there are a lot of similarities, but there are some certain distinct features that one needs to be very clear about.

So, we will not talk about much about assembly level program, we will not use it at all actually, but this is a most fundamental way to program stuff. So, you know all the registered details and like you say okay for this command to be executed, I know I need to move something from this register to that register and from then I need to do something, like that you need to know the sequence of operations to be carried out and you are programming each of the operations for just doing some simple tasks, that is too much of a treasury.

So, that nowadays is all done by the libraries, so libraries will do this task of whatever is to be done to get your interface working and your task is to now know the libraries, how to use those libraries and how do you make sense of different things that are there in the libraries or in the register level programming.

So, suppose we are doing register level programming, so see, one of the ways is the most basic way is assembly level programming. Then, little higher than that is in this register level programming, which you are, so in the assembly level you are actually writing assembly code, you are not using C.

So, actually C programs finally will convert the code into assembly level only, but you are directly writing that, but that is not the way just for the sake of completeness of this, but we will not use that. So, the next higher-level programming is where you have the registers and each of the registers you know their functionality, the functionality of how do know the functionality? Functionality will come from the datasheet, in the programming of XEP 100, for example.

So, for programming say digital input output interface, there is a register called DDR, data direction register, DDRX or DDR whatever ABCD there are different registers. Now, these registers you need to write something in those as some data and that will have some meaning and what data written has what meaning is given in the datasheet of the programming for that microcontroller. And that is how you initiate different registers to achieve different things, for example, in XEP 100 specifically to have those registers as output registers. So, I want A register as a port as an output port, then I will use DDRA to be equal to some FF number.

So, then all the pins of this port A will be configured as an output pins. In other cases, they can be configured as input pins as well. So, like that you write these as a C level commands, so these DDRA is equal to 0X FF is a command in the C that you will write and when it is compiled it will do all these jobs about whatever it needs to be done to, at assembly level to get make sure that this port A is configured as an output port.

So, these registers are called control registers, and then there are some other registers called data registers. So, you need to understand that functionality of this ratio, it is a control register or it is a data register and write appropriate values according to the datasheet in those registers and your programming is done. So, that is how you go ahead programming next level of program for microcontroller, where you have access, you are directly writing these registers in some sense.

And so, this is next level of programming where you are doing the register level programming, control registers and data registers separately you are initiating and doing some operation. So, some of you are familiar with XEP 100 it will be very easy thing to understand. For those who are not you will need to see those things and see the datasheet and then you can make sense of some of the things.

So, the idea here is not to understand everything from all the details, no. What we want to get a point here is that you have these registers names available as a variables and those variables are initiated to some kind of data values depending upon what is the functionality that is needed, that is one higher level of programming for microcontroller. And then there is yet another level of programming where you are now heavily dependent upon the libraries.

So, there are commands that are there in the libraries, that carry out the operation that you want to do and you have to supply those functions, some appropriate data as arguments. So, that is a way the Tiva based or ARM Cortex based microcontrollers will be programming in that way. So, we will get into more details about that as we go down, but for now maybe we will stop here.

We have some more details about the programming philosophy coming up next, different kinds of things that we need for, specifically for our mechatronics application from the microcontroller, what we should have in microcontroller, which we will use for mechatronics purposes. So, that is what will be coming up the next. And that is how we will move on. So, maybe we will stop here for now.