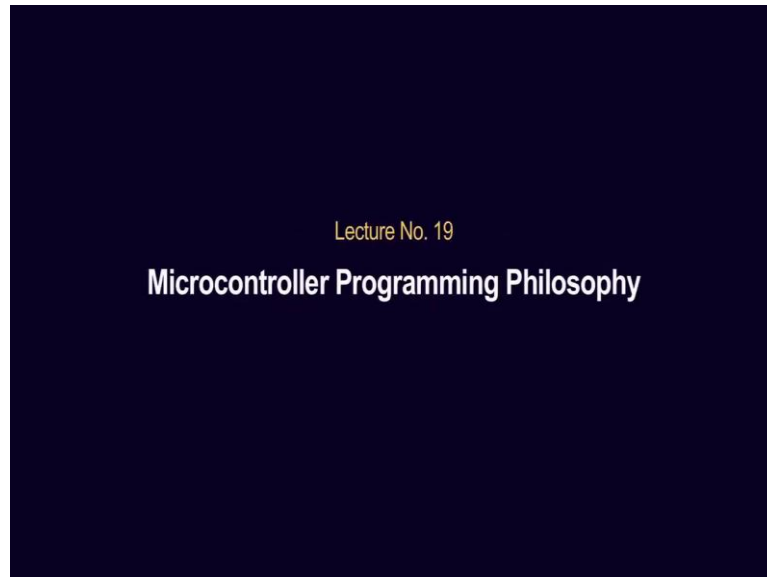


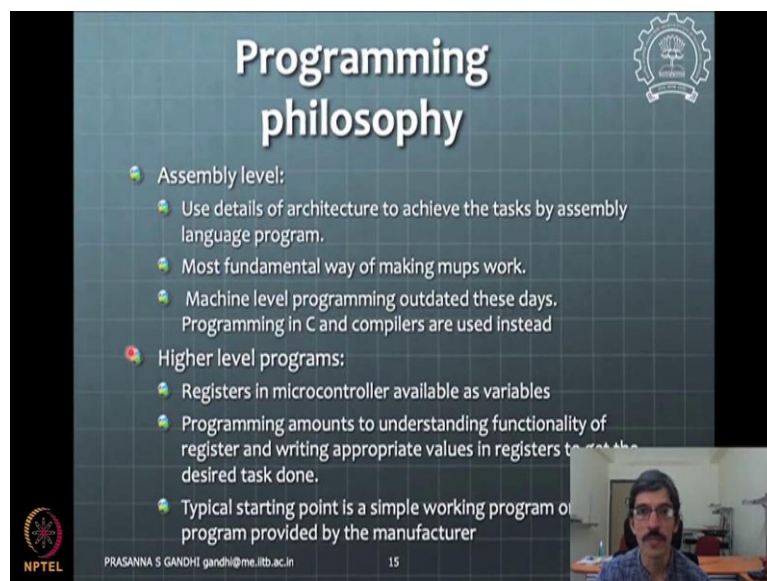
Design of Mechatronic Systems
Professor Prasanna S. Gandhi
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Lecture 19
Microcontroller Programming Philosophy

(Refer Slide Time: 00:11)



In today's class, we will start continuing further our discussion on microcontroller architecture and programming details. So, let's begin with the programming philosophy as we discussed in the last class also, this philosophy would center around registers.

(Refer Slide Time: 00:49)



So, we have seen there are different levels of programming and each one has like different ways to access different registers in the microcontroller platform. And these registers are

basically at their fundamental level, they are flip flops. So, you can see hardware has these different layers, one can understand the hardware in different layers, or and one can understand the software also will have like as we will see different levels of programming.

So, the base level is the assembly language where you can access each of the registers by directly assembly language. And we will see how you do that with some examples of some of the hardware of basic fundamental microprocessor 8085 system. And then, we will move on to like the more advanced system and see some examples of programming there and we will move to further advanced system, which is on ARM technology.

We have seen in the last class, general philosophical description or motivation, for which ARM technology was developed. And it has a lot of usages in the modern mechatronic systems. So, that is why I like it is good to study ARM kind of technology and its applications or its interfaces that it provides and things like that.

So, this way you will be able to develop your own mechatronic products in the future, if need be, very easy. And you know what is the state of the art that is happening. So, that is why we will work with this Tiva microcontroller and the Tiva microcontroller will give us lot of understanding about ARM technology!

(Refer Slide Time: 03:09)

8085 μ P Example assembly language

Programming 8085

- Instruction format (mnemonics)
 - Opcode
 - Operand
- Example

Opcode	Operand	Description
MOV	C, A	Move contents of register A to C
ADD	B	Add contents of B to accum
CMA		Invert each bit in acc

NPTEL PRASANNA S GANDHI gandhi@me.iitb.ac.in 16

Now, we will move on to little bit more details about programming. So, if you go to basic fundamental assembly level language, say 8085 system, the programming would have this something called opcode which is the instruction for, as in C you have some instructions like

that, you will have some instructions in the assembly level programming, which is given in some kind of MOV, ADD these are very short instructions are there.

This MOV as the name suggests it moves contents of register A to registers C, these are, A and C as we saw are registers in architecture of 8085 system somewhere. So, A is a special register called accumulator in that system. So, these are descriptions of these all things will be given in the datasheet and even these commands and their meaning will be given in the instruction manual for the microcontroller or microprocessor in this case.

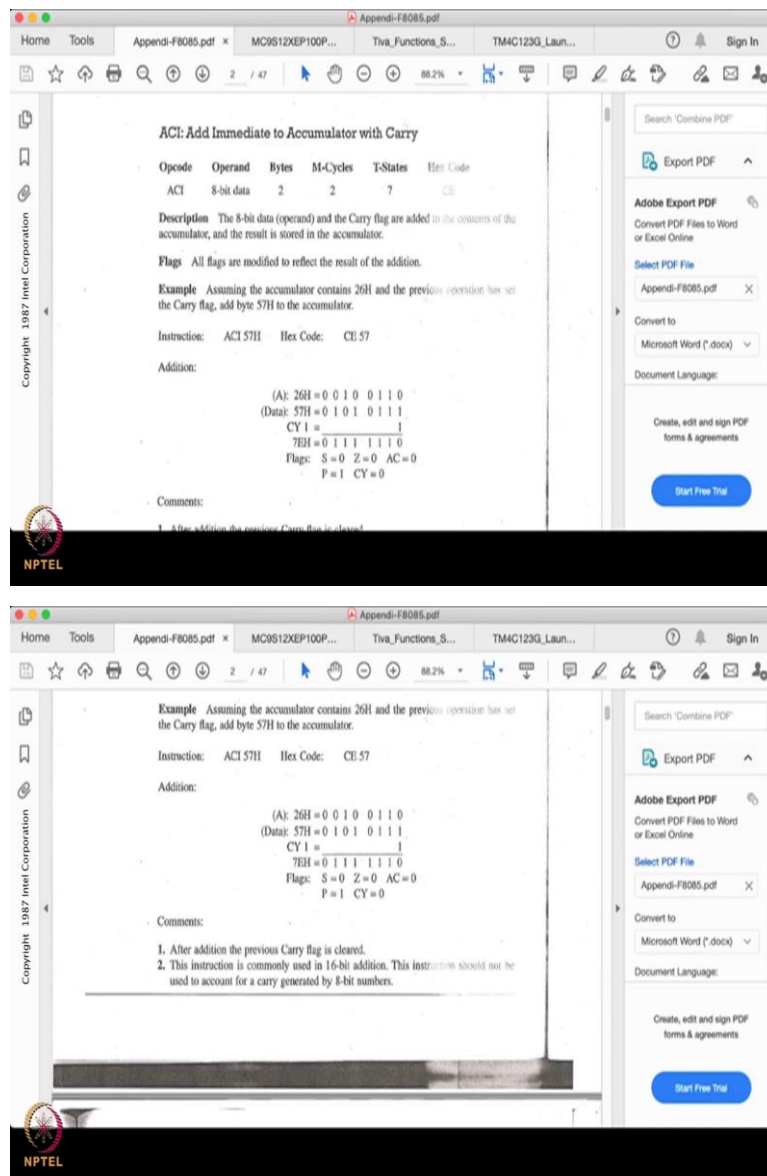
So, this is most fundamental level programming that one can do, where you actually write assembly language code, nobody does that these days actually. It is just for the sake of academic understanding that, oh look finally whenever we assemble our C code, it finally gets into this program and this program gets executed in its own way.

So, finally, each of these commands also will have some hex code associated with that. So, that is how that code will be loaded into the, into their respective registers. And then based on the timing and control circuitry that one-by-one lines of code will get executed and that is how likely your operation happens. So, this goes for fundamental level understanding of how things happen at a hardware level.

Say, for example, when I say moves contents of register A to register C, there are some hardware pins to be enabled and then needed to be made high. So, that contents of the register A are available on the data bus. And then, you enable some other pins so that the contents from the data bus are transferred to register C, these are micro instructions, which will be associated with each of these so-called opcodes.

And number of these micro instructions will determine how much time would take, how many clock cycles in particular would be required to execute this command MOV here. So, that is understanding here so for any command.

(Refer Slide Time: 06:02)



So, in the instruction set you can observe now that there are many different instructions that will be given. So, for example, this is instruction ACI. So, description is add immediate to accumulator with carry. This accumulator is this register A. So, if you have no understanding of architecture, then these commands will not make much sense.

But, if you have understanding that look this is accumulator and this command as this opcode and operand and then number of times state it takes is 7, or t-states cycles, number of cycles of the clock it will take to execute this command is 7 here. Like that this, all these details will be given and this instruction has hex code this CE 57.

So, olden days, you do not have this, the computer or keyboard to program these 8085 systems. So, what you do is, you will have this ADD display in which there are only like 8

fields. So, 4 are, 6 are, there are only 6 fields, there are 4 of them are address field, and 2 of them will be data field. So, you start at any address and you start coding these hex codes into these different memory locations. And that is how you start writing the program.

So, which was quite cumbersome to create right now, we have done that in our microprocessor course or lab that I was when I was MTech. student here, I did that in one of the labs actually. So, it is quite a time-consuming process. Also, in my B tech. project I did that actually. So, it is very interesting, now if I think about how this, so much of a drudgery to code this and then make sure that they execute properly and run your motors or any other applications.

(Refer Slide Time: 08:34)

CALL: Unconditional Subroutine Call

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CALL	16-bit address	3	5	18	CD

Description The program sequence is transferred to the address specified by the operand. Before the transfer, the address of the next instruction to CALL (the contents of the program counter) is pushed on the stack. The sequence of events is described in the example below.

Flags No flags are affected.

Example Write CALL instruction at memory location 2010H to call a subroutine located at 2050H. Explain the sequence of events when the stack pointer is at location 2099H.

Memory Address	Hex Code	Mnemonics
2010	CD	CALL 2050H
2011	50	
2012	20	

Note: See the difference between writing a 16-bit address as mnemonics and code. In the code, the low-order byte (50) is entered first, then the high-order byte (20) is entered.

Instruction	Flags	Description
CC	CY = 1	DC 2/9 (if condition is not true)
CNC	CY = 0	D4 5/18 (if condition is true)
CP	S = 0	F4 Note: If condition is not true it continues the sequence, and thus requires fewer T-states.
CM	S = 1	FC If condition is true it calls the subroutine, thus requires more T-states.
CPE	P = 1	EC
CPO	P = 0	E4
CZ	Z = 1	CC
CNZ	Z = 0	C4

Flags No flags are affected.

CMA: Complement Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CMA	None	1	1	4	2F

Description The contents of the accumulator are complemented.

Flags No flags are affected.

Example Complement the accumulator, which has data byte 89H.

Instruction: CMA Hex Code: 2F

Before instruction	After instruction
A [1 0 0 0 1 0 0 1] = 89H	A [0 1 1 1 0 1 1 0] = 76H

So, it will have all the instructions that are available for 8085 system to go, they will be all there in this. So, there are some conditional instructions, unconditional call will be there. There will be some if-else statements how do they execute that you will get to know say for column positive. So, if something is positive, then you will shift the address, next address which will be from where the programming execution will restart. Like that you will have some kind of nesting loops possibility and things like that, by using this basic fundamental level commands.

So, nowadays, all modern microcontrollers you just say if-else in C language and your job will be done. But in olden days there is no command if you will find in this whole set of instructions for this, there is no command- if or there is no command- while. So, there are these commands called call on positive, call on carry or call on 0. So, that means when the flag, you have seen the flag we studied in the last class, when the 0 flag and that flag register becomes 1, then the program some sub-routine will be called for the program and that is how this instruction works actually, call.

So, this is how olden days this is programming used to be, and that is what is done now also, but it is all the compilers to do that **targery**. So, human beings are spared from that. So, we as a programmer for hardware programming write all the codes in C language and they get executed. So, that is how things happen there.

So, main understanding here is that this 8085 system or any microcontroller system will ultimately have the program in the assembly language that will be downloaded to microcontroller and each of these instructions of course, they will have associated some hex code with that and given higher level program that you will write in C, the assembler will, or compiler will compile this program and generate these opcodes and then generate a code, which will be only with respect to these opcodes and that is a code which will be downloaded microcontroller memory that is how the things are going to happen.

And you will see later when we see the interface of Tiva microcontroller, we will be able to see that some of these opcodes or some of the assembly level parts of the code which are there in the microcontroller.

(Refer Slide Time: 11:55)

Fundamental Operations in microprocessor

- Internal data transfer operations
- Arithmetic operations
- Logical operations
- Branching operation
 - Microprocessor initiated: JNZ, JMP, JZ, JC commands in assembly language
 - Externally Initiated: Interrupts

Computing and interfacing using these operations: IMP from control implementation perspective

These are done in modern microcontrollers using C prog

NPTEL PRASANNA S GANDHI gandhi@me.iitb.ac.in 17

These are fundamental operations that will happen in microcontroller at the assembly level. So, all these operations are that way universal I would say, but one can see there are some opcodes or instruction sets which are in the assembly level for each of these fundamental operations. There are some logical operations, there are arithmetic operations, we saw this command ADD, then internal data transfer, we saw this command MOV.

So, like that there are this branching operation there this call command or jump or jump on carry, jump on 0, these are commands, which are for branching. So, you are executing some series, the commands, and from here now based on some condition, I want to execute something else, I will jump from there to something else execute that and come back and keep doing that. That way, you will have a facility to create loops and you have facility to create if-else statements and things like that as I said earlier.

So, these are the ways in which things actually happen at the assembly level. So, we do not need to get into whole lot of details here. But we need to have the sense that look, whatever commands I am writing here in the higher-level language, they will have, compiler will actually compile them and convert them into these opcodes. And then, those opcodes will be finally executed automatically right now.

(Refer Slide Time: 13:38)

Programming hardware interfaces

More modern micro-controller

- Special registers (variables in C) have certain functionality associated with them
 - Control registers**
Bits of control register control the operation:
for example register DDRA in D/I/O interface of XEP 100 controls if various pins of register A would act as input pins or output pins
 - Working registers**
- Working of interface and role of registers in this working needs to be understood for programming
- More details : See in the datasheets and sample programs of various microcontrollers you have handled

NPTEL PRASANNA S GANDHI gandhi@me.itb.ac.in 18

Now, let us move further for little bit higher level microcontroller. So, if you see your Arduino microcontroller or your XEP 100 their philosophy is (I mean philosophy) is the same you need to have a finally convert those codes into the assembly level language. But to access the registers, you do not have these register names A, B, C, D as you had in the 8085 kinds of a system, you will have these registers access there in a different way.

So, these registers are available as a variables in C, some variable in C. These variables would be defined in the datasheet of the microcontroller. And also, they will have header files provided to you in which those registers are mapped into, those variables are mapped into certain memory locations or registers finally. So, when you write to that variable, it will go to appropriate register in the microcontroller system, that data value whatever you have written in that variable, it will go to that particular register.

And there are two types of such registers, one is control register, and other is working register. Control registers for any interface, there will be these two kinds of registers. So, we will see some examples. Say for example, maybe I will explain first what this function is and then we will take a example.

So, control registers set up your interface for particular or configures your interface for a particular operation to happen and working register actually supply the working values which will be continuously changing.

So, control registers you set up only once in your program and then working register you may keep on changing based on whatever closed loop operation that you are carrying out. For

example, if you remember or some of you might have already done this programming of XEP 100 in that this DDRA register was there, DDRA variable was defined.

So, the call is DDRA now register only although it is a variable in C you would because this is mapped to a certain register which is fixed in the computer or a microcontroller memory, it can be referred to as DDRA register. So, say this register DDRA in digital input output interface in XEP 100 it controls as the name suggests direction definition register.

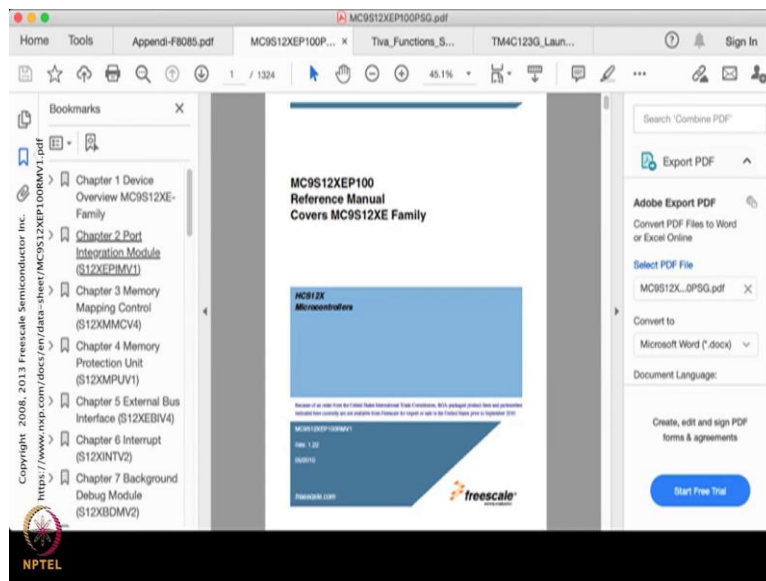
So, DDR is direction definition register. So, it defines the direction for the data transfer in this register pins of register A and this is A port that is available on the hardware pins of this microcontroller. So, this all will be given in the datasheet. We will look at this XEP 100 data sheet also in a minute. But this DDRA that way happens to be a control register then, you do not keep on changing the data directions typically in the register.

So, once you define, I am using this register as an output register and the pins are connected to some output LEDs, I will not change the direction now, once it is set, we will execute that for say some display of some numbers or dancing LEDs or some LED to glow for to indicate some operation happening that is the functionality that will be built into the microcontroller program.

So, that way this DDRA register is a control register. Now control register will just set up the things, it will not put values into any of the functions actually. So, for that you need a working register that is working registers will actually start putting values in that and outputting something that you want to execute in your final program.

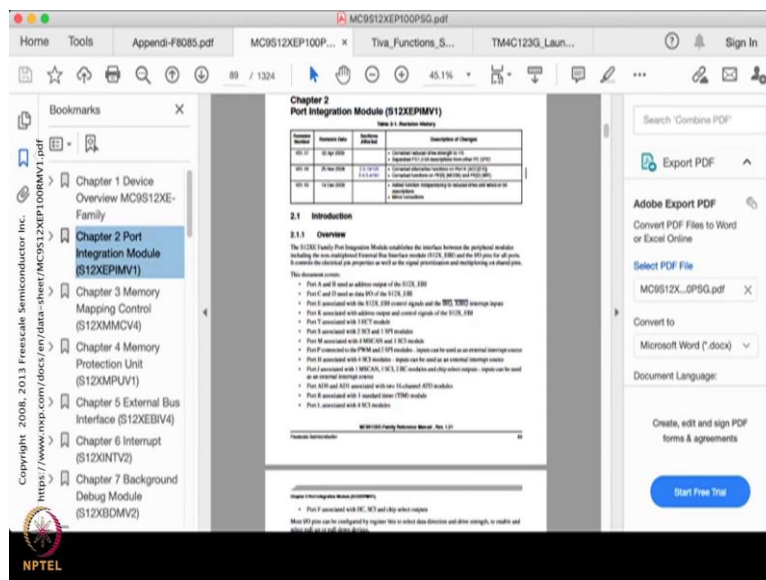
So, for example, DDRA will set this register to have an output definition for pins of port A and you actually write to this port A some data and then like that data will be displayed if it is output port, that is how you have written this program. So, we will see that now in a datasheet of this microcontroller. So, let us switch to that.

(Refer Slide Time: 18:33)



So, this is XEP 100 reference manual. Now, if you see this manual, it will have 1324 pages. So, we do not get into reading all these manuals to start putting programming. So, we want see this the module that is responsible for actually doing this data input output, is this port integration module and we can directly go to see what is there in that module.

(Refer Slide Time: 19:01)



MC9S12XEP100P5G.pdf

Home Tools Append-F8085.pdf MC9S12XEP100P... x Tiva_Functions_S... TM4C123G_Laun... Sign In

90 / 1324 45.1%

Bookmarks

- Chapter 1 Device Overview MC9S12XE-Family
- Chapter 2 Port Integration Module (S12XEPIMV1)
- Chapter 3 Memory Mapping Control (S12XMMC4)
- Chapter 4 Memory Protection Unit (S12XMPUV1)
- Chapter 5 External Bus Interface (S12XEBIV4)
- Chapter 6 Interrupt (S12XINTV2)
- Chapter 7 Background Debug Module (S12XBDMV2)

Copyright © 2008, 2013 Freescale Semiconductor Inc. <https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100RMV1.pdf>

NPTEL

This document contains the following information:

- Port A and B used as address outputs of the S12X_S180
- Port C and D used as data I/O of the S12X_S180
- Port E associated with the S12X_S180 control signals and the S12X_S180 interrupt inputs
- Port A associated with address output and control signals of the S12X_S180
- Port E associated with S12X enable
- Port A associated with S12X and S12X enable
- Port B associated with S12X and S12X enable
- Port F associated with S12X and S12X enable. Input can be used as an external interrupt source
- Port G associated with S12X enable. Input can be used as an external interrupt source
- Port H associated with S12X and S12X enable and the external interrupt input. Input can be used as an external interrupt source
- Port K and L associated with the H-Channel S12X enable
- Port I associated with S12X enable (TMR enable)
- Port J associated with S12X enable

MC9S12X Family Reference Manual, Rev. 1.0

Features

The Port Integration Module includes these distinctive features:

- Input and data direction registers for Port A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, U, V, W, X, Y, Z, and F when used as general purpose I/O
- Control registers to enable/disable pull-up/pull-down and when pull-up/pull-down is on Ports T, S, M, E, H, I, K, L, and F on per pin basis
- Control registers to enable/disable pull-up/pull-down on Ports A, B, C, D, E, and F on per pin basis
- Single control register to enable/disable pull-up on Ports A, B, C, D, E, and F on per pin basis and on S12X pin
- Control registers to enable/disable reduced output drive on Ports T, S, M, E, H, I, J, K, L, and F on per pin basis
- Single control register to enable/disable reduced output drive on Ports A, B, C, D, E, and F on per pin basis

Adobe Export PDF

Convert PDF Files to Word or Excel Online

Select PDF File

MC9S12X...0P5G.pdf

Convert to

Microsoft Word (.docx)

Document Language:

Create, edit and sign PDF forms & agreements

Start Free Trial

MC9S12XEP100P5G.pdf

Home Tools Append-F8085.pdf MC9S12XEP100P... x Tiva_Functions_S... TM4C123G_Laun... Sign In

91 / 1324 45.1%

Bookmarks

- Chapter 1 Device Overview MC9S12XE-Family
- Chapter 2 Port Integration Module (S12XEPIMV1)
- Chapter 3 Memory Mapping Control (S12XMMC4)
- Chapter 4 Memory Protection Unit (S12XMPUV1)
- Chapter 5 External Bus Interface (S12XEBIV4)
- Chapter 6 Interrupt (S12XINTV2)
- Chapter 7 Background Debug Module (S12XBDMV2)

Copyright © 2008, 2013 Freescale Semiconductor Inc. <https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100RMV1.pdf>

NPTEL

NOTE

If there is more than one function associated with a pin, the priority is indicated in the functions in the table from top (highest priority) to bottom (lowest priority):

Port	Pin Name	Pin Function (Priority)	Description	Pin Function after Reset
T	T0	TRST	Test Reset	TRST
		TRST	Test Reset	TRST
S	S0	SRST	Serial Reset	SRST
		SRST	Serial Reset	SRST
M	M0	MRST	Memory Reset	MRST
		MRST	Memory Reset	MRST
E	E0	ERST	External Reset	ERST
		ERST	External Reset	ERST
H	H0	HRST	Host Reset	HRST
		HRST	Host Reset	HRST
I	I0	IRST	Interrupt Reset	IRST
		IRST	Interrupt Reset	IRST
J	J0	JRST	Interrupt Reset	JRST
		JRST	Interrupt Reset	JRST
K	K0	KRST	Interrupt Reset	KRST
		KRST	Interrupt Reset	KRST
L	L0	LRST	Interrupt Reset	LRST
		LRST	Interrupt Reset	LRST

MC9S12X Family Reference Manual, Rev. 1.0

Adobe Export PDF

Convert PDF Files to Word or Excel Online

Select PDF File

MC9S12X...0P5G.pdf

Convert to

Microsoft Word (.docx)

Document Language:

Create, edit and sign PDF forms & agreements

Start Free Trial

MC9S12XEP100P5G.pdf

Home Tools Append-F8085.pdf MC9S12XEP100P... x Tiva_Functions_S... TM4C123G_Laun... Sign In

95 / 1324 45.1%

Bookmarks

- Chapter 1 Device Overview MC9S12XE-Family
- Chapter 2 Port Integration Module (S12XEPIMV1)
- Chapter 3 Memory Mapping Control (S12XMMC4)
- Chapter 4 Memory Protection Unit (S12XMPUV1)
- Chapter 5 External Bus Interface (S12XEBIV4)
- Chapter 6 Interrupt (S12XINTV2)
- Chapter 7 Background Debug Module (S12XBDMV2)

Copyright © 2008, 2013 Freescale Semiconductor Inc. <https://www.nxp.com/docs/en/data-sheet/MC9S12XEP100RMV1.pdf>

NPTEL

NOTE

If there is more than one function associated with a pin, the priority is indicated in the functions in the table from top (highest priority) to bottom (lowest priority):

Port	Pin Name	Pin Function (Priority)	Description	Pin Function after Reset
T	T0	TRST	Test Reset	TRST
		TRST	Test Reset	TRST
S	S0	SRST	Serial Reset	SRST
		SRST	Serial Reset	SRST
M	M0	MRST	Memory Reset	MRST
		MRST	Memory Reset	MRST
E	E0	ERST	External Reset	ERST
		ERST	External Reset	ERST
H	H0	HRST	Host Reset	HRST
		HRST	Host Reset	HRST
I	I0	IRST	Interrupt Reset	IRST
		IRST	Interrupt Reset	IRST
J	J0	JRST	Interrupt Reset	JRST
		JRST	Interrupt Reset	JRST
K	K0	KRST	Interrupt Reset	KRST
		KRST	Interrupt Reset	KRST
L	L0	LRST	Interrupt Reset	LRST
		LRST	Interrupt Reset	LRST

MC9S12X Family Reference Manual, Rev. 1.0

Adobe Export PDF

Convert PDF Files to Word or Excel Online

Select PDF File

MC9S12X...0P5G.pdf

Convert to

Microsoft Word (.docx)

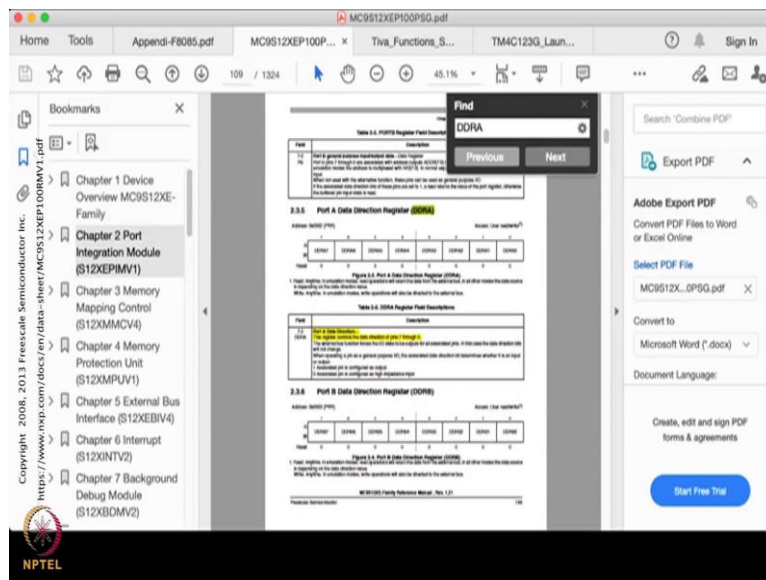
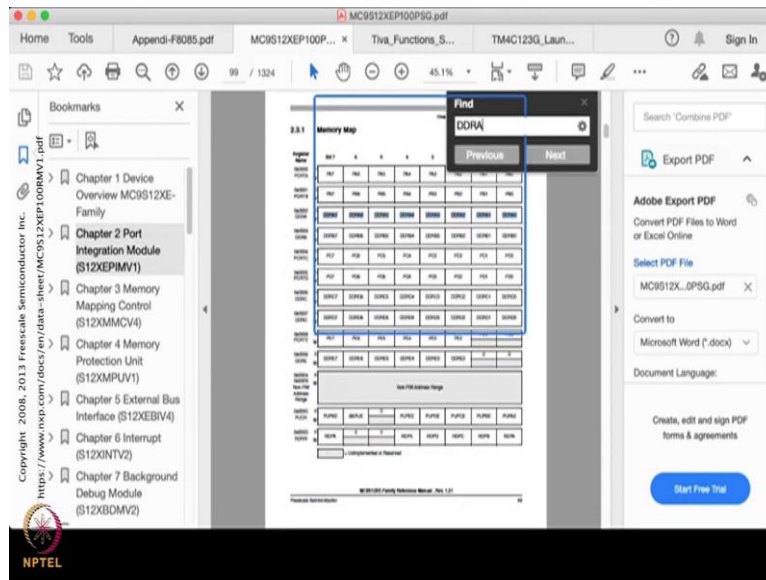
Document Language:

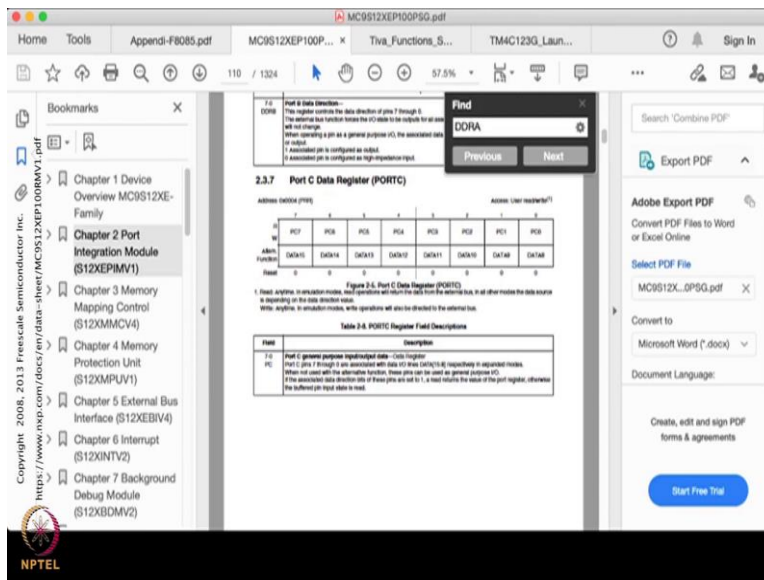
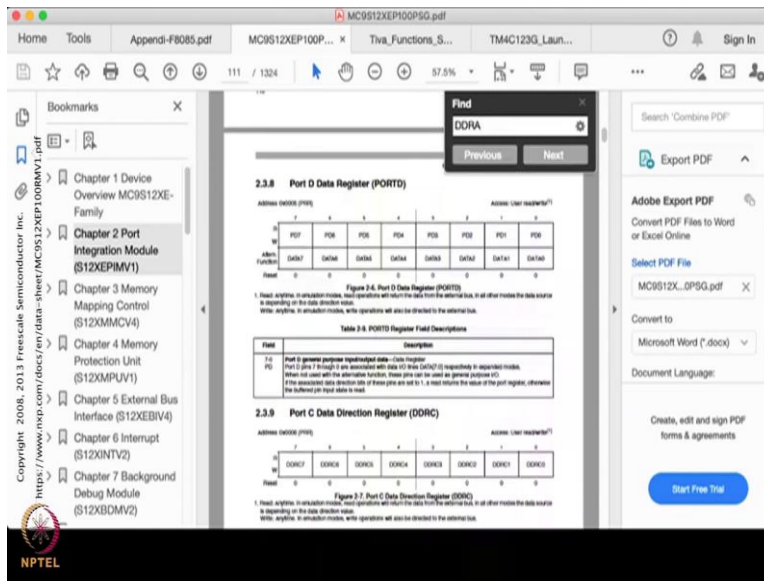
Create, edit and sign PDF forms & agreements

Start Free Trial

And you can see that you can read through some of these details as is port A port B something is given then there are these different variables will be given. Then number of pin definitions will be given like that all these things are given. So, you can, if you know this name of some registers, you can look for those registers as there can say control find and look for these registers.

(Refer Slide Time: 19:38)





So I say here DDRA and I get all the details about this register. So, we can see here, so this is data register DDRA data direction registers for port A. So now, it gives me all the definitions if the value is written there, it will be defined as input port or output port like that. It gives all the description and according to what is written in the datasheet if I set up this register some number, then accordingly the functions will start happening in the hardware.

So, that is how one goes about programming these different-different microcontroller interfaces. So, this is about this simple digital input output interface. These are many different you have DDRB, DDRC and things like that. Then you will have port register port A, port B, port C is a data register. So, port A when you say like the data will be given to this port or read from this port.

Port C means the data will be, suppose I want to write or this is port is output port, I will write this output number into this port and depending upon what is your data say that data is 10101010, then wherever that 1 is there that LED will glow, 0 is there LED will be shut off or something like that.

So, you are putting the hardware pin high and low values on the hardware pin by putting these data into respective registers, that is our operation that takes place in the input output ports.

So, this is how we look for programming for little higher-level microcontrollers. So, this thing you will be able to observe for many different kinds of interfaces. So, these different-different modules, so, on these, you see different chapters in the datasheet we will find what are modules that are provided by this microcontroller or in the datasheet you will find all these modules that are provided and more details of those interfaces will be observed in the respective chapters of the datasheet.

(Refer Slide Time: 22:15)

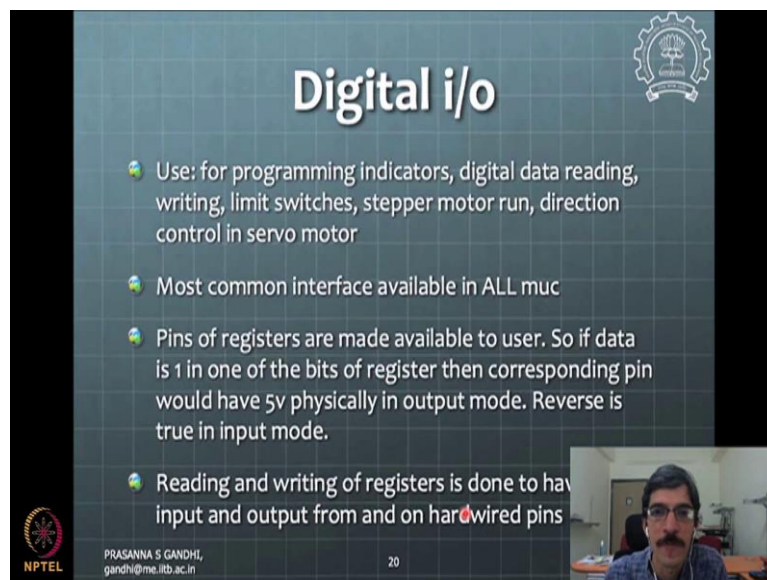
The slide is titled "Interfaces types" and is presented on a dark blue grid background. It lists two main categories: "Sensor" and "Actuator". Under "Sensor", the sub-items are: "Digital input output (D i/o): Names such as GPIO (Tiva), Port Integration Module (XEP 100)", "Analog input (ADC)", "Quadrature Encoder Interface (QEI)", and "Serial and other communication modes". Under "Actuator", the sub-items are: "PWM", "Digital to analog conversion (DAC)", and "Serial and other communication". The slide includes the NPTEL logo in the bottom left, the presenter's name "PRASANNA S GANDHI" and email "gandhi@me.iitb.ac.in" in the bottom center, and the number "19" in the bottom right. A small video inset of the presenter is visible in the bottom right corner.

So, typical interfaces will be of these different-different kinds. So, depending upon microcontroller that you choose you will have these different interfaces and interfaces names also will change from one microcontroller to another microcontroller. Say for example, this digital input output interface as we saw just now in XEP it is called port integration module or PIM. In the Tiva, it has a name GPIO or general-purpose input outputs. So, like that you will have different-different names in different microcontroller. So, do not get bogged up with that you can just understand this is your interface that you need to program.

Typically for mechatronic applications from sensors perspective, we will need these interfaces, general purpose digital i/o input outputs. Analog to digital conversion and then quadrature encoder interfaces and maybe some serial or other communication modes can be possibility. Especially if the sensors give directly the output in terms of serial port or port like a CAN interface then we will need these serial or other communication modules to be programmed.

Actuator typically will run by PWM module. In some cases, there is a possibility of running the actuator by using this digital to analog conversion also, but in most of our cases it may not happen.

(Refer Slide Time: 23:55)



The slide is titled "Digital i/o" and features a grid background. It contains four bullet points, each with a small globe icon. The first bullet point discusses uses for programming indicators, digital data reading/writing, limit switches, and stepper motor control. The second states it's a common interface in all microcontrollers. The third explains that pins of registers are made available to users, with data 1 resulting in a high voltage on the pin in output mode. The fourth states that reading and writing of registers is done via hardware pins. In the bottom right corner, there is a small video inset showing a man with glasses and a mustache. The slide footer includes the NPTEL logo, the name "PRASANNA S GANDHI", his email "gandhi@me.iitb.ac.in", and the number "20".

- Use: for programming indicators, digital data reading, writing, limit switches, stepper motor run, direction control in servo motor
- Most common interface available in ALL microcontroller
- Pins of registers are made available to user. So if data is 1 in one of the bits of register then corresponding pin would have 5V physically in output mode. Reverse is true in input mode.
- Reading and writing of registers is done to hardware input and output from and on hardware pins

So, I think maybe now we will just talk about little bit more details of these interfaces. But we will come back maybe in the in the next class. So that I have this small chunk of thing about just to maybe base fundamentals of these and then we can come back to this more details of these interfaces in the next part of the today's class. So, I will stop here for now.