

Design of Mechatronic Systems
Professor Prasanna S. Gandhi
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Lecture No 22

Interfacing Actuator using QEI in Tiva Launchpad + ISR

Last class, we looked at interfacing using Tiva Launchpad for the PWM interface. So, how do you run the motor by using PWM interface. We took example of Tiva, but that can be extended the syntax is different, but the philosophy remains the same. So, the same thing can be extended to any other microcontroller as well.

Now, today's class, we are going to look at interfacing of encoder by using this dedicated quadrature encoder interface, QEI interface in Tiva launchpad. So, again, here, we will introduce first this philosophy of how these encoder interfaces they were, typically, what kind of registers or control registers are there for getting these interfaces done. And then, we will look at a little bit more details about programming using Tiva launchpad.

And further than that, we are going to look at also the sampling time. How do you set up the sampling time in the mechatronic system working? So, microcontroller needs to have some specific sampling time for the system to work. By all those kind of things we will discuss in the class.

(Refer Slide Time: 01:44)

Working of incremental Encoders: recall

The diagram shows an optical-slotted vane rotary switch with a rotating shaft, a light source, a light detector, and a quadrature slotted vane. The text below the diagram reads: "The optical-slotted vane rotary switch".

- Just slits to generate pulses
- Two sensors A and B to sense direction
- Two outputs if at 90° out of phase with each other – quadrature output
- Third sensor required for determination of absolute position called Index pulse or I

NPTEL P S Gandhi, gandhi@me.iitb.ac.in

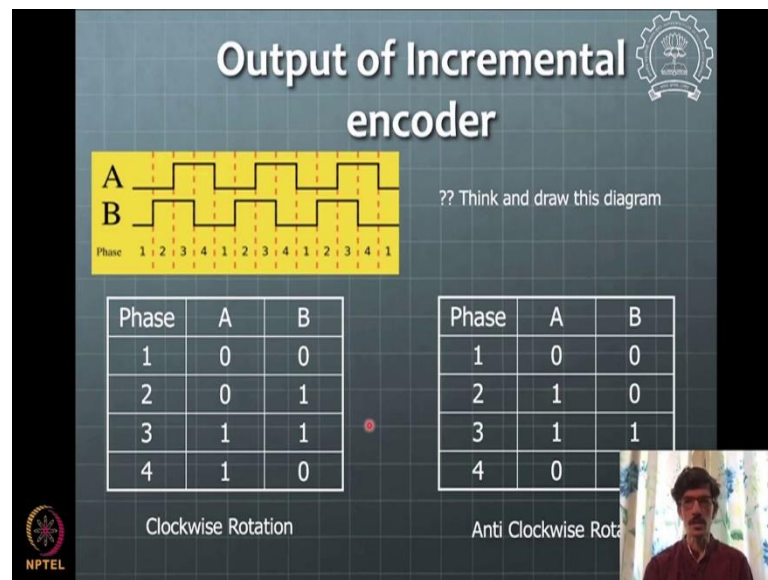
As you know, this working of incremental encoders we are just recalling, we have seen it before. So, what happens is, we have just slits in the wheel and then you have this light source and a light detector photodiode, and this light source and detector they work in pair.

And when the light passes you get a pulse. Light passes through that and then this the slit cuts the light you get a pulse.

And then, we know that there are two sensors needed, A and B to sense a direction in this case. So, these two outputs A and B are 90 degrees out of phase with each other. How they are made 90 degrees out of phase by their proper placement with respect to each other electrical domain. You do not need to have them so closely spaced. One can be sensing here and other can be sensing somewhere else.

Since these all the slits are uniform, you will get 90 degrees out of phase all the time. And we may use another sensor to get a position, absolute position of the encoder pin, that is called index pulse. It is typically termed as I said Z.

(Refer Slide Time: 04:01)



So, we know that these are the pulses that we get, and in one direction you will get this kind of pattern of the signals in other direction you get some other pattern of the signals. And A and B signals, they are to be processed in some way to read these patterns, and get you know encoder output. So, this is what encoder, any circuit for decoding this they do. They recognize this pattern, and then they produce some pulses corresponding to each of this phase or in each of these kind of a combination of A and B.

And then that pulses are to be counted to get the encoder count. So, that is how the internal circuitry would be designed or positioned to count these pulses and then change the direction based on change the direction the count up or count down will be changed.


And then, direction change will be sensed by saying, okay, look, after 00 if there is 01 then there is a clockwise direction. And after 00 there is a 10 then there will be anticlockwise direction. Something of that sort some logic will be used to get the direction sense.

(Refer Slide Time: 04:34)

Why incremental encoders? Uses and Advantages

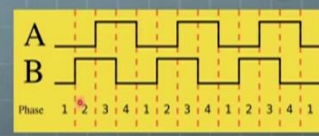
- 🌐 Digital output error free
- 🌐 Quadrature (4x) output: four pulses per one hardware slit
- 🌐 Accurate speed estimation possible by looking at the time between two pulses
- 🌐 Q: How pulses can be counted up and down by TIVA?
- 🌐 Q: How speed can be estimated?

P S Gandhi, gandhi@me.iitb.ac.in



Output of Incremental encoder

?? Think and draw this diagram




Phase	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Clockwise Rotation

Phase	A	B
1	0	0
2	1	0
3	1	1
4	0	0

Anti Clockwise Rotat



So, this is, we have many advantages of using this incremental encoders. As we have seen, they are digital output sensors, so, they are noise free. So, there is no noise error in the signals, and small amount of noise if at all that is there, that can be also taken care of by some filtering, as we will see in this QEI interface of Tiva. Then it has a quadrature output, a 4x kind of output. So, 4 pulses we will get per one hardware slit.

So, this is what will happen like. These are the four kind of signals you will get in one cycle of A, for example or one cycle of B, so to say. So, one slit will correspond to 4 pulses that is

why it is called the term quadrature. This is a quadrature output. And then you get a very accurate speed estimation by looking at time between two pulses or number of pulses in a specific amount of time.

That is a way you get very accurate estimate of the speed in these encoders. There are two ways. One is like, look at the time between the two pulses or you look number of pulses in a given amount of time. Now, how this is up and down count is done in Tiva? How the speed can be estimated we will see in detail for the Tiva interface.


(Refer Slide Time: 06:07)

QEI interface

- Use: Interfacing incremental encoders, up and down count of quadrature output of encoder, speed estimation
- Two QEI modules provided in TIVA. Each has input of A, B signals and index pulse Z. Features
 - Position integrator to track encoder position
 - Programmable noise filter (on the inputs A B Z)
 - Velocity capture using built-in timer
 - Input frequency upto $\frac{1}{4}$ of processor frequency

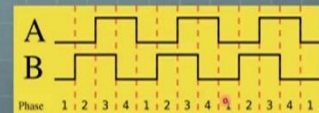
6

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



Output of Incremental encoder

?? Think and draw this diagram




Phase	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Phase	A	B
1	0	0
2	1	0
3	1	1
4	0	1

Clockwise Rotation Anti Clockwise Rot

NPTEL



So, as we have seen this interface is for up and down counting of the incremental encoders, as the position changes for the encoder, and then, also for the speed estimate. Now, see, in the absence of this interface you would need some kind of a circuitry to do this job. So, if you are

interfacing encoder with not with Tiva, but any other kind of a microcontroller and that does not have this QEI interface, then what to do?

Then you will use this chip called HCTL 2021. That HCTL chip is going to do this job of what this interface is doing. So, basically idea is that there is some kind of a clock and based on the clock you will take these inputs A and B and process them internally to as I said to generate 4 pulses per one kind of cycle of A, and then those 4 pulses or like those number of pulses will be counted by using some kind of a counter. And that up and down count will be changed based on the direction, what direction logic.

So, this is how this interface will typically work. And we will not get into the details of logic circuit to do that and all, but we just need to kind of understand that, okay, it will have to take inputs of these A and B and process them in some way to kind of get you the count. So, this will happen in QEI interface, which is already there in the microcontroller circuit itself, it is part of the microcontroller. So, you do not need any extra chips, as for a Tiva kind of a microcontroller if you are using this interface. So, there are two modules in Tiva for QEI, so you can interface 2 encoders in Tiva.

And each of the module has input signals A and B and index pulse also is input so three inputs are provided in each. And typical features are there is a position integrator to track the encoder position, of course, it will it should give us a position that is what we are interested in, then programmable noise filter is there on inputs A and B and Z, you can have some kind of a noise filter to be applied.

Then velocity capture that is done using some built-in timer. We will see, how this velocity capture is done. And the input frequency can go, as high, as up to $1/4^{\text{th}}$ of the processor frequency. So, this is really, really fast kind of a interface. So, then you can capture very large amount of speed or a very high-speed data can be captured. Because higher the speed the number of pulses coming you can see would be more, much more quicker.

So, for example, if you see these pulses that are coming, higher speed this time duration between the two pulses of this A or B is going to start reducing. And it may reduce at some very high speed to some extent that you may not be able to count that fast. So, that is a limit that is given here. $1/4^{\text{th}}$ of the processor frequency is what. So, if you have 50-Megahertz processing currency, typically, 12.5-Megahertz will be the frequency of input. So, that is pretty high. I mean, it depends upon the encoder and, of course, the resolution you are using.

If you are using an encoder with a nanometer kind of a scale resolution, then your speed will get limited by this number. But if you are using micrometer kind of resolution for the encoder then. I mean, I am talking of linear right now, but it is applicable in the rotary case as well. So, higher speed if you want, you need a faster kind of a microprocessor to get that higher speed encoder readings recorded, otherwise you will have a trouble. Otherwise, you will miss out the or you will lose the pulses. And when the pulses are lost then your count will not be faithful.

So, if you are using only the speed kind of a application there are some different kind of a ways we can, one can do this operation. You do not need then, depend upon how fast your speed application is. You can just make use of index pulse only to do the speed monitoring. If your speed fluctuations between the two kind of index pulses or in within the rotation are not too many to not bothering to your application then that can be done.

But for all the robotic applications, you will need an accurate positioning where you are interested in not just the Index pulse but you are interested in entire speed of the robot from starting to end to kind of a trajectory point trajectory then no you have to have a, plan your hardware, I would say, to do this high-speed kind of measurements faithfully.

(Refer Slide Time: 11:39)

The slide is titled "QEI interface" and features a list of "Additional Features" on a dark grid background. The features listed are: "Interrupt generation on Index pulse", "Velocity timer expiration", "Direction change", "Phase signal error detection", and "Index pulse can be used to reset the position counter". The slide includes the NPTEL logo in the bottom left, the speaker's name "PRASANNA S GANDHI" and email "gandhi@me.iitb.ac.in" in the bottom center, and a small video inset of the speaker in the bottom right corner.

Then, there are additional features where there is an interrupt generation is also possible. So, you can generate interrupt on index pulse, this will be useful for resetting your encoder value, for example, in any case. So, say for example, if you have a robot kind of application it is a homing or 0 position of that robot on a powerup can be done based on this kind of idea.

Then, you have, after velocity timer expiration there is some kind of a interrupt generation. So, those are these kind of different-different possibilities for interrupt generation, direction change happens, you generate the interrupt. Then phase signal error head, then in that case you can generate some interrupt to kind of halt your application or something like that. So, this interrupt, how we can use is up to you to use this, but this is a facility that is provided by the interface.

(Refer Slide Time: 12:40)

The slide is titled "QEI interface in TIVA" and is part of a "Basics" series. It features three bullet points: "The two phase signals, PhAn and PhBn, can be swapped before being interpreted by the QEI module in order to correct for miswiring of the system", "Input signals have a digital noise filter on them that can be enabled through FILTCNT bit of the QEICTL register", and "Mode of operation can be changed using SIGMODE bit of the QEICTL". At the bottom, there is a "QEI Module Block Diagram" and a small video inset showing a man speaking. The NPTEL logo is visible in the bottom left corner.

Now, let us see a little bit more detail of this interface in Tiva. So, there are two phase signals PhAn and PhBn that is what they call it, n is like channel 1, channel 2 something, and they can be swapped before interpreted by the module. So, these are additional facility that is provided in Tiva.

See, many time this happens, when you say rotate the encoder you see to set up things properly for our mechatronics system to work as what we do in our normal calculations or mathematical model, the positive voltage given to the motor should result in positive displacement. Many times, this is, this may not be true based on your hardware connections.

So, if that is the case then we need to just swap these phase signals two phase signals. Then, what will happen is, for positive, suppose for positive rotation, the count is actually going down or count is counting down then by swapping you can correct for that. That is a facility that is provided in the Tiva. So, to correct some miswiring of the system is what they say.

So, you miss in the wiring, so you can correct it by checking the wiring and doing it correct wiring also, but this is a software facility that is provided by the interface to avoid that

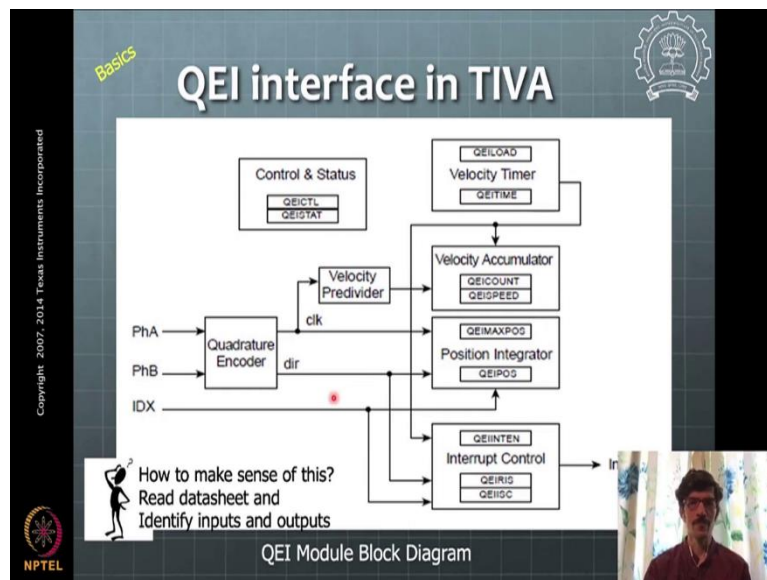
drudgery. Then input signals will have this digital noise filter that filter can be enabled by using this bit in this control register. So, see, these are the kind of registers that you need to look for doing this, using this facility.

So, if you want to filter, use noise filter, then you use this kind of bit in this control register, and you will get that facility. So, we will see there are two, again, as we saw in the case of PWM there are two ways of a philosophy of programming, and two ways of programming. That both ways can be okay to kind of go ahead with.

One is like, based on these registers understanding how these registers work in the case of Tiva in particular here, and other is by using API libraries that are provided by the manufacturer. So, we saw that with that philosophy and how that can be used in case of PWM we are going to look at similar kind of a thing in the case of this quadrature interface today.

Then the mode of operation that can be changed. So, as you can see, what are the modes of operations that are available, but that can be changed by using some other kind of a bit in. So, there are this kind of description will be given, and then this module block diagram is given.

(Refer Slide Time: 15:45)



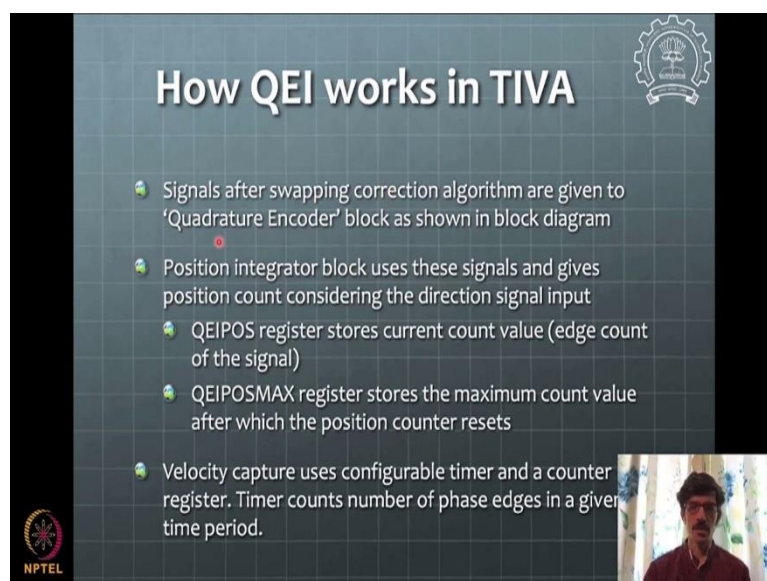
So, how do you read now, this block diagram? How to make sense of this? So, you need to read a little bit of a data sheet. See these names that you see here they are basically registers. QEISTAT is a status register, QEICTL is a control register, then some timer load register is there. Then you will have these signals are coming and getting into this some circuitry, which will give us output of the direction and number of pulses.

So, the phase A phase B for one cycle it will degenerate 4 pulses here. And those pulses are going to get kind of counted somehow up and down, and the output will be available in this QEIPPOS kind of register. And these are the kinds of ways one can think about reading this. Then there are these interrupt control that is there here. So, some bits to kind of do this interrupt control. Some control registers to do that interrupt control.

So, you need to see, what are the inputs out of these, what are the outputs and see whether, say for example, you can set this QEITIME or QEILOAD kind of register. And this QEICOUNT and QEISPEED are the results or outputs that are to be seen actually, so these are the output registers.

QEIPOSE is, again output register, QEIMAXPOS is where you can write this maximum position beyond which this counting will get reset. Like that there are some things, some of these things I am telling from reading the datasheet, but you can also find out these register details and you can start making sense of such diagrams. Then we will have a little more description now about these.

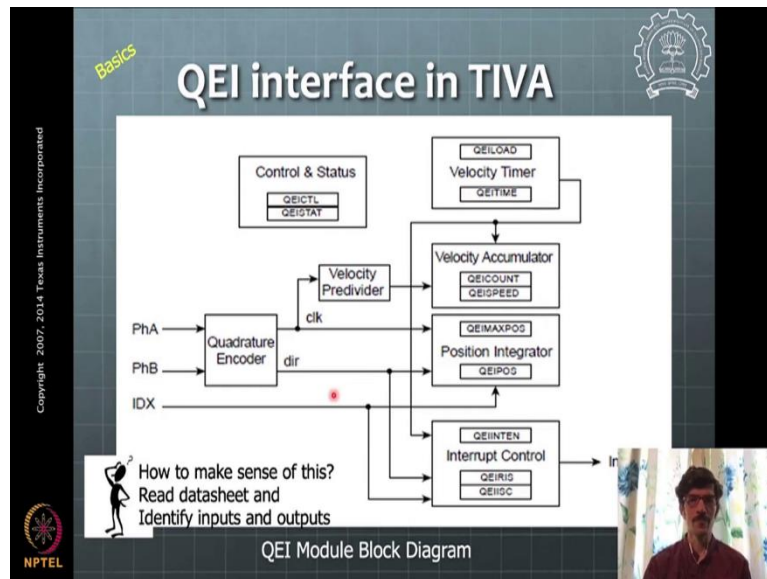
(Refer Slide Time: 17:59)



How QEI works in TIVA

- Signals after swapping correction algorithm are given to 'Quadrature Encoder' block as shown in block diagram
- Position integrator block uses these signals and gives position count considering the direction signal input
- QEIPPOS register stores current count value (edge count of the signal)
- QEIPPOSMAX register stores the maximum count value after which the position counter resets
- Velocity capture uses configurable timer and a counter register. Timer counts number of phase edges in a given time period.

The slide features a dark blue background with a grid pattern. In the top right corner, there is a circular logo with a gear and a scale. In the bottom left corner, there is a small circular logo with the text 'NPTEL'. In the bottom right corner, there is a small video inset showing a man with a beard and glasses speaking.



So, this after swapping these signals are coming up here, so these are already swapped signals. So, swapping algorithm is somewhere like something will be there in the control So, then, So, then we have this more description of QEI. So, position integrator block is what is using these signals, as we saw here. See, position integrator is here, it uses these signals which are processed by this quadrature encoder, and then directly gives us this position in the QEIPPOS register.

So, see, this is so easy. I mean, so, I do not know if you have an experience of reading encoder, but we had done it as a part of some other kind of activity. And where to interface that HCTL chip to get to this point where you are, actually start reading this, so you need to have this circuitry in place. So, you need to connect wires, you need to have a hardware or design some PCB and then get that HCTL chip to interface. Instead of that in Tiva it is happening all internally done. You just need to program it and start using.

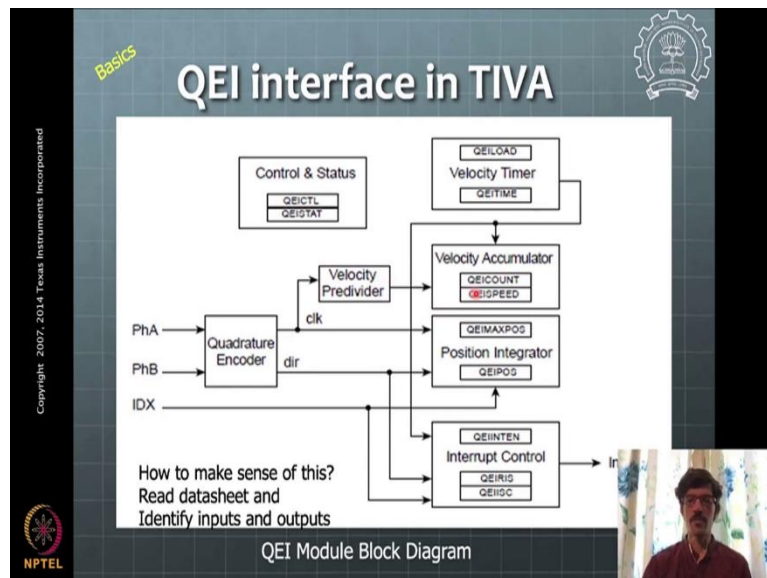


So, this way it is so easy. If you have this kind of interface that is directly available, as a facility for a user to program in the microcontroller itself. So, this is good to use these encoders this kind of microcontrollers which will have this QEI interface for many mechatronics applications, it will be very useful.

Because encoders are what, give very accurate kind of position estimator, we have seen, because of low noise and it is that is why it is more kind of a prominently used in mechatronics applications. And that is where these interfaces or microcontrollers with these interfaces will become important for the mechatronic engineer to develop some applications.

(Refer Slide Time: 20:19)

How QEI works in TIVA

- Signals after swapping correction algorithm are given to 'Quadrature Encoder' block as shown in block diagram
- Position integrator block uses these signals and gives position count considering the direction signal input
- QEIPOS register stores current count value (edge count of the signal)
- QEIPOSMAX register stores the maximum count value after which the position counter resets
- Velocity capture uses configurable timer and a counter register. Timer counts number of phase edges in a given time period.



Then velocity capture is using a configurable timer and a counter register. And then the timer is counts number of phase edges in a given time period. So, if you see this block diagram this is number of these edges. So, say, we have seen that, okay, these are the on this line you will have some pulses, which are coming based on the phase A phase B signals.

So, these pulses will be typically 4 pulses in one kind of a cycle. And these pulses will be counted for a given amount of time, and then you will know that this is the speed. So, that is what is being done, and then, there is some kind of a velocity pre-divider which will be available in the controlled QEI-control register you can set this pre-divider value so that you get some kind of a facility of reducing the number of counts to do some accuracy or like to make sure that the numbers do not overflow the register that are used to store them. So, this is



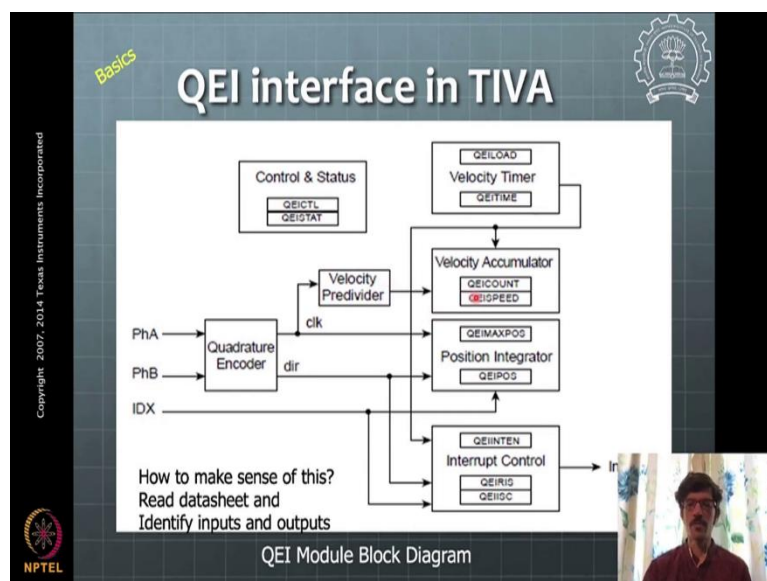
how this philosophy of a velocity capture works. The timer will count the number of phases in a given time period.

(Refer Slide Time: 21:51)

How QEI works in TIVA

- Velocity capture
 - Time period for capturing the number of edges can be specified using QEI Timer load (QEILOAD) register
 - Current edge count after velocity pre-divider (can be set in QEICTL register) is stored in QEICOUNT register
 - Edge count from previous cycle is available in QEISPEED register

Q: How to find out speed of motor shaft in rpm encoder specifications are known and relevant register values are known?

Now, how do you set up this time period for capturing? It is with QEILOAD register. So, this count in this register will count the number of clock cycles, which are there and then the QEILOAD. Suppose I have a QEILOAD value to be 100, so 100 clock pulses is counted before this timer is reset. So, for that 100 clock pulses we see how many edges of the microcontroller, how many edges of the encoder phase signal they are counted or how many counts of encoder happen in the 100 kind of a value of this QEILOAD?

So, QEILOAD when it is 100, it will count 100 clock pulses, so that much is a time that is given for this timer. That is how one can think about. See, typically, this comes from like,

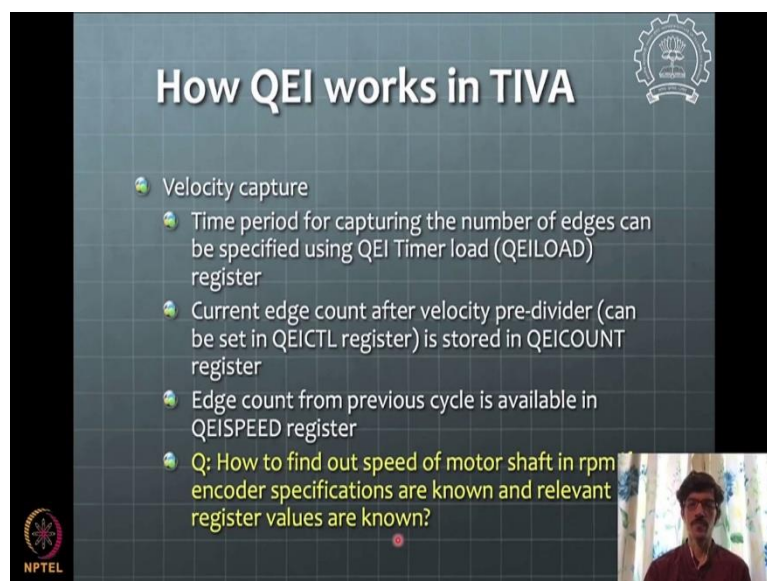
you cannot say, I will give some microsecond or millisecond kind of a time in this timer. No, that time is not typically given. So, everything in the microcontroller happens with reference to the clock.

So, if the clock is running at say whatever frequency, number of clock cycles is what you will typically say as a time. So, this when timer load register this is see, so typically, we will deal with all integer values in these registers. If you think carefully all these values that we are dealing with are integer values in the register. So, you cannot say some unit time, you say that unit in terms of number of clock cycles and that will be like a integer value. So, that is how like you specify typically in the microcontroller philosophy, these different values of registers.

Then the count is also pre-divided pre-division of the velocity count will be possible which can be set in the QEICTL register. And in this value, current value will be available in QEICOUNT register. And once this timer is finished, then that value will be transferred to this QEISPEED register.

So, this count is current value. And once the count, the timer is finished that value will be transferred to QEISPEED register and it will be available till the next time it is getting reset. So, current speed will be taken from the QEISPEED register. And then, it depends upon how fast you want to update the speed, that you can put a counter, put a timer value accordingly.

(Refer Slide Time: 24:53)



The slide is titled "How QEI works in TIVA" and features a gear icon in the top right corner. It contains the following text:

- Velocity capture
 - Time period for capturing the number of edges can be specified using QEI Timer load (QEILOAD) register
 - Current edge count after velocity pre-divider (can be set in QEICTL register) is stored in QEICOUNT register
 - Edge count from previous cycle is available in QEISPEED register

Q: How to find out speed of motor shaft in rpm encoder specifications are known and relevant register values are known?

The slide also includes an NPTEL logo in the bottom left corner and a small video inset of a man in the bottom right corner.

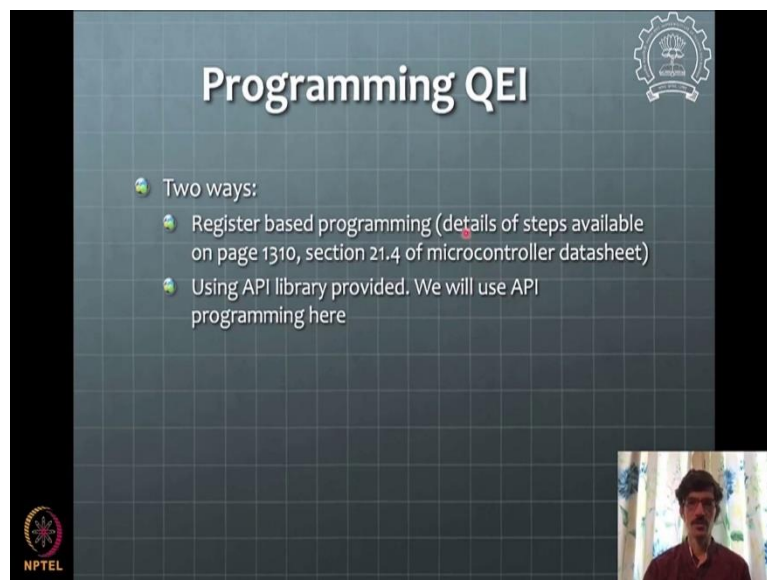
Now, let's see, how do you find the speed of motor shaft in RPM. If the counter specifications are unknown, and relevant register values are known. So, this is a question that

you need to ponder over. I will leave it at this point, you think about. Suppose, I have a motor, which is using 1024 kind of counts in one revolution or 1024 slits for the one revolution of the motor, that is an encoder specification that is given.

So, we have 1024 slits, so quadrature will have those many number of counts, 1024 into 4 counts will be there, 4096 counts will be there. Now these kind of counts are coming. And then we want to know what is the motor RPM. When we have set up some, we know the clock signal coming at some frequency, and we have set up timer for some duration, time duration. And in that time, we are counting how many number of pulses are coming.

Then based on some simple mathematics you may be able to estimate these value of motor RPM. So, think about how will you do this. Maybe, we can have this as your assignment or a quiz problem, exam problem.

(Refer Slide Time: 26:30)



The slide is titled "Programming QEI" and features a gear icon in the top right corner. It lists two programming methods:

- Two ways:
- Register based programming (details of steps available on page 1310, section 21.4 of microcontroller datasheet)
- Using API library provided. We will use API programming here

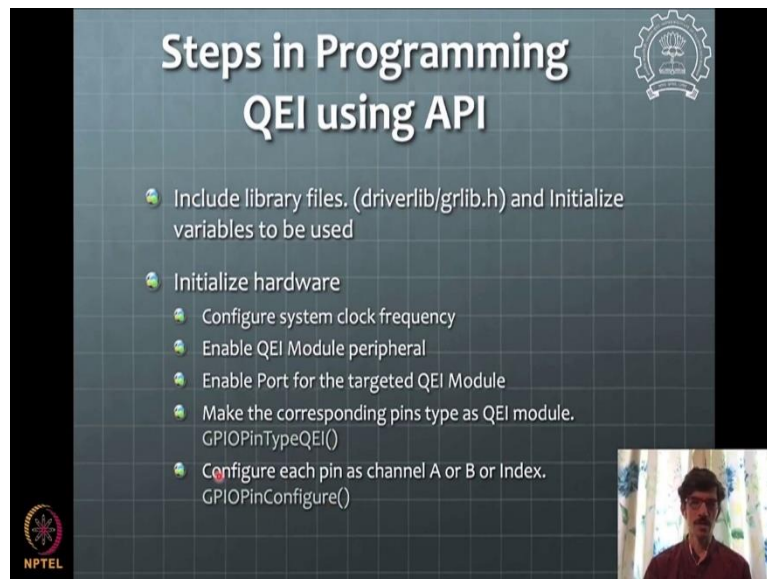
The NPTEL logo is visible in the bottom left corner, and a small video inset of a man speaking is in the bottom right corner.

Then, you have again, as I mentioned two ways to program QEI interface. First is a register-based programming. So, I am giving you some reference here also, so that you can go to that page 1310 of this microcontroller datasheet that we have shared. And in that you will find all the details of these different-different registers that are there, and steps that are required to be done to program this interface. And second way is, using API libraries.

So, these, again these functionalities of this API library are provided in the syntax of API. That is again, another data sheet that is provided. So, typically, microcontroller manufacturers will provide you with this datasheet the details of this how the microcontroller can be programmed.

In some cases, you will have this API library files, in some cases you will have to work with only this register. For XCP for example for XCP100 kind of a microcontroller, we have only these register-based programming happening, and in Tiva you have possibility of both. And see, these API libraries are nothing, but if you see these functions, if you see the function details, you will find the same thing these functions use finally the register-based programming. So, this is like the base level programming, register-based programming.

(Refer Slide Time: 28:16)



The slide is titled "Steps in Programming QEI using API" and features a list of five steps, each preceded by a gear icon. The steps are:

- 1. Include library files. (driverlib/gplib.h) and Initialize variables to be used
- 2. Initialize hardware
 - Configure system clock frequency
 - Enable QEI Module peripheral
 - Enable Port for the targeted QEI Module
 - Make the corresponding pins type as QEI module. `GPIOPinTypeQEI()`
- 3. Configure each pin as channel A or B or Index. `GPIOPinConfigure()`

The slide also includes the NPTEL logo in the bottom left corner and a small video inset in the bottom right corner showing a man speaking.

Now, typical steps that are used in programming this QEI using APIs are, first, you need to have these library files available, and initialize. These are important, so that you can you make use of the functionalities that are provided, and then initialize some variables that you want to use in the system.

Then for initializing hardware, these are the steps for QEI to be followed. So, configure clock frequency. This is a typical step for many interfaces we need to do. Then enable this module, this peripheral module enabling some functions will be there you use them to enable the module. Then now, see, we, if you see this map of the pins of this Tiva microcontroller launchpad in this there are programmable GPIO interfaces which can be changed to act as a QEI interface or it can be changed to act as some other kind of interface.

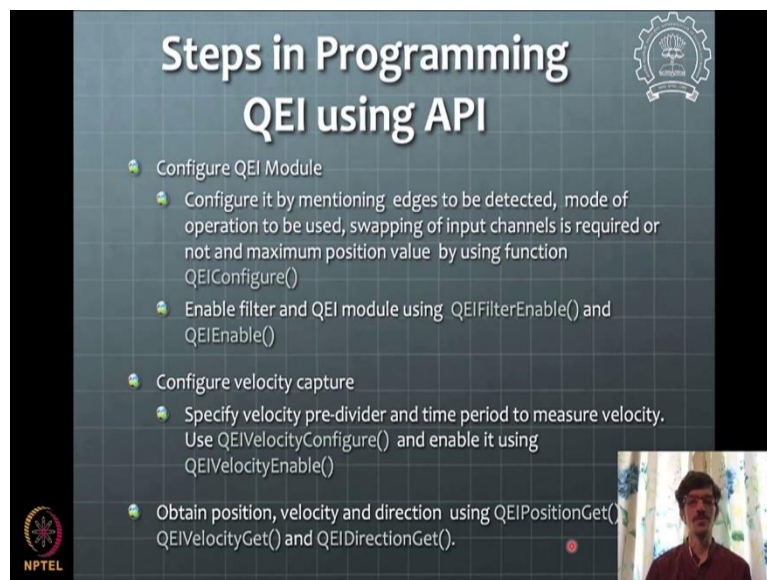
So, we need to, before we start using it as a QEI module, we need to tell microcontroller that, okay look, I want to use this particular pin provided for the QEI kind of a module. So, for example, D port has some pins which are used for QEI module. We need to enable those particular kind of ports and then further map the pins to act as a QEI.

So, this is done this GPIO pin Type is to be defined in some function. So, this is a function that is given for making these pins to act as a QEI pins. Then then pin configure which I configured these pins for A, B or index kind of a pins. So, these are the kind of typical steps that will be given. These is all given in the function library datasheet or manual.

So, you need to find these functions, how they are working. What is a input output, and all those details you need to find, and you will get that. So, see, I will give just kind of a brief of some of the things, but you can go ahead and look at these particular. You can just search for this function in the whole datasheet, otherwise, the datasheet is too huge.

So, to find you can get into QEI interface, and then start reading some of the part and then maybe if you want you can search for this particular kind of phrase, and then you will be able to get the details of that particular function.

(Refer Slide Time: 31:08)





Steps in Programming QEI using API

- Configure QEI Module
 - Configure it by mentioning edges to be detected, mode of operation to be used, swapping of input channels is required or not and maximum position value by using function `QEIConfigure()`
 - Enable filter and QEI module using `QEIFilterEnable()` and `QEIEnable()`
- Configure velocity capture
 - Specify velocity pre-divider and time period to measure velocity. Use `QEIVelocityConfigure()` and enable it using `QEIVelocityEnable()`
- Obtain position, velocity and direction using `QEIPositionGet()`, `QEIVelocityGet()` and `QEIDirectionGet()`.

The slide features a dark blue background with a grid pattern. In the top right corner, there is a gear icon with a circuit board inside. In the bottom left corner, there is the NPTEL logo. In the bottom right corner, there is a small video inset showing a man speaking.

Steps in Programming QEI using API

- Include library files. (driverlib/gplib.h) and Initialize variables to be used
- Initialize hardware
 - Configure system clock frequency
 - Enable QEI Module peripheral
 - Enable Port for the targeted QEI Module
 - Make the corresponding pins type as QEI module. `GPIOPinTypeQEI()`
 - Configure each pin as channel A or B or Index. `GPIOPinConfigure()`



And of course, you will have sample programs available that also kind of make quite a bit of sense. Once you have this hardware initialization part done, then you go to QEI module functions which are used for QEI module programming. So, this is one of first thing you need to do is your configuration. You look at this function, configure QEI configure and see it can be configured for mode of operation, swapping in the input channels, required or not required, maximum position value, all these kinds of things will be put in this function `QEIConfigure`.

Then filter enabling you may need to do, if you want to use filter. Then enable the QEI module also you by using `QEIEnable` kind of a function. Then you will do some kind of configuration of velocity capture, so by using this function. So, pre-divider, timer those kind of things you can set by using this velocity configuration. And then once this configuration is done, and the `QEIVelocity` or module also is `QEIVelocity` part is also enabled and QEI is enabled then, you will be obtaining position in this `QEIPositionGet` function, and `QEIVelocityGet` function.



You want to get the direction, you can get this in the, in this `QEIDirectionGet` function. So, these are the ways like, you can use these functions to get you the desired output of `QEIPosition`. So, this is makes our job quite simple. So, by directly using some functionalities, you are able to now read encoder and start using that encoder data for further control kind of a programming. If you want to use PD control or some kind of other kind of a control using this data, you will be able to do that very easily now.

So, we have seen PWM, and we have seen how this QEI interface in which by using these now one can start thinking of building entire closed loop system. We will talk about that a little bit, how do we build closed loop kind of system by doing that.

(Refer Slide Time: 33:25)

Classification of API functions used in QEI

- 📌 Configure QEI Module
- 📌 Position Capture:
 - 📌 Configuration: `QEIEnable()`, `QEIDisable()`, `QEIConfigure()`, and `QEIPositionSet()`
 - 📌 Output: `QEIPositionGet()`, `QEIDirectionGet()`, and `QEIErrorGet()`
- 📌 Velocity Capture:
 - 📌 Configuration: `QEIVelocityEnable()`, `QEIVelocityDisable()`, `QEIVelocityConfigure()`.
 - 📌 Output: `QEIVelocityGet()`
- 📌 Interrupt Handler:
 - 📌 Configuration: `QEIntEnable()`, `QEIntDisable()`, `QEIntStatus()`, `QEInt`

Sample Program

```



//Include libraries
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/qei.h"
#include "driverlib/pin_map.h"

int main(void)
{
    //Initialize Variables
    uint32_t position = 0;
    uint32_t speed = 0;
    uint32_t Direction = 0;
    
```

1. First of all you have to add library files required for the program. For using QEI Module, we need to add "driverlib/qei.h"
2. Initialize variables being used in the program
3. Initialize hardware: Set System Clock, Enable the ports you need to use. Here we will use QEI module 0 which is peripheral on Port D. Enable QEI Module and Pins to be used for QEI Module Assign specific pins as per pin diagram for index pin, channel A and B

```

//Hardware Initialization
SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); // Enable PORT D
SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI0); // Enable QEI Module 0
GPIOPinTypeQEI(GPIO_PORTD_BASE, GPIO_PIN_3|GPIO_PIN_6|GPIO_PIN_7); //Make PD3, PD6, PD7 pin type
GPIOPinConfigure(GPIO_PD3_IDX0); // Assign PD3 as Index Pin
GPIOPinConfigure(GPIO_PD6_PHA0); // Assign PD6 as Channel A
GPIOPinConfigure(GPIO_PD7_PHB0); // Assign PD7 as Channel B
    
```

4. Configure QEI
 QEI_CONFIG_RESET
 QEI_CONFIG_QUADRATURE
 QEI_CONFIG_NO_SWAP
 Enable QEI
 QEIFilterEnable
 QEIEnable
 Velocity Configure
 QEIVelocityConfigure
 QEIVelocityEnable

```
//Configure QEI
QEIConfigure(QEI0_BASE, QEI_CONFIG_CAPTURE_A_B|QEI_CONFIG_RESET_IDX|QEI_CONFIG_QUADRATURE|QEI_CONFIG_NO_SWAP, 1555);

//Enable QEI
QEIFilterEnable(QEI0_BASE);
QEIEnable(QEI0_BASE);



//Velocity Configure
QEIVelocityConfigure(QEI1_BASE, QEI_VELDIV_1, 32); //32 ms period
QEIVelocityEnable(QEI1_BASE);
```

5. Get position, velocity and direction. While loop is used for continuous operation

```
while(1)
{
  //Get QEI Position
  position = QEIPositionGet(QEI0_BASE);

  //Get QEI Velocity
  speed = QEIVelocityGet(QEI0_BASE);

  //Get Direction
  Direction = QEIDirectionGet(QEI0_BASE);
}
//return 0;
```

4. Configure QEI
 QEI_CONFIG_RESET
 QEI_CONFIG_QUADRATURE
 QEI_CONFIG_NO_SWAP
 Enable QEI
 QEIFilterEnable
 QEIEnable
 Velocity Configure
 QEIVelocityConfigure

```
//Configure QEI
QEIConfigure(QEI0_BASE, QEI_CONFIG_CAPTURE_A_B|QEI_CONFIG_RESET_IDX|QEI_CONFIG_QUADRATURE|QEI_CONFIG_NO_SWAP, 1555);

//Enable QEI
QEIFilterEnable(QEI0_BASE);
QEIEnable(QEI0_BASE);

//Velocity Configure
QEIVelocityConfigure(QEI1_BASE, QEI_VELDIV_1, 32); //32 ms period
QEIVelocityEnable(QEI1_BASE);
```

```
while(1)
{
  //Get QEI Position
  position = QEIPositionGet(QEI0_BASE);

  //Get QEI Velocity
  speed = QEIVelocityGet(QEI0_BASE);

  //Get Direction
  Direction = QEIDirectionGet(QEI0_BASE);
}
//return 0;
```




4. Configure QEI
 QEI_CONFIG_RESET
 QEI_CONFIG_QUADRATURE
 QEI_CONFIG_NO_SWAP
 Enable QEI
 QEIFilterEnable
 QEIEnable
 Velocity Configure
 QEIVelocityConfigure

```
//Configure QEI
QEIConfigure(QEI0_BASE, QEI_CONFIG_CAPTURE_A_B|QEI_CONFIG_RESET_IDX|QEI_CONFIG_QUADRATURE|QEI_CONFIG_NO_SWAP, 1555);



//Enable QEI
QEIFilterEnable(QEI0_BASE);
QEIEnable(QEI0_BASE);

//Velocity Configure
QEIVelocityConfigure(QEI1_BASE, QEI_VELDIV_1, 32); //32 ms period
QEIVelocityEnable(QEI1_BASE);
```

```
while(1)
{
  //Get QEI Position
  position = QEIPositionGet(QEI0_BASE);

  //Get QEI Velocity
  speed = QEIVelocityGet(QEI0_BASE);

  //Get Direction
  Direction = QEIDirectionGet(QEI0_BASE);
}
//return 0;
```

So, this is a classification of all these different functions. And this is a sample program. You can see this part where you have to initialize, add these libraries and get this set up here. Then, some of the variables if you want, whatever you want to use, you can initialize the variables in the main program, and then, you can initialize hardware, and you follow the steps that we just saw.

So, if you see some of these functions are doing this hardware initiation, basically here. So, you are mapping the pins, configuring them GPIO pins to act as a index pulse or A pulse or B pulse something like that. And you are enabling this QEIO module, you are enabling this GPIOD register. So, PORT D, we are enabling to kind of make sure, that the pins are PORT D, which are used for QEI interface are enabled and things like that.

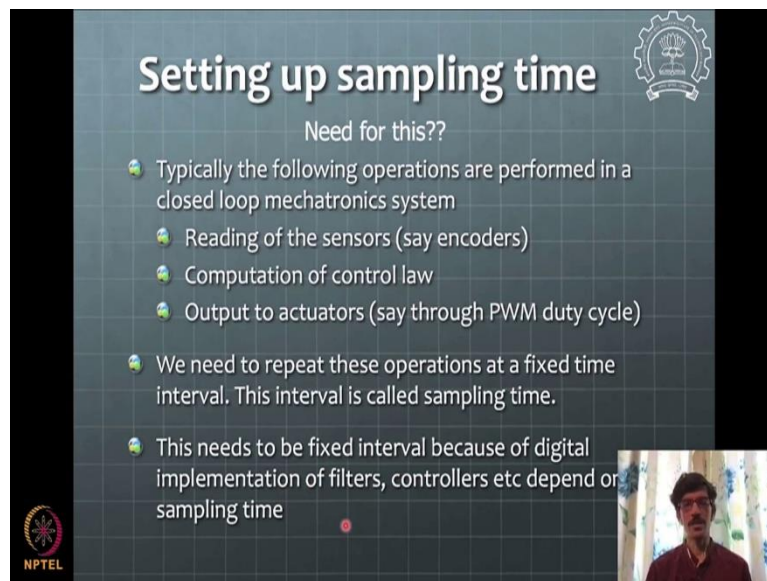
Then, you are doing this configuration of this QEI interface here. And you can see these are the details of the configuration register, QEIConfigure. You do all these different-different configuration parameters to be set here. And then you enable QEI interface. This is particularly QEIO interface, whereas, QEIO and QEI1 two interfaces will be there.

Then velocity configuration you are doing here. And once this velocity is configured, now you are, you will be able to start reading it. So, Get QEI Position, you say position is equal QEI, so, so, this is from function 0 you are using 0. So, this from interface 0 with this function by using this interface 0 value can be fetched in this position. So, we have PositionGet of QEIO port or QEIO interface you will get a encoder position of encoder which is connected to this pins corresponding to this QEIO interface.

Like that you can have a position QEI1 interface position also. So, if you are using two motors at a time, you can use this other interface as well. And then position you get position velocity and direction. So, this is how typically you go ahead, get, writing a program to get the QEI module working.

So, I would suggest that you should try this yourself with the, with this QEI module, and the Tiva Launchpad that you might have procured by now, and already started exploring different-different interfaces. So, use our motor in the kit to do this kind of programming and get a feel of what, how things are happening. And that is a best learning that will happen. If you have doubts, clear like any things you can post and we can help out with those things. So, these are module is going to work.

(Refer Slide Time: 37:12)



Setting up sampling time

Need for this??

- Typically the following operations are performed in a closed loop mechatronics system
 - Reading of the sensors (say encoders)
 - Computation of control law
 - Output to actuators (say through PWM duty cycle)
- We need to repeat these operations at a fixed time interval. This interval is called sampling time.
- This needs to be fixed interval because of digital implementation of filters, controllers etc depend on sampling time

NPTEL

(A small video inset in the bottom right corner shows a man with glasses and a beard speaking.)

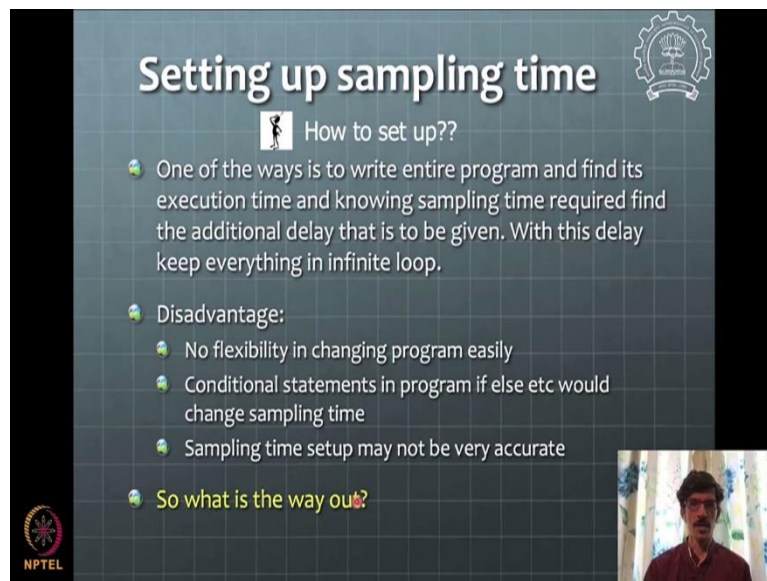
Now, next, we will look at how do we put together an entire thing now on the programming side of the microcontroller. So, theory part we will look at in the future classes to come, but we can now start putting things together once we have, once we know how do you get a position of the motor, and once we know how do you run the motor. So, we have seen both of these. So, what is that is needed is now to put together?

So, what we typically need to do is, to run these three operations in a sequence. First, is a reading of the sensors or reading of encoders, then computation of the control law based on the encoder feedback position and what we want. And then, finally, using this control law computation you get some kind of a view are output that is going to the motor that is to be implemented on a motor actuator. Typically, actuators will be actuated through PWM, and then PWM duty cycle is what you will change every sample time so to say or every cycle.

These three operations will go on in a cycle. They will keep on repeating with some kind of a time interval, this time interval called a sampling time. So, this is a very important concept in mechatronic systems, that you will have some kind of a fixed sampling time that is to be set so that these operations successfully get carried out every time interval of the sampling time.

Now, this time interval needs to be fixed because there are digital implementations of filters that happen in typically in this case, and they depend upon sampling time. So, all the implementation, digital implementations of the filters will have dependence on the sampling time. So, sampling time is very important to maintain in the system.

(Refer Slide Time: 39:23)



The slide is titled "Setting up sampling time" and features a gear icon in the top right corner. Below the title is a question "How to set up??" with a person icon. The main content consists of a list of points:

- One of the ways is to write entire program and find its execution time and knowing sampling time required find the additional delay that is to be given. With this delay keep everything in infinite loop.
- Disadvantage:
 - No flexibility in changing program easily
 - Conditional statements in program if else etc would change sampling time
 - Sampling time setup may not be very accurate
- So what is the way out?

In the bottom right corner, there is a small video inset showing a man with a beard and glasses speaking. The NPTEL logo is visible in the bottom left corner of the slide.

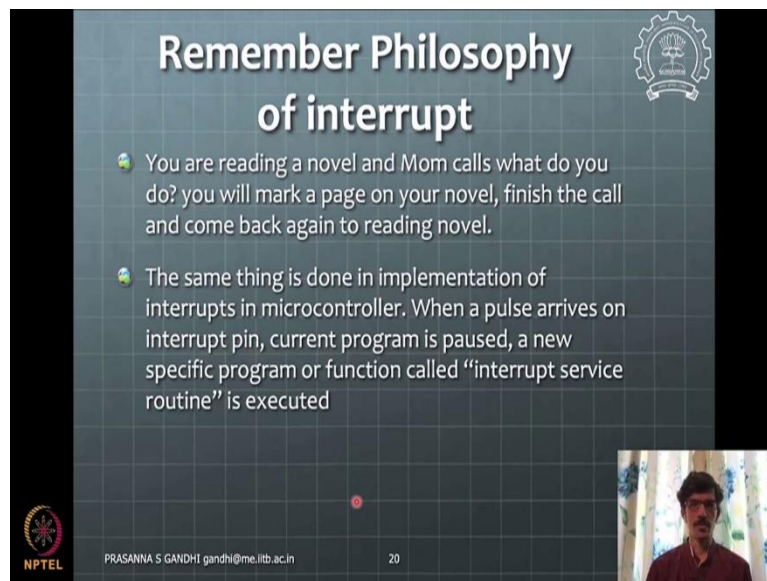
How do you maintain that? One of the ideas to do that is, how do you set up that or how do you maintain this time sampling time is to write entire program and find the execution time that it takes for this program to run. And once we know, okay, this is say it takes 1.5 milliseconds as a time to run the entire program.

And I want to set 2 milliseconds as a sampling time, then I know 0.5 millisecond needs to be a delay. So, I will put some additional delay that is by doing some kind of a delay function and doing I mean keeping microprocessor busy in some kind of additional operation in a loop for that my duration. And after this delay is performed then, again, it can start doing this next operation, next cycle.

So, this keeps on happening for infinite loop. That is one way of setting up sampling time. But the disadvantage is that, there is no flexibility in changing the program easily. You need to change program you need to change the delay also. Then there are conditional statements if there are there in the program if else kind of a program, then it will change sampling time.

So, it is a cumbersome way. And then you are keeping your microcontroller busy in doing some delay function, which is of not much of a use, it just kind of keeps microcontroller busy, so as to kind of have you give even that much kind of a delay. So, there should be some, there should be a good way out of this. We need to use some other way of doing things here. And that way is basically by using interrupts.

(Refer Slide Time: 40:59)



Remember Philosophy of interrupt

- You are reading a novel and Mom calls what do you do? you will mark a page on your novel, finish the call and come back again to reading novel.
- The same thing is done in implementation of interrupts in microcontroller. When a pulse arrives on interrupt pin, current program is paused, a new specific program or function called “interrupt service routine” is executed

NPTEL PRASANNA S GANDHI gandhi@me.iitb.ac.in 20

So, we recall, the philosophy of interrupts where say suppose you are reading a novel and your mom calls. So, you are sitting and peacefully reading the novel and then the call comes. Then you hear the bell ring, and then what you do, you put a mark in your page wherever you last you were working there, and finish the call, of course, if you like too. Many times you may or may not like to take a call, but let us say if you want to take a call, and you finished a call and come back again start reading the novel, that is what you do, exactly the same philosophy is there for the interrupt.

When the interrupt comes on some pin, then microcontroller halts whatever it is doing right now, and serve something and run some small other routine that is to be, that is programmed to run when that interrupt comes, and then again comes back to the main program after finishing that job. So, this is a kind of a philosophy of a interrupt that we have seen.

(Refer Slide Time: 42:07)

Remember Philosophy of interrupt

- After execution of ISR again main program execution is resumed. You may be doing nothing in main program, then its just a wait for another pulse to come on interrupt pin.
- This philosophy is executed along with a timer to generate interrupt pulses periodically in various microcontrollers. We will see specifics in TIVA

NPTEL PRASANNA S GANDHI gandhi@me.iitb.ac.in 21

The slide features a dark blue background with a white grid pattern. The title is in a large, bold, white font. Two bullet points are listed below the title, each preceded by a small globe icon. In the bottom right corner, there is a small video inset showing a man with a beard and glasses. The NPTEL logo is in the bottom left, and the presenter's name and email are at the bottom center. The slide number '21' is in the bottom right.

And if we have this, when you come back you may do nothing, you may just wait, to wait for the other signal to come. So, that way that much time is available for microcontroller now to do some other activities to be done. But now, the idea is now, if we get these pulses coming on this pin in some kind of a periodic manner, then this operation, same operation can happen again and again and again and again, whenever these pulses come.

So, idea is to use a timer to generate this interrupt pulses every few milliseconds or every whatever sampling time that is needed. So, if you want 2 millisecond kind of a sampling time you generate this pulse every 2 milliseconds then 1 pulse comes, this policy is connected to interrupt pin and that interrupt pin kind of receive this pulse, and once it has received the pulse it starts executing some program, which is your program, which is doing these three operations that we talked about. Reading the encoders like completing the controls and implementing to the actuator. So, this is how it really happens.

(Refer Slide Time: 43:16)

Setting up sampling time

Interrupt based method: Basic philosophy

- Use interrupts along with timer to do the job. How??
- Setup the timer to generate a pulse every "sampling instance", the pulse would trigger interrupt and start running a program called "interrupt service routine" (ISR). We write a program for these 3 tasks in ISR and program interrupts appropriately.
- We need to enable this timer for interrupt operation
- In TIVA this is done using API functions. ISR is simply a predefined named function in which we can write program pertaining to the three activities described earlier

NPTEL

So, that is what we have seen that we will generate the pulses every sampling instance or every sampling time and this pulse will trigger the interrupt and start running program called. This program that run, that we run where we have these three operations are done. These three operations we are writing, somebody can write any other kind of a thing in that interrupt service routine, but that program is called interrupt service routine.

So, this is interrupt service routine is a name that is used ISR. So, this is a terminology you need to be aware about. And the program starts running. Once, the program is finished, it will return to the main program. If there is any activity remaining to be done in the main program, it will continue, otherwise, it will just wait for the next pulse to come.

Then in Tiva, this is done using again API functions. And this interrupt service routine is simply predefined name function. So, no, nothing needs to be done in the hardware. So, all the things in the hardware are somehow connected in whatever way it is needed and you do not need to kind of worry about that.

You just need to use a timer and then like enable that timer for interrupt kind of a operation, and that is it, your job is done. So, we will see some of the steps here, but the idea here is to do this timer setting. So, typically, you will say okay this timer setting will have what all things to be done, okay I need to set a time.

I should look for something where I can set up the time. Then I should look for that okay this timer needs to be connected to this interrupt. So, it is interrupt-enabled timer should happen, and then I need to run this ISR, so I need to look for what is ISR. So, ISR in this case will be

simply some predefined name function in which, in that function if we write something that whatever we write in that function that will get executed every time they interrupt is happening.

So, this is how you set up the sampling time in modern kind of microcontroller applications. So, you are very high-fidelity or I would say much better kind of operations of closed loop control can be achieved by using this fixed sampling time setup by interrupt base philosophy.

Rather than just to using like, continuously in the loop without really giving any attention to what is a sampling time happening in your system, it is better to kind of understand what is this we set, what is the sampling time we want to set and that my sampling time is set properly for the application, your application is going to run much better with a with a high performance.

(Refer Slide Time: 46:22)

Setting up sampling time

Interrupt based method: TIVA

- In TIVA use the `TimerLoadSet()` function (see its syntax in the Tiva function syntax manual)
- For example for timer 0 `TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1)` where `ui32Period` is set according to sampling time. For example for 0.1 sec sampling time (10Hz freq) , `ui32Period = (SysClockGet()/10-1)`
Look at detailed description of these functions in the manual
- Sample program

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);  
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);  
ui32Period = (SysCtlClockGet()/10);  
TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period);  
IntEnable(INT_TIMER0A); TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); IntMasterEnable();  
TimerEnable(TIMER0_BASE, TIMER_A);
```
- Observe
`TimerIntEnable()`

NPTEL

So, this is typically like some functions that are used. So, this `TimerLoadSet` is a function used to you can look at this function, search for that and read the function in API library datasheet, and see what are the arguments to this function, and what it does and things like that.

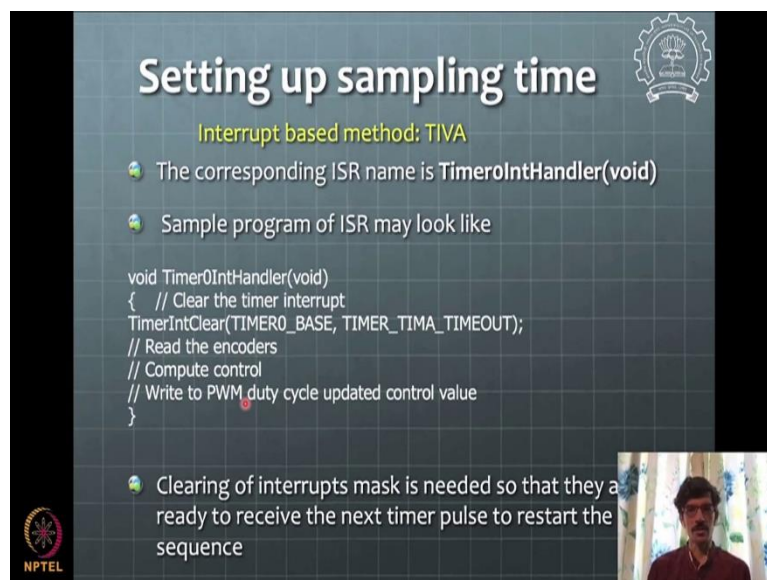
Now, for this is an example for timer 0 this `TimerLoadSet` function will have these arguments to be used, and then this is a period to be set according to sampling time. So, you have a, again, the same philosophy that, okay, so sampling time for example, to be 0.1 second or 100 millisecond with frequency or sampling frequency to be 10 hertz then this will be set this value.

Is it like a period value that is set based on the system clock, whatever is the system clock frequency divided by like this 10 minus 1. So, this is how this this number, which is to be loaded in the, it is like a loading that is happening in some register, this number will be registered in some load and it will be count to 0 and then that pulse will be generated, that is a kind of a philosophy the same way it is used as in the case of timer in the in the QEI module.

And this is a sample program that you can look at. So, this is a PeripheralEnable enabling happening for the timer peripheral enable. Then configuring this timer and then you are setting some kind of timer frequency, and then you are enabling here Interrupt Enable. So, Interrupt Enable happens here, then this timer Interrupt Enable there is some interrupt-enabling functions are there, and then the Master Enable function happens, and then you enable the timer for the timer 0 that is, what you are using for the interrupt.

So, these are some functions you need to read through the details of this function what they are doing, and understand, and start making use of them. And once you set up this interrupt kind of a routine then you need to look for what is the function in which I should write my interrupt service routine, where, and how do you, how do you look for that.

(Refer Slide Time: 48:55)



The slide is titled "Setting up sampling time" and features a gear icon in the top right corner. Below the title, it specifies "Interrupt based method: TIVA". The slide contains two bullet points: "The corresponding ISR name is Timer0IntHandler(void)" and "Sample program of ISR may look like". A code block follows, showing the implementation of the Timer0IntHandler function. The code includes comments for clearing the timer interrupt, reading encoders, computing control, and writing to the PWM duty cycle. A second bullet point states: "Clearing of interrupts mask is needed so that they are ready to receive the next timer pulse to restart the sequence". The NPTEL logo is visible in the bottom left corner, and a small video inset of a presenter is in the bottom right corner.

```
void Timer0IntHandler(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // Read the encoders
    // Compute control
    // Write to PWM duty cycle updated control value
}
```

So, the Interrupt Service Routine in this case is this particular kind of a name, Timer0IntHandler. And in this function, you write some kind of statement. So, first statement that we will write is this clear of this interrupt. So, this clearing is doing some clearing up the interrupts mask. This is done so that now the interrupt is ready for the next pulse to receive.

So, if the next pulse arrives then this will again start reading. If you do not do this command, then the operation run once and it will just kind of wait, because the clearing is not happened this clearing is important to happen. So, once a clearing happens then one can, the next pulse comes again this. So, this command if you miss then you may find that, okay, we are seeing some, your program is not running, you will wonder why. So, this is important concept here, the clearing of the interrupt mask.

So, this is how we will go ahead and put together the things now to really read the encoders then compute control and write PWM duty cycle value. And now how do you compute control based on the dynamics and control some of the theoretical aspects we will look at and give some kind of a practical implications of some of the theoretical results. And then we will discuss how do we now know.

So, we are here basically getting ourselves completely ready now to implement any kind of closed loop kind of control with a microcontroller. So, some theory we will look at, and again, we can kind of come back and do this programming or parallely, we can do this programming for some of the mechatronic applications. That is how we will go ahead. Thank you very much.