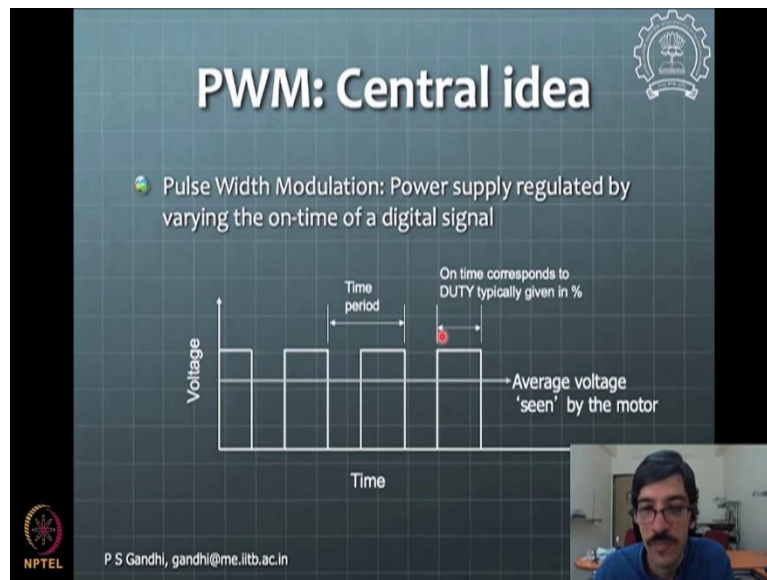


Design of Mechatronic Systems
Professor Prasanna S. Gandhi
Department of Mechanical Engineering
Indian Institute of Technology, Bombay
Lecture No 21

Interfacing Actuator using PMW in Tiva Microcontroller

So, next class, today's class, we will look at this microcontroller interfacing, specifically, the PWM interface. So, last class we have already seen some kind of a basic philosophies of these interfacing. Now, we will go a little bit more specific into the aspects of somewhat hardware kind of aspects, at more like a philosophical level and not really talking of all the details of the things, but getting the concept like how do we think through like the way things can be implemented in microcontroller. That is a kind of idea that we will slowly evolve.

(Refer Slide Time: 01:18)



So, let's begin with that. So, as you all know this PWM Central Idea is pulse width modulation just to flash this thing to make sure like you are recollecting your memory that PWM has this kind of a feature. It has some kind of a time period and some duty, which is one can change that in operation. So, this is how like, you see basic PWM signal. Now, we want to generate such signal. And what is a way one can think off in microcontroller to generate such kind of signals.

(Refer Slide Time: 01:54)

Pulse Width Modulation

- Defined by two parameters: Frequency, Duty (changing)
- Signal is shown in the figure below for various values of duty cycle
- Applications: switching power supplies, motor control, servo positioning and lighting control.

0% Duty Cycle
25% Duty Cycle
50% Duty Cycle
75% Duty Cycle
100% Duty Cycle

Average Output Voltage

NPTEL

So, there are mainly two parameters here for PWM. One is frequency, and other is a Duty. So, the frequency parameter usually is fixed parameter, and the duty is what changes continuously during operation. Like you want more power to be delivered or depending upon what your control input is or control computation is then you change this PWM duty cycle, and you can see here signals for different, different duty cycles, how they look like.

So, this is a 0 level here and this is 1 level here and 75 percent of the time this would like on time is 75 percent of the total cycle time. And then that is how you drive some kind of average voltage into the application. So, that is what would happen.

(Refer Slide Time: 02:49)

Why PWM?: Uses and Advantages

- Voltage regulation in actuators
- No effect of low amplitude noise (robustness)
- Ease of digital control implementation: just by changing the duty cycle of PWM control power input can be changed

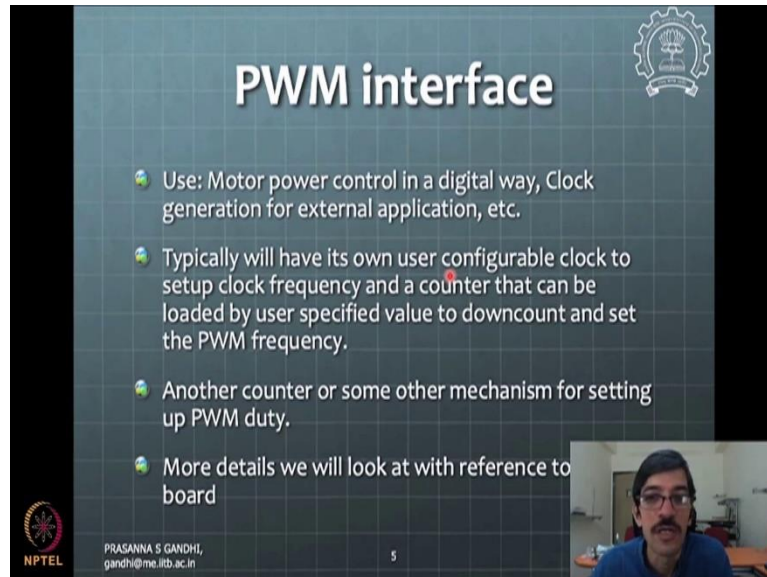
Q: How can it be generated by microcontroller?

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

NPTEL

Now, we already have talked about these. Why we need PWM and all these details to make their advantages, because it is a digital signal and there are no noise effect that would come. We are just reiterating some of these things that we have already seen. Now, the question is how this PWM can be generated by a microcontroller?

(Refer Slide Time: 03:12)



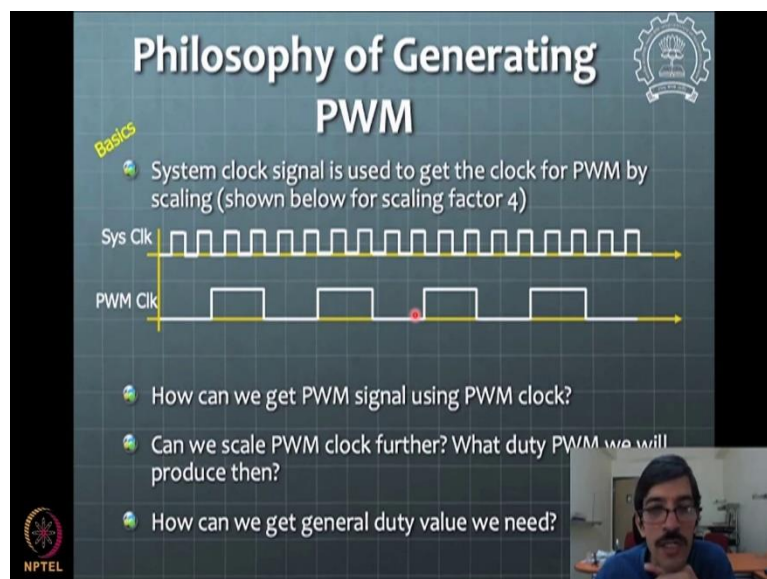
PWM interface

- Use: Motor power control in a digital way, Clock generation for external application, etc.
- Typically will have its own user configurable clock to setup clock frequency and a counter that can be loaded by user specified value to downcount and set the PWM frequency.
- Another counter or some other mechanism for setting up PWM duty.
- More details we will look at with reference to board

NPTEL PRASANNA S GANDHI, gandhi@me.iitb.ac.in 5

So, we start off with a microcontroller clock. So, PWM signal is some kind of a periodic signal. So, clock is the best thing to begin with. So, the clock's frequency is if you scale it down you will get a PWM signal or you get a some kind of a frequency controlled signal of a clock or frequency controlled clock or clock at PWM frequency, like that. But is that sufficient?

(Refer Slide Time: 03:52)



Philosophy of Generating PWM

Basics

System clock signal is used to get the clock for PWM by scaling (shown below for scaling factor 4)

Sys Clk

PWM Clk

- How can we get PWM signal using PWM clock?
- Can we scale PWM clock further? What duty PWM we will produce then?
- How can we get general duty value we need?

NPTEL

So, say for example, you start off with the system clock and scale it down. See, here you can see that this is a system clock coming, and I have here 4 kind of a clock pulses covered in one cycle. So, I just put some kind of a scaling factor of 4 and I get this PWM clock signal. So, this is one of the ways to kind of produce the clocks or produce some signals, which are at a lesser frequency than the clock frequency, you use some kind of a scaling factor. Now how that happens at hardware and all we are not getting into the details.

Now, the question is, once you have this some kind of a clock signal for PWM, then how do you get a duty cycle for that? So, that is what is can you think about like just a minute we will pause here and think about it and then like you proceed. You can pause here and think about. Suppose you have this PWM clock signal, which is running at a frequency which is some kind of a clock frequency for PWM interface. This is not really PWM frequency.

So, we want to kind of get two things. One is a PWM frequency out of this PWM clock so PWM clock will typically be running at very high kind of a frequency. So, system clock for example in TIVA is 80 megahertz, so, PWM clock will be scaled down maybe by a factor of 4 or 6 or 32 but some factor. So, that will generate PWM clock. Now, using that clock, we want to kind of get our PWM frequency and somehow that duty cycle. Can you think of a way to do that?

So, see, because if we just scale this clock further down, and get PWM frequency clock signal that will have a fixed duty, that will have 50 % like a duty cycle. So, scaling may not be a way, directly. Because scaling will kind of give you only a 50 % duty cycle value. So, let us proceed with and see, now, what is that we can do?

(Refer Slide Time: 06:19)

Philosophy of Generating PWM

Basics

- PWM clock is further scaled to get PWM frequency using count down of a counter and a part is used for PWM duty

Counter Loaded with = 10 Duty count = 3

30 % duty cycle Resolution 10 %

- Signal can be left or right justified, some control options are provided. Counter can be only down count or up count and some such options could be there

NPTEL

Programming PWM in Tiva

- Two methods (in general for all interfaces)
 - Using directly registers to be set
 - Control registers
 - Working registers
 - Using API or library function already defined (also defined in ROM of Tiva for fast execution)
- We will focus on 2nd method

PRASANNA S GANDHI gandhi@me.iitb.ac.in 8

NPTEL

So, here, you can see that, this is our PWM clock signal. So, this is already scaled kind of a clock signal. So, I have drawn this scale in a different kind of manner here. So, our clock signal was looking like this, but now, our PWM clock is looking like this. So, as we said time scaling is done here to kind of illustrate the aspect.

So, now, you start off here with a counter, which is a counting pulses. It counts 1, 2, 3 like that it counts, and it counts with say 10 pulses, and then you do something. You say that is your period for PWM. So, the okay signal clock period will be something like 10 counts of the clock here. And then, further if you say my duty will be out of these 10 pulses, like duty will be 3 pulses, so then I can get like a 30 % duty cycle kind of a value. So, I need these two numbers once we have a PWM clock.

One is like the value which will be corresponding. So, this is somewhat kind of a scaling factor you can say. I count these many numbers of cycles to generate one PWM cycle. So, 10 PWM clock cycles will give one PWM cycle. And out of that 3 counts or 3 cycles will correspond to a particular duty.

So, so here now, since I am using this number 10 here my resolution of PWM. Can you think like what is the resolution that is happening here? The resolution here you can see that, I can have only one clock cycle as a duty, as a minimum count for the duty. And it will correspond to out of 10, 1 that means it is like a 10 % resolution. So, I cannot say, I want PWM signal of 5 % duty cycle. No, I cannot produce with this particular kind of a thing. Then what I will need to do is, I will need to kind of increase this factor.

So, this is a very important understanding of basics that look, if you want higher resolution what you are supposed to do. So, if you want, given some kind of a scenario and we say that, in this scenario, I want the best possible resolution. What should I do? So, I need to kind of then have this PWM clock's frequency much higher value.

So, we see, if it is higher value, then I will have a lot of numbers because PWM, frequency of PWM is fixed even by the application. So, to have more cycles in the same kind of a period of the PWM frequency I will need a clock, PWM clock frequency to be higher. So, like that one can start thinking. So, this is basic kind of idea.

Then you can have additional controls that you have left or right justified signals or the signal is delayed by some with respect to some other signals something like that, then counter can be only counting down or counting up or all those kind of things can be possible. So, this is counter-loaded with 10. Now these 10 can be counted up or content down. And then depend upon that something will happen, but basic concepts is basically these for getting the PWM signal produced in the system.

(Refer Slide Time: 09:53)

The slide is titled "Programming PWM in Tiva" and features a list of two methods. The first method is "Using directly registers to be set", which is further divided into "Control registers" and "Working registers". The second method is "Using API or library function already defined (also defined in ROM of Tiva for fast execution)". A note at the bottom states "We will focus on 2nd method". The slide includes logos for NPTEL and IITB, and the presenter's name and email address: PRASANNA S GANDHI, gandhi@me.iitb.ac.in. A small video inset in the bottom right corner shows the presenter, a man with glasses and a blue shirt, speaking.

So, for programming, if you go into now a little bit more programming details like if there are two methods. So, again, we have talked about this philosophically in general. But with respect to Tiva also you can have direct programming by control registers. This XCP 100 code, we write based on like the registers. So, you fill in these registers with some values, and then like something will get executed.

So, there are these two types of registers that we have seen, control registers and working registers. For each interface there will be some control registers, which you need to be initiated for configuration. They are just initiated once, and then working registers are maybe getting continuously updated by, say for example, in the PWM case, PWM frequency is fixed. So, it is a part of that control register setting. And then working register is PWM duty, which is changing as per the control competition, every sampling instance. So, that will be like a working register.

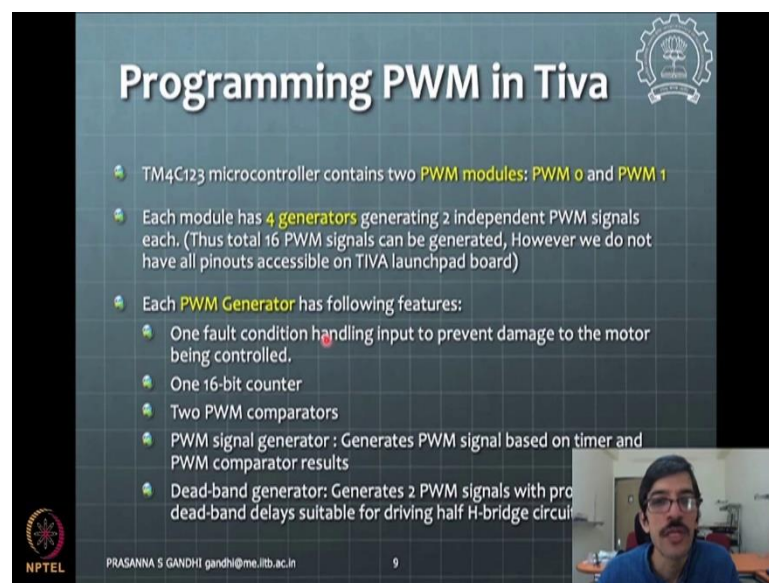
And then second philosophy of method of having the programming done is using API or library functions that are already defined or already given by this Tiva ware. So, you have seen that Tiva ware library has a lot of these functions written. The header files some variables are defined and then some functions addition as functionalities can be directly used. So, there is a manual which gives you the details of how these functions are and which is already posted in Moodle.

So, some of these functions are also defined in the ROM of a Tiva for execution, which is going to be very fast. Because then the compiler will not have to face these functions from your hard disk and then compile and dump this entire compilation to Tiva memory. You will

save some memory if these functions on the ROM are directly used, because they are already there in the Read-Only Memory part of the Tiva board.

So, if you have started using those functions, then, you do not need to download these functions again during compilation and put them onto the Tiva board or a memory of the microcontroller. So we will focus on the 2nd method, but we will get into some kind of insights into how this module particularly in Tiva is configured or working. So, you can understand, we will not get into control and working registers in detail, but just get a just kind of a feel for how this is till it done in Tiva.

(Refer Slide Time: 12:55)



The slide is titled "Programming PWM in Tiva" and features a grid background. It contains the following text:

- TM4C123 microcontroller contains two PWM modules: PWM 0 and PWM 1
- Each module has 4 generators generating 2 independent PWM signals each. (Thus total 16 PWM signals can be generated, However we do not have all pinouts accessible on TIVA launchpad board)
- Each PWM Generator has following features:
 - One fault condition handling input to prevent damage to the motor being controlled.
 - One 16-bit counter
 - Two PWM comparators
 - PWM signal generator: Generates PWM signal based on timer and PWM comparator results
 - Dead-band generator: Generates 2 PWM signals with pro dead-band delays suitable for driving half H-bridge circuit

At the bottom left is the NPTEL logo, and at the bottom center is the text "PRASANNA S GANDHI gandhi@me.iitb.ac.in" and the number "9". A small video inset of a man is visible in the bottom right corner of the slide.

So, Tiva, like if you see the datasheet you will see that it has two PWM modules, PWM 0 and PWM 1, correspondingly there will be pins which are named, and each module will have 4 generators, generating 2 independent PWM signals. So, how many total PWM signals will be there? 16 PWM signals can be generated there. So, 2 modules, 4 generators and each generator generate 2 signals. So, 8 signals is what will be generated by one module and you have two modules each module has 4 generators. So, you have a 16 PWM signals that can be generated.

And, although, I must caution you that all 16 may not be available for the Tiva launchpad that we have been working with. Because only some few signals will be available for programming different interfaces, because otherwise there will be too many pinouts that have to be done on the board and that will be too cumbersome for the board.

So, it has like some kind of a bare minimum base minimum functionalities that they have provided out. Although the chip that is there existing on the board will have all this capacity, but the hardware pins that are gotten out from the board will not be all the pins. There will be some selected pins have been taken out. So, then, each PWM generator has all these kind of features. So, you can read through these and see, if these things can make sense.

(Refer Slide Time: 14:31)

Programming PWM in Tiva

Each PWM Generator Block diagram →

The diagram illustrates the internal structure of a PWM Generator Block. It starts with a PWM Clock input that feeds into a Control block (containing PWMCTL) and a Timer block (containing PWMLOAD and PWMPCOUNT). The Timer block outputs a 'time base' signal to two Comparators (PWMCMP0 and PWMCMP1). The Comparators output 'cmpa' and 'cmpb' signals to a Signal Generator (containing PWMGENA and PWMGENB). The Signal Generator outputs 'pwm0' and 'pwm1' signals. The Signal Generator also feeds into an Interrupt and Trigger Generator (containing PWMINTEN, PWMTRIG0, and PWMTRIG1). The Interrupt and Trigger Generator outputs 'pwm0' and 'pwm1' signals to a Fault Condition block (containing PWMFLT0, PWMFLT1, PWMFLT2, PWMFLT3, PWMFLT4, and PWMFLT5). The Fault Condition block outputs 'Fault0' and 'Fault1' signals. The Signal Generator also feeds into a Dead-Band Generator (containing PWMDB0, PWMDB1, and PWMDB2), which outputs 'pwm0' and 'pwm1' signals. The final outputs are 'pwm0' and 'pwm1'.

Think for example what you can do and cannot do? With re...
typical PWM generator

PRASANNA S GANDHI, gandhi@me.iitb.ac.in 10

Programming PWM in Tiva

- TM4C123 microcontroller contains two PWM modules: PWM 0 and PWM 1
- Each module has 4 generators generating 2 independent PWM signals each. (Thus total 16 PWM signals can be generated, However we do not have all pinouts accessible on TIVA launchpad board)
- Each PWM Generator has following features:
 - One fault condition handling input to prevent damage to the motor being controlled.
 - One 16-bit counter
 - Two PWM comparators
 - PWM signal generator : Generates PWM signal based on timer and PWM comparator results
 - Dead-band generator: Generates 2 PWM signals with pro dead-band delays suitable for driving half H-bridge circuit

PRASANNA S GANDHI, gandhi@me.iitb.ac.in 9

So, how do you kind of make sense of these things? So, to make sense of like some of these write-up or functionality details, you need to refer to the block diagram. This is a block diagram of this. You see this PWM generator block. So, this is a block diagram for this PWM generator block and in that there are these different- different kinds of components here. There is a timer, there is a comparator, then there is something called signal generator, and

then dead band generator and then you will get your PWM signals out for these particular generator block.

So, each module, you remember, has 4 such generator blocks. We are just looking at one generator block, which will generate these two PWM signals A and B. And input, it will take as a PWM clock. So, by seeing this diagram you should think what you can do, what you cannot do. Can you think about that, pause for a moment and think about what you can do, and what you cannot do?

So, one can see that the PWM clock is same for one particular PWM generator block or for that matter this clock may be going in other blocks also in the same way. So, you cannot have independent clock for these PWM A and B signals. Because the PWM clock will be same for both the signals or for that matter all the generators. So, like that, one can start kind of like concluding some things.

Then there is some kind of a timer, which will do some further kind of a timing for this clock and then there will be comparator, so we will see how these things work. So, the signals will go typically, like, this is a flow of the signals then you need to start with a timer and then timer.

These signals will go to comparator or some timer signals will also go to the signal generator. And then from comparator you have these two outputs coming compare A compare B and they use some kind of a timing part here also and produce this PWM A and PWM B signals and then those signals are further processed by these dead band generator block and here. So, this is a kind of a signal flow is what one can understand. So, like that you can think about, and you can have your own conclusions drawn. Some of these conclusions I have written in the next slide.

(Refer Slide Time: 17:07)

PowerPoint Slide Show - [11_mac_PWM/Tiva]

Programming PWM in Tiva

- Each **PWM Generator** Block diagram → What can be done cannot be done?
- For example:
 - For each generator there is only 1 PWM clock so for two signals coming out of same generator the PWM clock frequency is same
 - PWM signals however can have Independent frequency if the count is specific to a channel

Think for example what you can do and cannot do? With respect to a typical PWM generator

NPTEL

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

11

What this means

I need to get this pointer again here. So, you can read through the some of these conclusions. So, PWM signals can have independent frequency or PWM A and B signals that are coming they would have, if the count is specific to the channel. So, you need to check whether there is count for the specific channel is same or same count is given for both channels something like that if you check that data, and you will be able to see PWM signal frequencies can be changed by changing the count for each specific channel.

(Refer Slide Time: 17:54)

PowerPoint Slide Show - [11_mac_PWM/Tiva]

Functional Description PWM

- PWM Clock:** Can use system clock itself or apply pre-divisor to system clock to obtain PWM clock. Clock source is selected by programming Run-Mode Clock Configuration (RCC) register.
- PWM Timer:** PWM Timer has two modes; Count-Down & Count-Up/down
 - In count-down mode, timer counts from load value (predefined) to zero and immediately goes to load value and continues the cycle. Used for Left or Right aligned PWM signals.
 - In count-up/down mode, timer counts from load value to zero, then zero to load value and continues the cycle. Used for center-aligned PWM signals
 - Timer has 3 output signals
 - 1. Direction Signal: 1 when counting up and 0 when counting down
 - 2. Load Signal: Becomes 1 (for single clock cycle period) when timer is at load value
 - 3. Zero Signal: Becomes 1 (for single clock cycle period) when timer is at zero value
- PWM Comparators:** Each PWM generator produces 2 PWM comparator signals
 - Used to monitor counter value
 - When either of the comparators matches with counter, it produces high pulse (for single clock cycle period)

NPTEL

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

12

What this means

PowerPoint Slide Show - [11_mac_PWM.tiva]

Programming PWM in Tiva

Each PWM Generator Block diagram →

Think for example what you can do and cannot do? With respect to a typical PWM generator

PRASANNA S GANDHI gandhi@me.iiit.ac.in

NPTEL

10

So, these are some functional descriptions that will be given. So, these are directly taken from their datasheet, and you can see whether these things make sense to you. You can read through. So, this is just to kind of give you some feel of the typical interspaces will have this kind of description, and how to make sense of this description to see how things are working inside internally.

Say, for example, if you are directly focused on the API's and functions you may or may not need to know so many details. You will be able to kind of program even without knowing these details. But we need to know some of these to do little finer programming. So, once you have some things going you want little more finish control signals that are generated. You need to understand these and then see what is possible, what is not possible.

So, if you understand this. For example, there is, these modes of timer, which has countdown timing a timer mode or count up and down timer mode. So, in count down mode, it will only count down and then at the end of the count or wherever this count becomes 0, it will again load the same value and again start counting from that to 0.

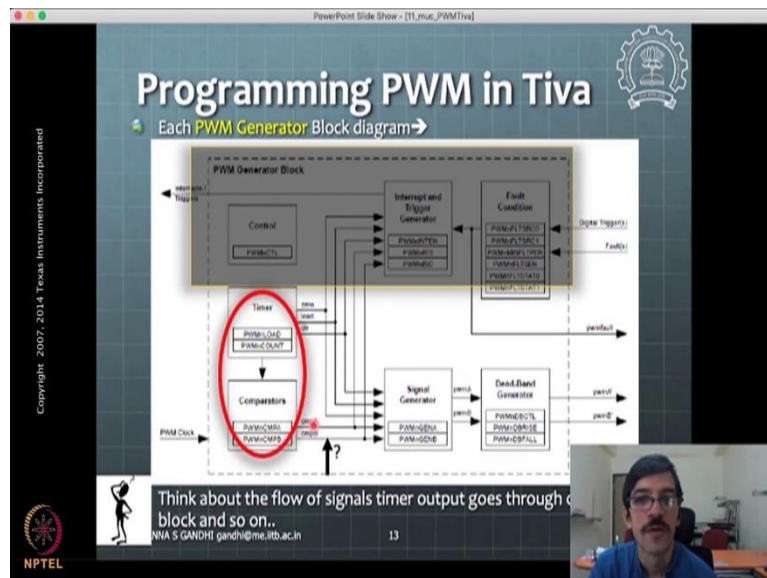
So, this kind of things is one kind of a mode of operation for timer. Other mode, we will start off with counting up and then again in count down. Again, count up from 0 and then again count down, like that it will start. So, how do you, how do they use this further, then it will have some more kind of a details that are given.

Then there is a direction signal for counting up and down. So, this signal will be useful typically when we are doing count up and down mode, so that you know, which is a direction getting going ahead with so that signal will be coming out of the block.

So, if you see there will be some direction signal coming out to the counter block somewhere. See, this is a direction signal from the timer. So, like that you can look at the diagrams with respect to the description that is given. So, let us now get into a little bit more kind of a meaning of some of these things in terms of how things happen inside the timer.

So, this is like functional description for different-different blocks now, we are going through details. And first is a timer block, and timer block has these two modes of operation. And then there is a comparator block, which will keep, like, whenever this count happens to be the value which is given in the comparator block then it will produce some kind of a pulse that is what a function of comparator block is. This will be more, clear, when we see the actually the figure of this.

(Refer Slide Time: 21:03)



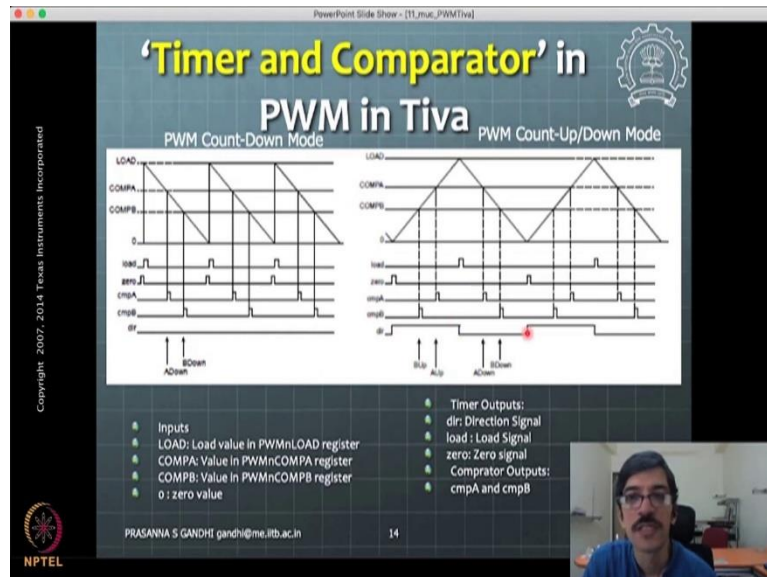
So, we are looking at these two blocks the timer and comparator, they are typically working in sync with each other. The timer keeps on like giving the count, which is kept on reducing as for every PWM clock cycle. So, PWM clock cycle takes and the counter reduces by 1. Next tick the counter reduces it by another value. Like that, it keeps on decreasing successively the value of the count.

And when the count becomes equal to say, this PWM compare A signal then some kind of a pulse will be produced on this compare A output, and same thing will happen with this beyond. What is the signal that will look like on these lines, is that what we will look now.

Can you think typically, suppose, I am having a down timer, then like, if I say, when the countdown value becomes equal to this particular value PWM CMPA then I get some small

clock pulse on these compare A signal, otherwise this compare A signal is going to be 0. Given this description, can you think and plot, for example, how these signals are going to look like? That is exercise.

(Refer Slide Time: 22:40)



So, this slide now actually gives you this data. So, this is a value that is loaded, some value which is loaded in the counter and it starts counting it down every clock cycle one countdown will happen. And then it goes to 0 and again it starts adorning the same value, so this is like a countdown mode-only. You are not counting up any time. Only loading the fresh value, and again counting down, loading fresh value and again counting down.

So, a similar kind of thing happens when we go for the count up and down. Where A is here now, you will have a count up. First, you have reached a load value then you start counting down, and these signals are going to look like that.

Now, whenever this compare A and compare B values are hit during the countdown process, of course, for that, compare A compared B values have to be lesser than this whatever value you have loaded in the counter. If it is more than that, I mean there will be some error that will be popped up or it may consider the highest value here. So, we do not know what is (I mean) that must be, like, we need to see through the details of the datasheet to kind of see what happens in under such kind of scenarios.

Now, what happens when these compare A signal is reached by the counter the CmpA on that signal you will get these little pulse, clock pulse, and same thing happens with compare B

signal. On the compare B signal you get this little clock pulse. And now these clock pulses are used further for processing and generating PWM.

But you can see that, similar kind of thing happens with the PWM countdown. So, wherever this counter hits on the up count it will generate pulse, and on the down count also it will generate pulse on these CmpA, CmpB signals. So, these are the kinds of signals that will be getting generated. And then this is a direction counter. So, this is a direction counting up, is positive, and in counting down is 0. So that is how these things work inside.

(Refer Slide Time: 24:59)

PowerPoint Slide Show - [1].mac_PWM[Tiva]

Programming PWM in Tiva

Each PWM Generator Block diagram →

Think about the flow of signals timer output goes through block and so on..

PRASANNA S GANDHI gandhi@me.iitb.ac.in 15

NPTEL

PowerPoint Slide Show - [1].mac_PWM[Tiva]

'Timer and Comparator' in PWM in Tiva

PWM Count-Down Mode PWM Count-Up/Down Mode

- Inputs
 - LOAD: Load value in PWMnLOAD register
 - COMPA: Value in PWMnCOMPA register
 - COMPB: Value in PWMnCOMPB register
 - 0: zero value
- Timer Outputs:
 - dir: Direction Signal
 - load: Load Signal
 - zero: Zero signal
 - Comparator Outputs: cmpA and cmpB

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

NPTEL

Now you are using these further in this block, which is signal generator block. So, these signals are going into signal generator block. And this signal generator is using like some of

these and converting that into this PWM A and B signals here. So, how that happens? You can see here.

Say for example, if you have this kind of pulses coming, how can I generate PWM out of these pulses? So, say for example, this load corresponds to like some kind of a frequency that could be my PWM frequency. And then this compare A value whatever I am writing that can kind of give you some kind of a handle over what should be my PWM duty cycle, things like that. So, one can think about that kind of a logic to do the signal generation in the signal generator block. So, you can see here what is happening now in Tiva.

(Refer Slide Time: 26:04)

The slide is titled "PWM 'Signal Generator'" and contains the following text:

- It takes load, zero, cmpA and cmpB as input and generates two PWM signals pwmA and pwmB
- It uses events such as zero, load, match A up, match A down, match B up, match B down depending on the mode(count-down or count-up/down)

The timing diagram shows a triangular wave for the LOAD signal. The COMP A signal is a square wave that is high during the rising slope of the LOAD signal and low during the falling slope. The COMP B signal is a square wave that is high during the falling slope of the LOAD signal and low during the rising slope. The pwmA signal is high when COMP A is high and low when COMP A is low. The pwmB signal is high when COMP B is high and low when COMP B is low.

Text on the slide: "Signal generation in count up-down mode", "Copyright 2007, 2014 Texas Instruments Incorporated", "NPTEL", "PRASANNA S GANDHI, gandhi@me.tlb.ac.in", "Similar exercise in count down mode and then figure will be the duty cycle 16", and a small video inset of a person speaking.

So, you can see in signal generator whenever this compare B is reached B signal that pulse now is getting converted into a trigger, which will make this pwmB signal go high. So, this value is low, this signal goes high. And when this same value is reached on the down count then again this value is made low.

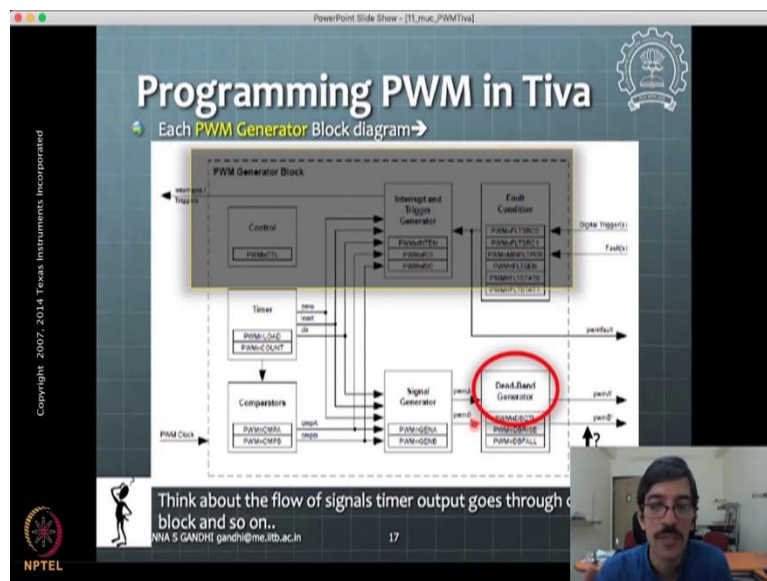
So, that is how this compare A, compare B signals work and produce pwmB signal in the signal generator block. So, those little pulses that are produced by the comparator block are used in some way to create some kind of a logic circuit, which will get this PWM signal B out in such a way that moment if this value is compared matches, count value matches pwmB signal is made high.

Same thing for PWM signal A. Whenever the value matches from 0 the pwmA signal will be turned to 1 and then whenever on the down count the value matches this signal will be turned to 0, if this logic is used. Now, you do an exercise. Now same logic if it is happening for the

sawtooth waveform, which was like no countdown signal-only. So this was a countdown kind of a signal-only then what is going to happen. So, you can think about.

What kind of logic needs to be there, such that, like compare A value will be somewhat changing this binary signal from 0 to 1 or 1 to 0 or whatever. So, you think about that, and think what kind of logic can be there? So, we will leave it right now. I am not going to explain you that, but using the same, extending the same concept one can think about, (why) what will happen to these pwmA and B signals in the countdown mode alone. And that is what probably we will be using for our PWM generation.

(Refer Slide Time: 28:24)



So, that is what is that we need to think about. And then there are these additional facilities that dead band generator. Suppose you want to have some dead band introduced in the signal at the start or falling edge of the of the PWM signal then how do you do that. Normally, if this is not enabled this signal directly comes as output pwmA and pwmB signals.

So, for one generator you get now this complete flow from timer to comparator to signal generator, and then your actual signal now. So, this particular part will not get into more details here. There are some things, which probably we may not be needing to use, but you can kind of go through the datasheet and like a little bit kind of understand how this control and the interrupts in PWM 1 if fault conditions also can be can be displayed.

So, you can go and look at that those details if at all you are interested in more into that, but we may not need that PWM interrupt kind of mode of operation of PWM. What we want to kind of do is, generate these PWM signals and be able to control the duty cycle continuously,

so that, we can deliver in the actuator in mechatronic systems, right amount of energy that is needed.

(Refer Slide Time: 29:58)

The slide is titled "Dead-Band Generator" in yellow text. It contains two bullet points: "If disabled, then pwmA & pwmB signals will be forwarded without modification" and "If enables, then pwmB signal is lost and pwmA signal is used to produce new 2 signals (pwmA' and pwmB') by adding programmable delays as given in figure." Below the text is a timing diagram with three signal lines: pwmA, pwmA', and pwmB'. pwmA shows a square wave. pwmA' is a square wave that is delayed at both the rising and falling edges compared to pwmA. pwmB' is a square wave that is zero during the dead-band periods of pwmA. The diagram is labeled with "Rising Edge Delay" and "Falling Edge Delay". At the bottom, there is a presenter's video feed and a text box that says "This understanding along with knowledge of correspo you will need to get for register level programming! B APIs instead to begin with". The slide also includes logos for NPTEL and Texas Instruments.

This slide is identical to the one above, showing the "Dead-Band Generator" concept with a timing diagram and a presenter's video feed. It includes the same text, diagram, and logos.

So, this is a dead band generator with a little more details about, you can go through. So, if it is disabled and you do not have these shifting of the signals happening, but if there is enabled and then they depend upon the value that is given in some registers, you will have some delay produced in pwmA prime signal, which is the actual output of the PWM.

So, these signals will get delayed a little bit. So, this is like a, this part is called a dead band. So, you can have a rising edge delay or you can have a falling edge delay and it can bring some signals. So, this much understanding along with some kind of a particular knowledge of some registers. So, this register knowledge will be given in this, if you see this datasheet.

specific value to be placed on the **INVERT** signals during a fault condition, that value is inverted if the corresponding bit in this register is set.

PWM Output Inversion (PWMINVERT)

PWM0 base: 0x0028000
 PWM1 base: 0x0029000
 Offset: 0x00C
 Type: RW, reset: 0x00000000

BitField	Name	Type	Reset	Description
31:8	reserved	RO	0x00000000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7	PWM7INV	RW	0	Invert INVERT 7 Signal Value Description: 0 The INVERT 7 signal is not inverted. 1 The INVERT 7 signal is inverted.
6	PWM6INV	RW	0	Invert INVERT 6 Signal

Tiva™ TM4C123GHFPM Microcontroller

BitField	Name	Type	Reset	Description
1	GLOBALSYNC1	RW	0	Update PWM Generator 1 Value Description: 0 No effect. 1 Any current update to a load or comparator register in PWM generator 1 is applied the next time the corresponding counter becomes zero. This bit automatically clears when the updates have completed. It cannot be cleared by software.
0	GLOBALSYNC0	RW	0	Update PWM Generator 0 Value Description: 0 No effect. 1 Any current update to a load or comparator register in PWM generator 0 is applied the next time the corresponding counter becomes zero. This bit automatically clears when the updates have completed. It cannot be cleared by software.

June 12, 2014

Texas Instruments—Production Data

1243

Pulse Width Modulator (PWM)

Offset	Name	Type	Reset	Description	Page
0x0C4	PWMINTEN	RW	0x0000.0000	PWM2 Interrupt and Trigger Enable	1271
0x0C8	PWMRES	RO	0x0000.0000	PWM2 Raw Interrupt Status	1274
0x0CC	PWMISC	RW1C	0x0000.0000	PWM2 Interrupt Status and Clear	1276
0x0D0	PWMLDAD	RW	0x0000.0000	PWM2 Load	1278
0x0D4	PWMCOUNT	RO	0x0000.0000	PWM2 Counter	1279
0x0D8	PWMCMPA	RW	0x0000.0000	PWM2 Compare A	1280
0x0DC	PWMCMPB	RW	0x0000.0000	PWM2 Compare B	1281
0x0E0	PWMCENA	RW	0x0000.0000	PWM2 Generator A Control	1282
0x0E4	PWMCENB	RW	0x0000.0000	PWM2 Generator B Control	1285
0x0E8	PWMDRCTL	RW	0x0000.0000	PWM2 Dead-Band Control	1288
0x0EC	PWMDRRISE	RW	0x0000.0000	PWM2 Dead-Band Rising-Edge Delay	1289
0x0F0	PWMDRFALL	RW	0x0000.0000	PWM2 Dead-Band Falling-Edge Delay	1290
0x0F4	PWMLTSRCD	RW	0x0000.0000	PWM2 Fault Source 0	1291
0x0F8	PWMLTSRC1	RW	0x0000.0000	PWM2 Fault Source 1	1293
0x0FC	PWMMINFLTPER	RW	0x0000.0000	PWM2 Minimum Fault Period	1296
0x100	PWM3CTL	RW	0x0000.0000	PWM3 Control	1296
0x104	PWM3INTEN	RW	0x0000.0000	PWM3 Interrupt and Trigger Enable	1271
0x108	PWM3RES	RO	0x0000.0000	PWM3 Raw Interrupt Status	1274
0x10C	PWM3ISC	RW1C	0x0000.0000	PWM3 Interrupt Status and Clear	1276
0x110	PWM3DADR	RW	0x0000.0000	PWM3 Load	1278

Table 20-2. PWM Register Map (continued)

Offset	Name	Type	Reset	Description	See page
0x0C0	PWMDCTL	RW	0x0000.0000	PWM2 Control	1296
0x0C4	PWMINTEN	RW	0x0000.0000	PWM2 Interrupt and Trigger Enable	1271
0x0C8	PWMRES	RO	0x0000.0000	PWM2 Raw Interrupt Status	1274
0x0CC	PWMISC	RW1C	0x0000.0000	PWM2 Interrupt Status and Clear	1276
0x0D0	PWMLDAD	RW	0x0000.0000	PWM2 Load	1278
0x0D4	PWMCOUNT	RO	0x0000.0000	PWM2 Counter	1279
0x0D8	PWMCMPA	RW	0x0000.0000	PWM2 Compare A	1280
0x0DC	PWMCMPB	RW	0x0000.0000	PWM2 Compare B	1281
0x0E0	PWMCENA	RW	0x0000.0000	PWM2 Generator A Control	1282
0x0E4	PWMCENB	RW	0x0000.0000	PWM2 Generator B Control	1285
0x0E8	PWMDRCTL	RW	0x0000.0000	PWM2 Dead-Band Control	1288
0x0EC	PWMDRRISE	RW	0x0000.0000	PWM2 Dead-Band Rising-Edge Delay	1289
0x0F0	PWMDRFALL	RW	0x0000.0000	PWM2 Dead-Band Falling-Edge Delay	1290
0x0F4	PWMLTSRCD	RW	0x0000.0000	PWM2 Fault Source 0	1291
0x0F8	PWMLTSRC1	RW	0x0000.0000	PWM2 Fault Source 1	1293
0x0FC	PWMMINFLTPER	RW	0x0000.0000	PWM2 Minimum Fault Period	1296

In the datasheet, you can get all the gory details about say registers. Say for some examples of this register block you can take and from this previous block diagram, and then search for those names, you will get these register details. So, as you see these master-controlled registers. So, a lot of many, many different registers will be there and each research all the details will be given.

So, this is like a registered map, all the registers are given here. So, you can go through these registers and make some familiarity. If, you directly want to use these registers in program, great. You can do that thinking if I understand all the registers. Now that I understand its main philosophy, if I further understand the register details, I will be able to control this

register-based programming or I will be able to generate this PWM signal at the register kind of a level.

This is how, we will go about programming for the register level. But we will do the programming, mainly, in API's to begin with. So, later, we may not need also to get into the register level programming. But maybe as a part of some academic exercise, we can take some simple thing to do program by using the register level details, so that we have that grip over the programming of this maximum controller. So, for typically for using this API or library for programming, we will need to do something some process needs to be followed or some steps need to be followed.

(Refer Slide Time: 32:54)

PowerPoint Slide Show - [11_mac_PWM1v1]

Other Blocks

- Interrupt and Trigger Generator:**
 - It takes load, zero, cmpA and cmpB as input to generate an interrupt or ADC trigger
 - It can either use any one of the event (load, zero, matchA up etc) directly or some set of these events as a source for interrupt or ADC trigger.
- Fault Conditions:**
 - It signals the controller to stop PWM functions and set PWM signal to safe state.
 - Two basic situations causing fault conditions:
 1. Microcontroller is stalled and cannot perform the necessary computation in the time required for motion control.
 2. External error or event is detected
- Output Control Block:**
 - This takes care of final conditioning of signals pwmA and pwmB before going to the pins

NPTEL

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

19

So, we will look at those steps now. These other block details, I will leave it to you to kind of go through and understand, if at all, the situation or needs comes in your project this may be useful.

(Refer Slide Time: 33:08)

PowerPoint Slide Show - [11_mac_PWMTrw]

Steps for programming with API lib functions

- Include library files. (driverlib/pwm.h) and Initialize variables to be used
- Initialize hardware
 - Configure system clock frequency. `SysCtlClockSet()`
 - Enable PWM Module to be used
 - Enable Port corresponding to the targeted PWM Module
- Configure PWM Module
 - Configure PWM Clock using `SysCtlPWMClockSet()`
 - Make the corresponding pins type as PWM module. `GPIOPinTypePWM()`
 - Configure each pin as channel A or B or Index. `GPIOPinConfigure()`

NPTEL

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

20

So, these are the typical steps that are there for the programming with the library functions. So, you need to make sure, that your, these header files are there in the library. So, in driver library, you will have pwm.h and pwm.c files, they will be there, and they will initialize a lot of variables that are needed for this. So, the register level programming will, you will need to know some of the base addresses and the values of the hardware register addresses, which will be already put in this some of these header and library files, then you do not need to know those detailed names and description.

So, you just need to know, these are some of these functions, how they work and things will be fine. So, this function description is given in the other file that is given to you. All these functions are given, their description is given. Maybe, we will see, if we can figure out one of the, such files.

(Refer Slide Time: 34:20)

Copyright © 2007, 2014 Texas Instruments Incorporated. All rights reserved. http://www.ti.com/.../products/2271MAC123104609

Table 20-2. PWM Register Map (continued)

Offset	Name	Type	Reset	Description	See page
0x00	PWMDCTL	RW	0x0000.0000	PWM2 Control	1266
0x04	PWMDINTEN	RW	0x0000.0000	PWM2 Interrupt and Trigger Enable	1271
0x08	PWMDRIS	RO	0x0000.0000	PWM2 Raw Interrupt Status	1274
0x0C	PWMDISC	RWIC	0x0000.0000	PWM2 Interrupt Status and Clear	1276
0x00	PWMDLOAD	RW	0x0000.0000	PWM2 Load	1278
0x04	PWMDCOUNT	RO	0x0000.0000	PWM2 Counter	1279
0x08	PWMDCMPA	RW	0x0000.0000	PWM2 Compare A	1280
0x0C	PWMDCMPB	RW	0x0000.0000	PWM2 Compare B	1281
0x00	PWMDGENA	RW	0x0000.0000	PWM2 Generator A Control	1282
0x04	PWMDGENB	RW	0x0000.0000	PWM2 Generator B Control	1285
0x08	PWMDDBCTL	RW	0x0000.0000	PWM2 Dead-Band Control	1288
0x0C	PWMDDBRSE	RW	0x0000.0000	PWM2 Dead-Band Rising-Edge Delay	1289
0x00	PWMDDBFALL	RW	0x0000.0000	PWM2 Dead-Band Falling-Edge Delay	1290
0x04	PWMDFLTSRCD	RW	0x0000.0000	PWM2 Fault Source 0	1291
0x08	PWMDFLSRC1	RW	0x0000.0000	PWM2 Fault Source 1	1293
0x0C	PWMDMINFLTPER	RW	0x0000.0000	PWM2 Minimum Fault Period	1296

NPTEL

Copyright © 2007, 2014 Texas Instruments Incorporated. All rights reserved. http://www.ti.com/.../products/2271MAC123104609

21 Pulse Width Modulator

Introduction
API Functions
Programming Example

21.1 Introduction

Each instance of a Tiva PWM module provides up to four instances of a PWM output control block. Each generator block has two PWM output signals, **independently or as a pair of signals with dead band delays inserted**. Each has an interrupt output and a trigger output. The control block determines the signals and which signals are passed through to the pins.

Some of the features of the Tiva PWM module are:

- Up to four generator blocks, each containing
 - One 16-bit down or up/down counter
 - Two comparators
 - PWM generator

NPTEL

Copyright © 2007, 2014 Texas Instruments Incorporated. All rights reserved. http://www.ti.com/.../products/2271MAC123104609

- Dead band generator
- Control block
 - PWM output enable
 - Output polarity control
 - Synchronization
 - Fault handling
 - Interrupt status

This driver is contained in `driverlib/pwm.c`, with `driverlib/pwm.h` declarations for use by applications.

21.2 API Functions

Functions

- `uint32_t PWMClockGet (uint32_t ui32Base)`
- `void PWMClockSet (uint32_t ui32Base, uint32_t ui32Config)`
- `void PWMDeadBandDisable (uint32_t ui32Base, uint32_t ui32Gen)`
- `void PWMDeadBandEnable (uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Period)`

NPTEL

This driver is contained in driverlib/pwm.c, with driverlib/pwm.h for declarations for use by applications.

API Functions

Functions

- uint32_t PWMClockGet (uint32_t ui32Base)
- void PWMClockSet (uint32_t ui32Base, uint32_t ui32Config)
- void PWMDeadBandDisable (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMDeadBandEnable (uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Fall)
- void PWMFaultIntClear (uint32_t ui32Base)
- void PWMFaultIntClearExt (uint32_t ui32Base, uint32_t ui32FaultInts)
- void PWMFaultIntRegister (uint32_t ui32Base, void (*pfnIntHandler)(void))
- void PWMFaultIntUnregister (uint32_t ui32Base)

Functions

- uint32_t PWMClockGet (uint32_t ui32Base)
- void PWMClockSet (uint32_t ui32Base, uint32_t ui32Config)
- void PWMDeadBandDisable (uint32_t ui32Base, uint32_t ui32Gen)
- void PWMDeadBandEnable (uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Fall)
- void PWMFaultIntClear (uint32_t ui32Base)
- void PWMFaultIntClearExt (uint32_t ui32Base, uint32_t ui32FaultInts)
- void PWMFaultIntRegister (uint32_t ui32Base, void (*pfnIntHandler)(void))
- void PWMFaultIntUnregister (uint32_t ui32Base)

Pulse Width Modulator (PWM)

Introduction
 API Functions
 Programming Example

1 Introduction

Each instance of a Tiva PWM module provides up to four instances of a PWM generator block. Each generator block has two PWM output signals, which independently or as a pair of signals with dead band delays inserted. Each generator block has an interrupt output and a trigger output. The control block determines the polarity of the signals and which signals are passed through to the pins.

Some of the features of the Tiva PWM module are:

- Up to four generator blocks, each containing
 - One 16-bit down or up/down counter

So, let us see it here. The Tiva functions here. So, these pulse width modulation if you see this base, they have all API functions given, and then programming example also is given. So, you can use that to get started with this. So, see this driver is contained in this pwm.c and pwm.h file. So, there are a lot of functions. These functions directly, reading those function details you may not get to this procedure.

So, for procedural aspect what functions to be used and when or what is the sequence of using some functions, that is a little bit tricky to figure out. But you may be able to figure out based on the block diagram understanding that I need to first initialize this, first initialize that and then do this, then do that or you can say, see the directly here the examples that are given, programming examples that are given, that will kind of give you a better understanding.

(Refer Slide Time: 35:44)

Steps for programming with API lib functions

- 1. Include library files. (driverlib/pwm.h) and initialize variables to be used
- 2. Initialize hardware
 - 1. Configure system clock frequency. `SysCtlClockSet()`
 - 2. Enable PWM Module to be used
 - 3. Enable Port corresponding to the targeted PWM Module
- 3. Configure PWM Module
 - 1. Configure PWM Clock using `SysCtlPWMClockSet()`
 - 2. Make the corresponding pins type as PWM module. `GPIOPinTypePWM()`
 - 3. Configure each pin as channel A or B or Index. `GPIOPinConfigure()`

NPTEL
PRASANNA S GANDHI,
gandhi@me.iiit.ac.in
29

So, here, I am just giving you some kind of a detailed steps for the programming this PWM interface. So, you need to initialize the hardware, so, clock frequency needs to be set and enable some PWM model that are to be used and enable the port corresponding to the target PWM module. So, these are kind of steps.

Now, I will not give you the entire thing here. So, you need to figure out based on some of the steps that are listed here, and the understanding that is given in the PWM. So, it is a little bit of a skill or art to master like seeing appropriate information at appropriate place. See, if you start reading the datasheet or the entire chapter on PWM things will become too lengthy procedure and you may not need a lot of such things.

So, once you are given a task that we have specified in our assignment 2 where you want to run a motor by using PWM, so to get to that task you need to generate PWM signals on two signals, two pins actually. And one pin will be connected to one of the motor inputs and other pin will be connected to other motor inputs. There are two motor digital inputs that are to be given for the motor driver.

So, the motor driver datasheet also will be uploaded. So, these are different, different things. The important part here is, you need to configure this PWM module. There are many different kind of aspects of configuring. So, configuring the clock, so we saw that block diagram needs a clock, so this is done by this command.

Then, you want to this pin size, as you have seen in the couple of last classes also, same pin is used for many different purposes in Tiva. So, in Tiva our many microcontrollers is for that matter. So, you need to specify that, I want now to use this pin as for the PWM kind of a generation. So, I will say, is a pin as a part of GPIO, but I want to use it for PWM purposes. So, that you need to specify somewhere. So, that is where you need to make the pin type as like PWM pin.

So, once you configure this as a PWM pin. I mean, you typically will not use this as a input output pin again in that application. Because, your hardware is connected to this pin now. So, you cannot say I will now make it use in my same program. Once sometimes it is used as a PWM pin and sometimes it is used as a digital input, output pin, that may not happen here because your hardware is fixed there.

So, you need to kind of make a choice of which PWM modules you need to use or given the number of pinouts and model application needs, we need to kind of make a choice of which PWM model you need to use and which kind of pin you need to configure as PWM pin. So, that those decisions are fairly simple you will be able to see the pin maps and from that maps you will be able to match your application needs versus module which pins are available. Then, you need to configure pin for channels, different channels.

(Refer Slide Time: 39:28)

PowerPoint Slide Show - [11_mac_PWM/Tiva]

Steps for programming with API lib functions

- Configure QEI Module
 - Configure PWM generator to be used in program using `PWMGenConfigure()`
 - Set the period for the PWM using `PWMPeriodSet()`
 - Enable PWM Output state using `PWMOutputState()`
 - Enable PWM Generators using function `PWMGenEnable()`
- Configure PWM Output
 - Set the width of PWM signal to get desired output. This can take value from zero to PWM period, and it will decide the duty cycle

NPTEL

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

21

And then, so it is PWM generator configuration, so now we know what is PWM generator in the block. So, in that, you need to configure it PWM generator. So, it will typically have whether this will have a count up or count down, those kinds of things will be somewhere specified. These are QEI module. This is PWM all these are function for PWM modules.

Set the period of the PWM, and then you enable finally the PWM. So, like that you will have these different kinds of functionalities which are regarding the configuration of module, and then you need to have some functionalities which are regarding some output of the PWM. So, like that you set a PWM period, we set somewhere PWM duty and get your program going.

So, I will leave you with this kind of understanding. And you can see now, the example program and other thing to figure out how things are really working. What are this each of these functionalities might be doing. You play around with the parameters and see, how they are affecting your PWM generation signal and all those kind of things you can play around and do it yourself.

So, that is where we will stop here now. And then, so further we may look at a little bit like this a little more details about quadrature encoder interface as well, because these are the two primarily important interfaces for us, to use from Tiva system, so we will do this small kind of a presentation for quadrature encoder interfaces as well.

And then we will start off with the modeling part of the thing. And while modeling we will start exploring how these models really work in practice. So, that is our future direction to go. So, here we will start getting basic understanding about how to run the motor. Once we are

starting running the motor, then we will have understanding about how to read the motor position by using encoders.

So, now you will get really-really high-fidelity position output from the position sensor. Now, we want to use this for further control. So, this is how, we will proceed. So, before going into control, we will start sensing this position and use it for some identification of parameters of the system. Particularly, we will look at friction identification in the motor system. So, this is how this future direction is what we will go. So, we will proceed like that, and I will stop here for now.