Design for Internet of Things

Prof. T V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science-Bengaluru

Lecture - 32
Power Optimization

Folks, welcome back. I am going to tell you one or two important instructions okay, specific to ARM controllers, because ARM is a very powerful controller and it has invaded several offerings from the IP. ARM IP processor is there integrated into several systems. One of the first things that you may want to know, again this is a module on low power and therefore, everything is around low power.

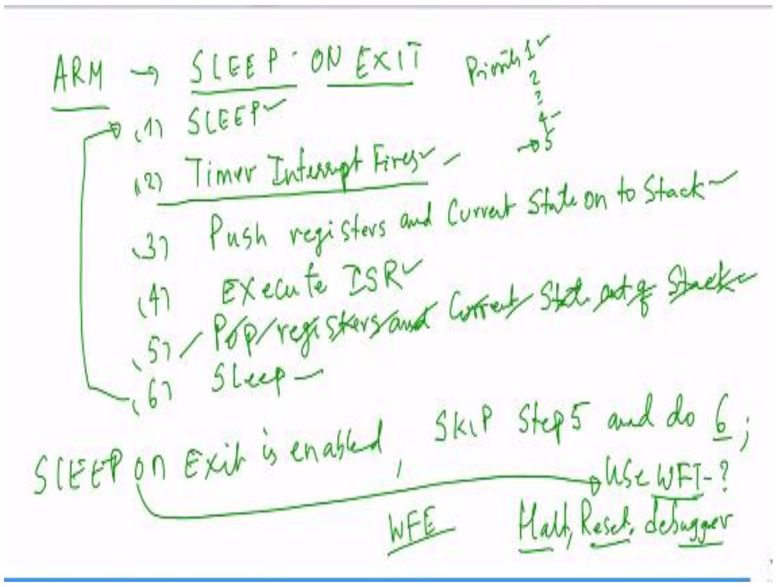
One important you know feature of ARM is related to what is known as sleep on exit okay, sleep on exit. Now this sleep on exit feature, you have to understand in the context of what you might have studied in your engineering, okay? What happens when there is an interrupt? The first thing that happens when there is an interrupt is that the current state of the controller and the registers are pushed to stack, correct?

After you push it to stack, you go and execute the instructions that is the interrupt service routines are executed. These are nothing but the ISR or interrupt handler, okay. So you go and finish the interrupt service routine. After you finish the interrupt service routine, what do you do? You have to come back and get back the current state of the processor. And you also have to get back the register status, right?

Whatever was there before entering the interrupt service routine, you have to get it back. So there is this operation of I am sleeping, controller is sleeping, there is an interrupt given to the controller, controller now has to go and service the interrupt. So push the current state, push the registers, enter the ISR, finish the ISR, come back, get back the current state, come back and go to sleep.

This is what you would do. Why I keep saying sleep. Because in IoT systems, let us say there is only an intermittent signal coming in. Most often you are in sleep because you want to conserve on battery life, okay? So you are beginning with let us say sleep. I put down this instructions here. Look at what happens.

(Refer Slide Time: 02:49)



ARM → SLEEP-ON EXIT

Priority 1 2 3 4 5

(1) SLEEP

(2) Timer Interrupt Fires

(3) Push registers and Current State on to Stack

(4) Execute ISR

(5) / Pop/registers and Current State out of Stack

(6) Sleep

SLEEP on Exit is enabled, Skip Step 5 and do 6;

Use WFI-?

WFE

Halt, Reset, debugger

You begin with sleep. Then you have timer interrupt, which fires. You push registers and current state on the stack. You execute the ISR. You pop the registers and current state out of the stack. And you go back to sleep. That means this is going back here, okay. But ARM says, Hey guys, why are you doing all of this? You started with sleep. Your timer fired. You pushed the registers and current stack, the state on stack.

You executed the ISR. Why do you want to pop? Just keep this stage 5, step 5. Just go and sleep again. You do not need to go and pop it back. Do not do that. Of course a lot of it will depend on your application. But the point is, it gives you this fantastic feature and saying shave this off. You do not need to do this, if you enable sleep on exit, okay. So sleep on exit is enabled means skip 5 and go after 4.

What do you do? You go and do, immediately go and do 6. This is important, okay? This is sleep on exit. Now to this sleep on exit, I am going to add a little more spice, okay? What is that spice I am adding on it? That is related to another command in ARM which is called WFI, okay? WFI is shown here. Wakeup on interrupt, okay. It is, you can look up the instruction set. You will see WFI.

WFI simply means you can invoke it in sleep on exit. Normally it is chosen along with sleep on exit. Why is, what is the importance and significance of WFI? Let us say the timer fired. To begin with you had the third highest priority. So you had priority 1, 2, 3, 4 and 5 for interrupt. This is only an example, okay? Let us say you went into timer interrupt fires when you were because of this interrupt, which had a priority of 3, okay?

That means you enter this with priority 3. Now WFI says, if you come with 1 when I am sleeping, I am not going to wake up. If you come with 2, I am not going to wake up. But if you come with 4, I will wake up. If you come with 5, I will wake up. In other words, whichever interrupt you entered sleep on exit, if you come with a priority higher than that, you can wake up.

If it is lower than that, it is impossible for you to get it out of sleep on exit. So if you say it is impossible to get it on sleep on exit, how are you ever going to get the ARM processor out of sleep on exit? There are only three ways by which you can do folks, there are only three ways. Supposing you entered, think of the situation, you entered with 5, you entered with 5 that is the highest, right?

You entered with 5, there are no there cannot be anything higher than 5, and you are gone on sleep on exit and you have invoked WFI, finished. You cannot get the ARM processor out of sleep on exit. It is going to just sleep there. Very nearly it is going to sleep. No interrupt is ever going to wake it up. It become like Kumbakaran, okay? So that situation you will end up.

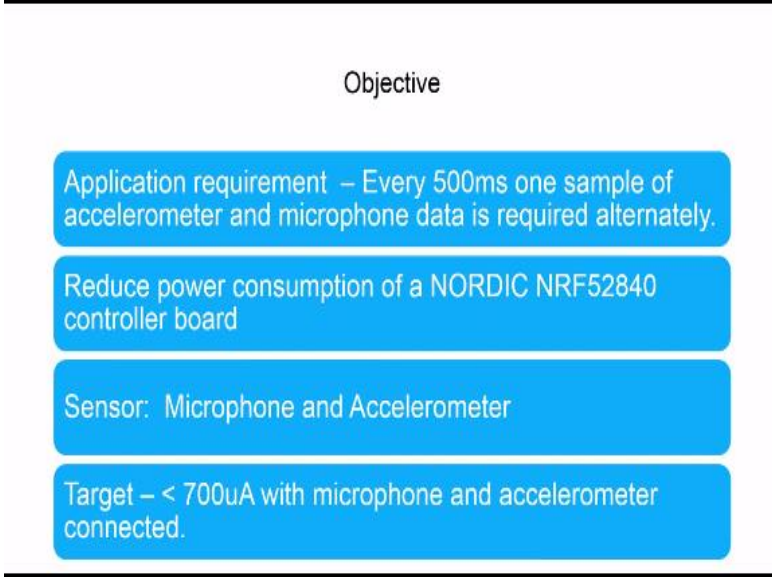
Careful therefore on what was the interrupt that invoke sleep on exit? This is important, okay. Now there are three ways by which you can get it out of sleep on exit only, if you have entered with the highest interrupt, and what are the three? I have written it here; halt, reset, debug. Any one of them you give, you reset, you can come out. If you execute halt, you can come out.

If you connect a debugger and you give something from the debugger you can come out. Not in any other way. Not in any other way. So folks, this is fantastic, right? So read the ARM manual and each time connect to low power. This is important. All these are essentially the hooks that you have, that will allow you to design, to write extremely good software, particularly when you exploit the instruction set and the features of that particular controller that you are using.

So this is important. All right. Having said this, you may want to look up other things. There is something called WFE as well. Look up WFE. Understand this and we can discuss this anytime you think that you would like to, you have questions on that we can handle them separately. Great. Now what we will do is we will shift to a demo, but you must know what that demo is, what is the application that we designed in the lab.

And how do we show this low power features for this particular Nordic controller that we have of interest. For that what I will do, I will take you to this slide deck, okay.

(Refer Slide Time: 08:56)



Objective

Application requirement – Every 500ms one sample of accelerometer and microphone data is required alternately.

Reduce power consumption of a NORDIC NRF52840 controller board

Sensor: Microphone and Accelerometer

Target – < 700uA with microphone and accelerometer connected.

And I will put it in full screen mode. Here is what you want to do, okay. There is an application requirement. Every 500 milliseconds, one sample of accelerometer and microphone is required alternatively. That means in this 500 millisecond, you want accelerometer, the next 500 milliseconds you want the microphone and it goes on in that way. Essentially one this one that, one this and one that, okay.

The second thing is you want to reduce the power consumption of the Nordic NRF52840 controller board. Your sensors as we discussed are microphone and accelerometer. And what is the target you have 700 microamps with microphone and accelerometer connected. This is what it should be consuming if you configure your controller very well. Now how do you start?

You will end up with 10s of milliamperes of current, okay? And that is not going to be easy for you to optimize unless you know where are the hooks by which you can pull. And those hooks will give you low power features. All right. So let us see what exactly happens in this system. And what is this system? Let me show you the system first.

(Refer Slide Time: 10:28)

Here is a ball, there is a ball. And here is a controller board, okay. There is a controller board, if you see inside there is a PCB, okay? That PCB is essentially a microphone, it has a microphone. And on this board mainboard, you have an accelerometer, okay. And so there is one accelerometer, and then there is one microphone. But one microphone is not going to help.

We need three microphones. I will tell you why. Supposing I bundle all of this. In fact there are three, you can see. There is one microphone here, which you see. And then there are two microphones here, to the left, and to the right, okay. There are three microphones. And what is this application? And what is its importance? And why should you do any power optimization for this application?

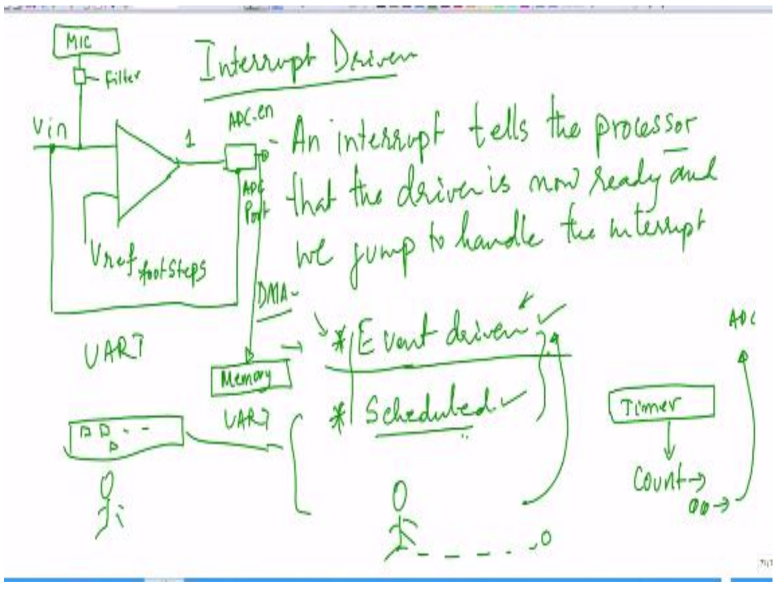
Think about all of this bundled inside this ball, you screw it up, and then you throw it, with a battery inside you throw it, okay. Now think about it. First of all, there is no GPS, I have not put any GPS here. You know the location from where you threw it, okay. From here, you have to somehow find out, where does it land? And how does it roll? What is the terrain on which it rolls, and roughly where does it sit?

Where does it rest, okay? You must be able to find out from the accelerometer, number one. That means roughly. You have to note that this is very rough. You will not be off depending on the terrain, right, if it lands on a terrain, and it is flat after it lands, then you only know it is rolling. But if it lands in a valley, just at the tip of it, then simply slides down and you do not know where it is.

So let us make some good assumptions, some practical assumptions. It lands, rolls and halts. That you may want to do with an accelerometer. Now once it halts there, you want to detect footsteps. You want to detect footsteps. Now that means you have to adjust this for no external noise, only footsteps, human footsteps. It should be thud, thud, thud big sound should be there.

Only then it should recognize that and pick that signal. And then use it for, give it to further processing. Folks, now you can think of what I mentioned little while ago about the different ways by which you write your code.

(Refer Slide Time: 13:30)



Interrupt Driven

An interrupt tells the processor that the driver is now ready and we jump to handle the interrupt

* Event driven
* Scheduled

Recall we spoke about interrupt driver. And we said there are two ways. One is event driven. And the other is scheduled, okay. Clearly you can see that footsteps to be detected, does not come under scheduled folks. You do not know when the human is going to cross, right? Here is the ball. And here is the human and he is walking towards the ball. Actually ball should be shown small, right?

It should be like this, because it is on the floor. Yeah, he is walking this way. Let us put down some reasonably good proportions. This is the size of the ball. You do not know when he is going to come. Therefore it should be event driven. Definitely it should be event driven. You do not know when it is going to happen, which clearly indicates how do you write such a code?

How do you ensure that event driven code is written even in your system? Well folks, the SOCs that you know very well have highly integrated components, highly integrated components. Please note that. One of the most important interesting component that you can think of is the comparator, okay. Comparator gives you a 1 or a 0 at the output. Simplest you can think of is a comparator is like this.

You apply a V ref and you apply the Vin here, okay. And you may essentially want to find out whether you want a 1 or a 0 right here, okay. Now what you should do is you should set the V ref that it is significantly high that only footsteps will generate a signal. Now where is V ref coming from? You have to program that voltage. V in is coming from microphone, okay.

Now I will just take this. It is all very abstracted pictures folks, do not catch me on this. I am just showing you microphone. I will put a filter, okay. Filter is very important because you do not want noise. Your Vin is coming here. If Vin is significantly high and it compares with V ref then you will get a 1. If you get a 1, you will take that 1, 1 or 0 I wrote because it depends on whether you want to invert or you want a straight out, okay.

Anyway that I will not put that here. I will just show you that. Let us make an assumption that is you need a 1 here, okay. This one I will connect to what? I will connect it to ADC enable. If I connect it to ADC enable, ADC will get enabled. What should I do? I should take, let me write it small, V ref footsteps I will say, V ref footsteps okay, I wrote here. This is given to enable.

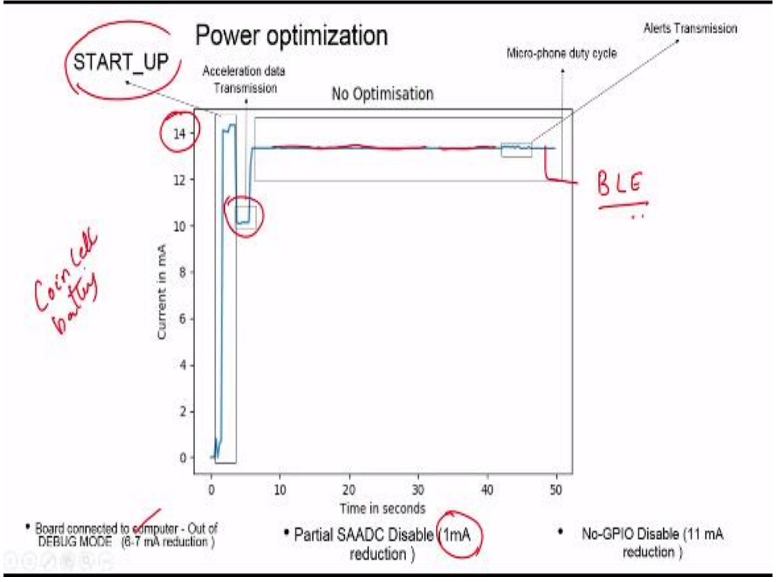
Now analog input has to be given directly is it not? So what do you do? Tap out, take it like this and connect it to the ADC port. Then you will get the code word here, clear? This is event driven in a way. Only if there is a footstep you will get an event and only if you get an event you will do ADC signal processing.

Therefore, this is the picture of interest that you may have to design your hardware such that the microphone, only if it crosses a certain footsteps threshold you will be able to detect. Alright, so this is an important aspect. And you can take these things forward folks by connecting it to the next bullet. What did we say about DMA? We said DMA can be enabled between a peripheral and memory.

So what do you do? Your code word output can be transferred to a memory location, memory using DMA. CPU is not there in the picture at all, okay, CPU is sleeping all the time. Fantastic, right? That is where you get low power benefits. And that is what essentially you can configure one of the channels of the DMA.

Typically you will get three or four channels, you can use one of the channels and you can connect them accordingly. That is the trick that you can apply to ensure that you do enable event driven processing of interrupts. Alright, so let us go back and look at this picture here again. Having done that little bit, let us start all over here. Our target is 700 microamps and we have seen the piece of hardware.

(Refer Slide Time: 19:13)



Now this picture is essentially a bad picture, a glim picture, a gloomy picture, which is not anywhere close to your 700 microamperes, correct? So now you see, what is actually happening? You will see different sections. These are by the way, actual measurements that Vasanth and others have done in the labs. Please note, these are actual measurements. I am not cooking up any result here, okay, not cooking up anything here.

Let us look at different regions. There is one region here. Then there is another region here. There is another region here. And of course this whole thing is one region, right? What have you done there with those regions? Let us see what the region is. This region is startup, he has already mentioned here. Now this is acceleration data transmission, this is acceleration data.

That means the accelerometer is giving some data. After startup it has settled down here and then it is essentially sending out acceleration data. Then it has settled down to something close to 13 milliamperes or so and then it is all the time on there. And every time there is a footstep, you get one jerky motion here, which is essentially microphone giving you some signal and then you transmit the alert back to the network, right?

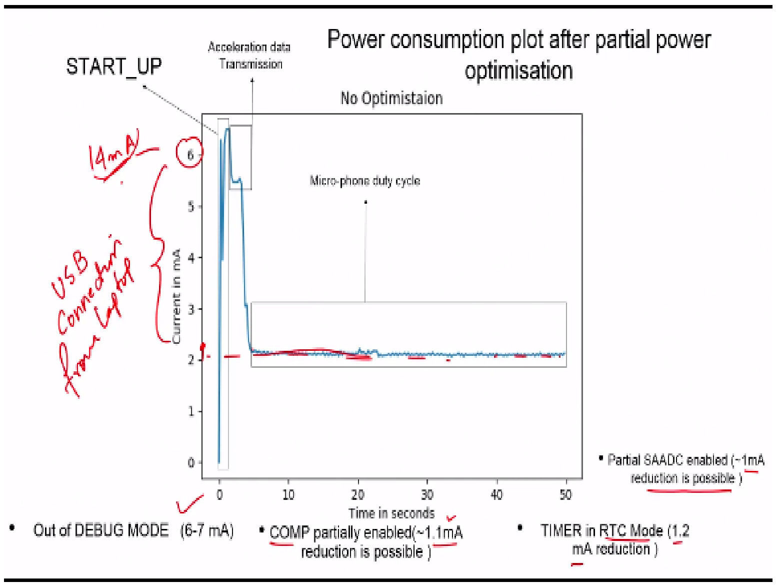
From this ball you enable something like Bluetooth Low Energy or one of the communication modules and you do a transmission. So you can see that this is pretty high you know current consumption and your battery will run out of energy in no time. But the good news is you can do fantastic amount of optimization and three points come to our mind.

First point is if you connect the debugger, the debugger consumes certain amount of current, which is close to 6 to 7 milliamperes more current because the debugger is connected and it is in the debug mode, okay? This can be shaved off because after you finish your programming part, anyway you would not be connecting any debugger. So therefore, straight away you can reduce by 6 milliamperes, okay?

So 6 to 7 milliamperes can be reduced. Whoever it is, however dumb your program is, you can move it down by anywhere between 6 and 7 milliamperes. The second part is the about the ADC, okay. If you disable the ADC, like the tricks that I showed you earlier, then you can shave off 1 milliampere of current okay, you can reduce it by an additional 1 milliampere.

So 6, 7 milliamperes from here, 1 milliampere from here. And if you take all the GPIO pins and you disable those GPIO pins, humungous amount of savings you will get, about 11 milliamperes of current gets reduced, okay. So that is what would actually happen.

(Refer Slide Time: 22:37)

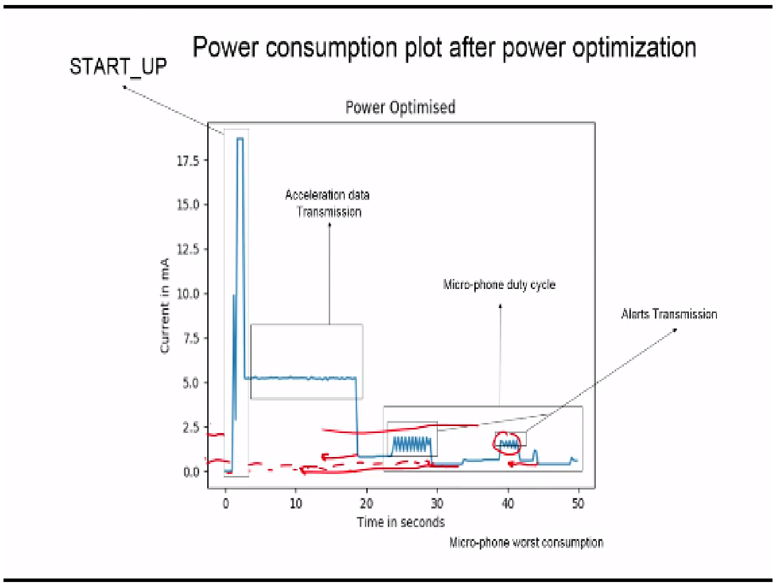


Now you see, this is a nice picture. It is back to the same one that you know. This region is the startup. This is acceleration data being transmitted, but your steady state current has now come down drastically, but the microphones are sort of sensing. There are three microphones, right? Any one of them may be sensing and you need to move between the different microphones.

So you will be, microphone will be on some sort of a duty cycle and that is what essentially you are showing here, right. So this is very good, because you are, you saved quite a bit and let us see what all we have done. For this picture to come you have removed the debugger. So 6 to 7 milliamperes are removed. Comparator is partially enabled.

So it is taking only enabled, which means it is roughly 1.1 milliampere reduction is what we have done with respect to comparator and timer is in RTC mode, okay. The timers which are used inside this program are in RTC mode. There is a 1.2 milliampere reduction and partial SAADC enabled, which means 1 milliampere reduction is possible. So you can see that several nice things are done in order to push the steady state current consumption down to an extremely small value.

(Refer Slide Time: 24:09)



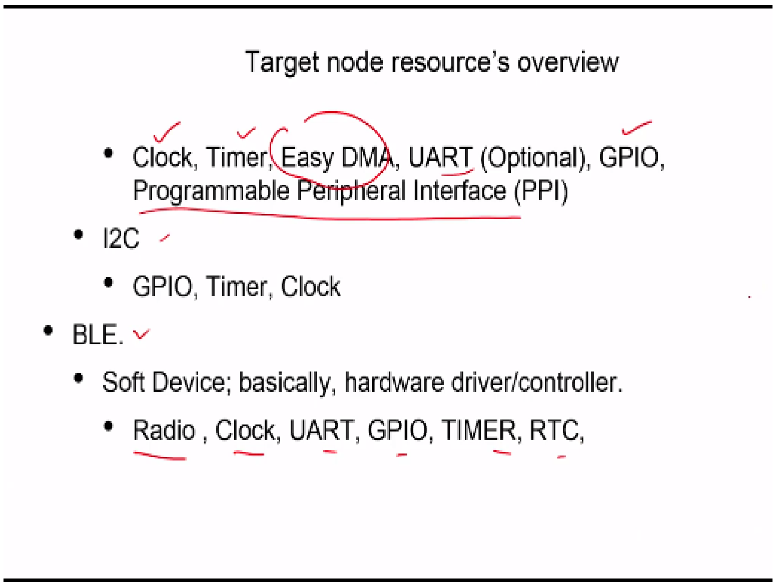
Now the next plot, this plot is even more interesting, because it is pushing it even more lower, right? If you look here, you were consuming around a little over 2 milliamperes and here you are consuming less than, 2.5 is here. And you will see that it is even more lower here. For example, this is even more lower and this is slightly higher, but then your 2 milliampere is here, right?

So this is 2.5. Your 2 milliampere maybe something like, something like here, right? So that reduction is there. And further reduction is possible. And how did we do that? We did because we put the microphone on super amount of duty cycle. Out of the three, we have disabled other two microphones there. And there are alerts which are coming, you can see these are alerts and we are communicating the alerts.

But after the alerts are gone, and you put back the comparator mode, you will be able to consume lower current. And you can see you are almost in the little over about a 1 milliampere or so of current consumption is now possible, little over 1 milliampere is what it is showing. But every time there are alerts, the communication unit kicks up, there is a certain amount of current consumption there.

And then again, it goes back to low power mode. So this is basically microphones case power consumption. And this is what you can do in terms of optimizing this hardware that we have in hand.

(Refer Slide Time: 26:03)



Target node resource's overview

- Clock, Timer, Easy DMA, UART (Optional), GPIO, Programmable Peripheral Interface (PPI)
- I2C
  - GPIO, Timer, Clock
- BLE.
  - Soft Device; basically, hardware driver/controller.
    - Radio , Clock, UART, GPIO, TIMER, RTC,

Now what are all the things that we did in order to come to these numbers? We played with the clock, we played with the timer. Of course, you have to use DMA. You have to use the UART, GPIO, and all the programmable peripheral interfaces, the I2C bus, the BLE, which is the communication module. And we did lot of optimizations on the radio clock, UART, GPIO, timer and RTC modules.

And thus, we were able to reduce it to a extremely low under 1 milliampere current consumption, which is a very encouraging thing. This will allow the node to live for a very long time and start sensing. After all the power optimizations, you can see that under 1 milliampere current consumption is actually achieved. And this is essentially the trick behind the whole process of reducing the current consumption in the node, okay.

So these are all more or less identical pictures. And you are back to other related things. Like we were using Bluetooth Low Energy mesh. If you come out of debug mode, I said, you will reduce by 6 to 7 milliamperes. A mesh enabled, if you disable the mesh, you will reduce by 1.2 milliamperes and so on. So lot of settings. If you read the datasheet, you would know several settings that you may have to play with.

COMP here indicates comparator, okay. Read about the comparator, read about the mesh parts and so on, which will allow you to sort of cycle it down to an extremely low current consumption of the system. You may be wondering folks, if you look at this picture, the y axis is at 14. And in this picture, the y axis has come down to 6. Why is it not 14? You may ask this question.

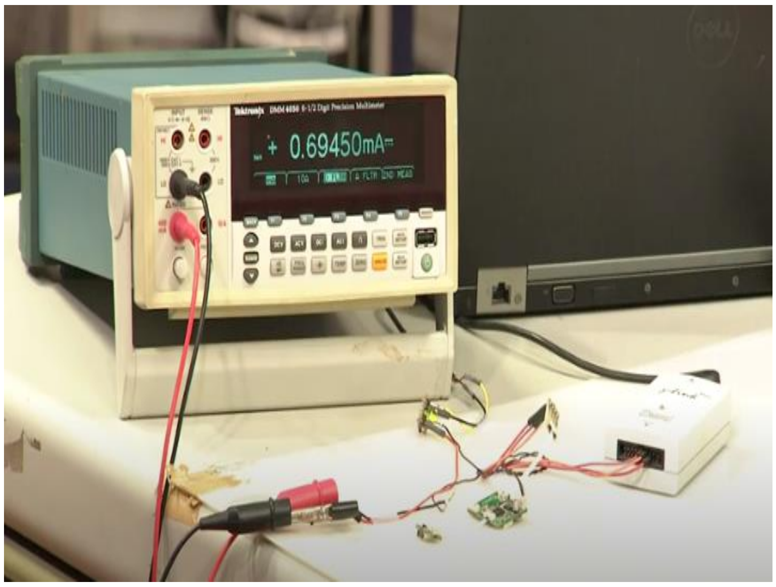
Well, I wanted to show you some contrasting things, right? That is not important, what is it that is here. But what is it that we have optimized for is important. This is important, you can see, but you may still be wondering why is it gone down to 6? It should have come, it should have fallen from 14 to this number. Instead it is only falling from 6 to this number, right. So that could be your question.

The answer to this riddle is we did this with USB connection to the mode okay, USB connection from laptop, okay. This one was done with coin cell battery okay, this is done with coin cell battery. So we shifted to USB. You can see that it took 6 milliamperes. Similarly, this 17.5 milliampere is another strange thing.

This is to keep the power constant, to keep the power constant from a coin cell battery, from a coin cell battery. Remember a coin cell battery is discharging continuously, okay? Whenever the voltage drops, the current has to increase so that the power is kept constant, correct? Therefore, the current consumption has gone up to whatever number you see here, which is 17.5 milliamperes.

So this is the reason why you see on the y axis different numbers. And therefore, remember that these kind of measurements may also happen to you and you have to take cognizance of the numbers you see on the y axis. Okay folks, let us put our attention to the demonstration.

(Refer Slide Time: 29:50)



We had promised 700 microamps, right? You are getting 700 microamps. At least that is what the current meter is showing for a significantly long time. Suddenly it goes to 2.3, if you look carefully, it goes to 2.3 milliamperes. Yeah, there you are. That is when the accelerometer is being sensed, okay. Sometimes you will also see 1 milliampere, a little over 1 milliampere.

And that is the time when ADC is enabled, and it is acquiring something in terms of data coming from the microphone. Sometimes you will also see 4 milliamperes. You will see suddenly spikes of 4 milliamperes. Essentially, it is this application folks, which we put together, which will show you different states of the current consumption.

And by and large if you deploy this ball for monitoring footsteps, with all the optimizations we have done is about 700 microamperes. What I have not done, what I have not done, and what I have not shown you here is can I reduce it even lower when there are footsteps, when there are footsteps? What should you do for that? Think about it, make your calculations and tell me what and how much more can we reduce from the 700 microampere that you see on the screen. Thank you very much.