In this next video we will show you how to initialize the ADC on the TM4C123 microcontroller.

VIDEO 14.3. ADC INITIALIZATION RITUAL

Help

0:00 / 12:24 1.0x

PROF.

RAMESH YERRABALLI: So in the last module,

we saw how stimuli in the external world can be captured by our computer

as long as it's rendered as a voltage.

So in this module, we'll look at how your micro-controller does this job.

That is, we'll look at the process of setting it up,

and then we'll look at the process of capturing the samples.

So on our micro-controller, which is our launchpad,

we have several of the GPIO pins that play dual roles.

So what we mean by the dual role is we'll choose their alternate function.

So there are 12 pins which are called 12 ADC pins which

are capable of receiving analog inputs, which are as rendered voltages.

and then converting them to 12-bit numbers using the ADC module on board.

So among these 12 ADC pins, two pins are the ones we

going to focus in this course.

One is PE2.

And the other is PD3.

This is referred to as Analog in 1.

And this is referred to as the Analog in 4.

By the way, we've already been using PD3 all along.

This is that analog pin that is used to measure voltages.

This was our way of measuring voltage on our circuit all along.

But AIN1, which is what we're going to use today,

is going to be used to measure voltages from an input device.

So let's get started on looking at all the device

registers that we have to manipulate to make ADC work.

So like any device that we've looked at, in order to manipulate the device,

we have to work with an initialization sequence.

The initialization sequence is done once,

or what we call as a set-up sequence.

And the setup sequence for ADC involves 13 steps.

I'm going to summarize the first few steps.

The first one through five steps are things that you already

area familiar with, which are make sure that the GPIO

pin-- here a GPIO pin for port E-- pin 2-- is properly configured.

So which means we turn on the clock.

We make it an input.

So we manipulated the direction register.

We enable its alternate function, which is the most important thing, which

is different from what we were doing so far.

So we enable its alternate function.

And then we (disable) it.

DEN, we (disable digital) .

And once we are done with it, we have another one,

which is an AM select function.

We do that.

So this is my five-step sequence, which is something

that you're already familiar with.

So what is different to make it an analog ADC pin

is the next sequence of steps.

So our first step that's specific to ADC is we turn ADC clock.

Like we turn on the clock here, we'll turn this one on.

We do that by making sure that RCGC0 bit 16 is set to 1.

So the next step we're going to do, which is step seven,

is we're going to configure the speed.

And the speed is the speed at which we're going to capture our samples.

There are multiple speeds ranging from 1 million samples per second all the way

to 125K samples per second.

And we effect that-- we make that change--

by making sure that we manipulate these two bits.

And for now, I'm going to make these two bits

0 0, which says that I've chosen the 125K.

So that's 125K samples per second, which is

the minimum speed at which I can sample.

The eighth step, then, is our sequencer.

We have four sequencers.

We don't have to worry about them.

We're using sequencer 3.

Sequencer 3 is chosen.

And we also set the priority of it.

So because there are four sequencers, the priority is set by a two-bit number

here.

And we're making it the highest priority by making it 0 among the four.

The step nine is while I'm configuring the rest of the system,

I'm going to disable.

And when I'm done at the end, which is my last step, which is step 13,

I will enable.

This way, I'm making sure that while I'm configuring it,

I can make sure that the sampling doesn't occur.

So I do that by manipulating this bit, which is my activation bit.

So I turn this off.

And I turn it on to affect that.

So it goes from 0 to 1 eventually.

So our next step is step 10, which is we choose the trigger.

That is, when we have analog input connected,

it can be triggered by various mechanisms.

But in this class, we're going to use a software-initiated trigger.

So it means that I will not expect any interrupt or anything.

I'm simply going to tell when I want to choose to sample.

And when the sampling is done, I will read it myself.

So I do that by manipulating these four bits here.

And these four bits for us is going to be the simplest case, which

is all 0s, which says that I'm using software as a trigger.

PROF.

JONATHAN VALVANO: Professor Yerraballi, how do we

tell it which channel to sample?

PROF.

RAMESH YERRABALLI: OK, so everything we did here doesn't really

tell us anything about what channel we're using.

So our next step will involve exactly that,

which is my step 11, which says that I want to choose channel 1.

Remember, there are 12 channels.

They go from 0 through 11.

And so I'm going to choose channel 1 out of those.

Help

So 1 is the I'm trying to choose.

So I need to specify that.

And I do that by writing a 0, 0 0 1 here.

And that's my channel select.

Channel is AIN1.

PROF.

JONATHAN VALVANO: Now is channel 1 always connected to PE2?

Or is that some kind of choice?

PROF.

RAMESH YERRABALLI: It is always connected to PE2.

And it's hard-wired.

And that's why we have selected the alternate function for PE2

to be its alternate function, which happens to be analog AIN1.

PROF.

JONATHAN VALVANO: Ah, OK.

One more step.

PROF.

RAMESH YERRABALLI: So we have one last step left, step 12,

which involves manipulating these four bits.

The only bit of real interest to us is this bit here, which is the IE0 bit.

And what it's telling us is that we want a flag to be set when the sampling is

complete-- set flag on sample capture.

The other bits, for now, are going to be 0 here, a 1 here, and a 0 here.

So that's going to be 0 1 1 0 into this, which

is a 6 is what we're going to write.

PROF.

JONATHAN VALVANO: So we do this once.

And so next, we're going to have to show you

how to write software that actually does the conversion-- starts, waits

for it to be done, and captures it.

PROF.

RAMESH YERRABALLI: That is correct.

So we looked at the initialization ritual.

Let's look at the sample capture procedure.

So to capture a sample, I'm going to follow this logic, which

says start with initialization after sampling by writing to this bit a 1.

So I make that bit a 1.

And then the ADC device starts its capturing.

And then I'm going to check for a flag.

So I'm going to check whether RIS bit 3 is a 0 to indicate that it is Busy.

So if it's busy, I keep checking back again and again.

And when it's 1, it tells me that the capture is complete.

So once the capture is complete, I'm going to read the data.

So I will read data, which means that I've done this busy step, which

I found this to be 0 for a good bit of time and eventually it became a 1.

So I come down here and then I'm going to read this data.

So read data.

So the last thing I need to do is to make sure that I clear the flag.

I clear this flag by writing a 1 here.

So the act of writing a 1 there will make sure

that this flag goes back to a 0 so I'm ready for the next sample.

And then I return.

PROF.

JONATHAN VALVANO: I have a question.

Do these registers act like memory?

In other words, if I write a 1 to them, it will become a 1?

And if I write a 0, it will become a 0?

PROF.

RAMESH YERRABALLI: Well, these are really not memory registers.

These are device registers.

PROF.

JONATHAN VALVANO: Oh.

PROF.

RAMESH YERRABALLI: So they don't behave like memory.

They behave more like inputs and outputs to a device.

Help

PROF.

JONATHAN VALVANO: So if I write to one register,

it clears a bit in another register?

PROF.

RAMESH YERRABALLI: Yeah.

Because it's a memory-mapped I/O.

PROF.

JONATHAN VALVANO: I see.

Cool.

All right.

Enough talking.

Let's build it.

PROF.

RAMESH YERRABALLI: All right.

Help

We perform the following steps to configure the ADC for software start on one channel. Program 14.1 shows specific details for sampling PE2, which is channel 1. The function **ADC0_InSeq3** will sample PE2 using software start and use busy-wait synchronization to wait for completion.

Step 1. We enable the port clock for the pin that we will be using for the ADC input.

Step 2. Make that pin an input by writing zero to the **DIR** register.

Step 3. Enable the alternative function on that pin by writing one to the **AFSEL** register.

Step 4. Disable the digital function on that pin by writing zero to the **DEN** register.

Step 5. Enable the analog function on that pin by writing one to the **AMSEL** register.

Step 6. We enable the ADC clock by setting bit 16 of the **SYSCTL_RCGC0_R** register.

Step 7. Bits 8 and 9 of the **SYSCTL_RCGC0_R** register specify the maximum sampling rate of the ADC. In this example, we will sample slower than 125 kHz, so the maximum sampling rate is set at 125 kHz. This will require less power and produce a longer sampling time, creating a more accurate conversion.

Step 8. We will set the priority of each of the four sequencers. In this case, we are using just one sequencer, so the priorities are irrelevant, except for the fact that no two sequencers should have the same priority.

Step 9. Before configuring the sequencer, we need to disable it. To disable sequencer 3, we write a 0 to bit 3 (**ASEN3**) in the **ADC_ACTSS_R** register. Disabling the sequencer during programming prevents erroneous execution if a trigger event were to occur during the configuration process.

Step 10. We configure the trigger event for the sample sequencer in the **ADC_EMUX_R** register. For this example, we write a 0000 to bits 15–12 (**EM3**) specifying software start mode for sequencer 3.

Step 11. Configure the corresponding input source in the **ADCSSMUXn** register. In this example, we write the channel number to bits 3–0 in the **ADC_SSMUX3_R** register. In this example, we sample channel 1, which is PE2.

Step 12. Configure the sample control bits in the corresponding nibble in the **ADC0SSCTLn** register. When programming the last nibble, ensure that the **END** bit is set. Failure to set the **END** bit causes unpredictable behavior. Sequencer 3 has only one sample, so we write a 0110 to the **ADC_SSCTL3_R** register. Bit 3 is the **TS0** bit, which we clear because we are not measuring temperature. Bit 2 is the **IE0** bit, which we set because we want to the **RIS** bit to be set when the sample is complete. Bit 1 is the **END0** bit, which is set because this is the last (and only) sample in the sequence. Bit 0 is the **D0** bit, which we clear because we do not wish to use differential mode.

Step 13. We enable the sample sequencer logic by writing a 1 to the corresponding **ASENn**. To enable sequencer 3, we write a 1 to bit 3 (**ASEN3**) in the **ADC_ACTSS_R** register.

```
void ADC0_InitSWTriggerSeq3_Ch9(void){ volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x00000010;   // 1) activate clock for Port E
  delay = SYSCTL_RCGC2_R;         //    allow time for clock to stabilize
  GPIO_PORTE_DIR_R &= ~0x04;      // 2) make PE2 input
  GPIO_PORTE_AFSEL_R |= 0x04;     // 3) enable alternate function on PE2
  GPIO_PORTE_DEN_R &= ~0x04;      // 4) disable digital I/O on PE2
  GPIO_PORTE_AMSEL_R |= 0x04;     // 5) enable analog function on PE2
  SYSCTL_RCGC0_R |= 0x00010000;   // 6) activate ADC0
  delay = SYSCTL_RCGC2_R;
  SYSCTL_RCGC0_R &= ~0x00000300;  // 7) configure for 125K
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority
  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 1;             //    set channel Ain1 (PE2)
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_ACTSS_R |= 0x0008;         // 13) enable sample sequencer 3
}
```

Program 14.1. Initialization of the ADC using software start and busy-wait (C14_ADCSWTrigger).

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring 14.2. ADC on the T... to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.

(http://www.facebook.com/EdxOnline)

(https://twitter.com/edXOnline)

(https://plus.google.com/108235383044095082735/posts)

(http://youtube.com/user/edxonline)

Help