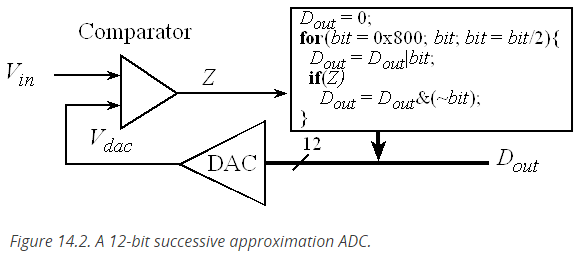
**C14 ADC and Data Acquisition**

**C14.1 Approximating Continuous Signals in the Digital Domain**

The ADC **precision** is the number of distinguishable ADC inputs (ex. 4096 alternatives, 12 bits). The ADC **range** is the max and min ADC input (ex. 0 to +3.3V). The ADC **resolution** is the smallest distinguishable change in input (ex. 3.3V/4096 which is about 0.81 mV).

One of the most common methods for ADC conversion is the successive approximation technique. For each clock, the successive approximation hardware issues a new “guess” on the by setting the bit under test to a “1”. If is less then , then the bit under test remains a 1. is the ADC digital output, and X is the binary input that is true if is greater than .



**C14.2 ADC on the TM4C123/LM4F120**

Bits 8 and 9 of the **SYSCTL\_RCGC0\_R** specify the maximum sampling rate. The actual sampling rate is determined by the rate at which we trigger the ADC. On the TM4C123, we will need to set bits in the **AMSEL** register to activate the analog interface.

Table 14.2 shows the configuration to set the maximum sampling speed. The actual sampling rate is determined by how many times per second the software starts the ADC.

Table 14.2. The ADC MAXADCSPD bits in the SYSCTL\_RCGC0\_R register specify the maximum sampling speed.

|  |  |
| --- | --- |
| *Value* | *Maximum sampling frequency* |
| 0x3 | 1M samples/second |
| 0x2 | 500K samples/second |
| 0x1 | 250K samples/second |
| 0x0 | 125K samples/second |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| IO | Ain | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 14 |
| PB4 | Ain10 | Port |  | SSI2Clk |  | M0PWM2 |  |  | T1CCP0 | CAN0Rx |  |  |
| PB5 | Ain11 | Port |  | SSI2Fss |  | M0PWM3 |  |  | T1CCP1 | CAN0Tx |  |  |
| PD0 | Ain7 | Port | SSI3Clk | SSI1Clk | I2C3SCL | M0PWM6 | M1PWM0 |  | WT2CCP0 |  |  |  |
| PD1 | Ain6 | Port | SSI3Fss | SSI1Fss | I2C3SDA | M0PWM7 | M1PWM1 |  | WT2CCP1 |  |  |  |
| PD2 | Ain5 | Port | SSI3Rx | SSI1Rx |  | M0Fault0 |  |  | WT3CCP0 | USB0epen |  |  |
| PD3 | Ain4 | Port | SSI3Tx | SSI1Tx |  |  |  | IDX0 | WT3CCP1 | USB0pflt |  |  |
| PE0 | Ain3 | Port | U7Rx |  |  |  |  |  |  |  |  |  |
| PE1 | Ain2 | Port | U7Tx |  |  |  |  |  |  |  |  |  |
| PE2 | Ain1 | Port |  |  |  |  |  |  |  |  |  |  |
| PE3 | Ain0 | Port |  |  |  |  |  |  |  |  |  |  |
| PE4 | Ain9 | Port | U5Rx |  | I2C2SCL | M0PWM4 | M1PWM2 |  |  | CAN0Rx |  |  |
| PE5 | Ain8 | Port | U5Tx |  | I2C2SDA | M0PWM5 | M1PWM3 |  |  | CAN0Tx |  |  |

Table 14.3 shows which I/O pins on the TM4C123 can be used for ADC analog input channels. You will use PE2 for Labs 14 and 15, and TExaS uses PD3 for the voltmeter and oscilloscope.

Table 14.3. Twelve different pins on the LM4F/TM4C can be used to sample analog inputs.  You will use ADC0 and PE2 to sample analog input. If your PE2 pin is broken, you will have the option to perform Lab 14 with PE3 or PE5. We use ADC1 and PD3 to implement the oscilloscope feature.

The ADC has four sequencers. To use sequencer 3, we set the **ADC0\_SSPRI\_R** register to 0x0123 to make the sequencer 3 the highest priority. Because we use just one sequencer, we just need to make sure each sequencer has a unique priority. We set bits 15-12 (**EM3**) in the **ADC0\_EMUX\_R** register to specify how the ADC will be triggered. For this course, we use a software start to trigger the ADC (**EM3=0x0**). The software writes an 8 (**SS3**) to the **ADC0\_PSSI\_R** to initiate conversion on sequencer 3. We can enable and disable the sequencers using the **ADC0\_ACTSS\_R** register. There are twelve ADC channels on the LM4F120/TM4C123. Which channel we sample is configured by writing to the **ADC0\_SSMUX3\_R** register. The mapping between channel number and the port pin is shown in Table 14.3. For example, channel 9 is connected to pin PE4. The **ADC0\_SSCTL3\_R** register specifies the mode of the ADC sample. We set **TS0** to measure temperature and clear it to measure the analog voltage on the ADC input pin. We set **IE0** so that the **INR3** bit is set when the ADC conversion is complete, and clear it when no flags are needed. When using sequencer 3, there is only one sample, so **END0** will always be set, signifying this sample is the end of the sequence. In this class, the sequence will be just one ADC conversion. We set the **D0** bit to activate differential sampling, such as measuring the analog difference between two ADC pins. In our example, we clear **D0** to sample a single-ended analog input. Because we set the **IE0** bit, the **INR3** flag in the **ADC0\_RIS\_R** register will be set when the ADC conversion is complete. We clear the **INR3** bit by writing an 8 to the 8 to the **ADC0\_ISC\_R** register.

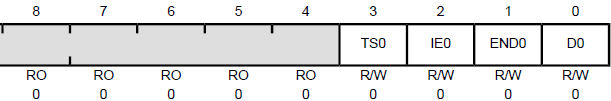
|  |  |
| --- | --- |
| *Value* | *Event* |
| 0x0 | Software start |
| 0x1 | Analog Comparator 0 |
| 0x2 | Analog Comparator 1 |
| 0x3 | Analog Comparator 2 |
| 0x4 | External (GPIO PB4) |
| 0x5 | Timer |
| 0x6 | PWM0 |
| 0x7 | PWM1 |
| 0x8 | PWM2 |
| 0x9 | PWM3 |
| 0xF | Always (continuously sample) |

*Table 14.4. The ADC EM3, EM2, EM1, and EM0 bits in the ADC\_EMUX\_R register.*

**Configuration Steps**

We perform the following steps to configure the ADC.

1. We enable the port clock for the pin that we will be using for the ADC input.
2. Make that pin an input by writing zero to the **DIR** register.
3. Enable the alternative function on that pin by writing one to the **AFSEL** register.
4. Disable the digital function on that pin by writing zero to the **DEN** register.
5. Enable the analog function on that pin by writing one to the **AMSEL** register.
6. We enable the ADC clock by setting bit 16 on the **SYSCTL\_RCGC0\_R** register.
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 to 125 kHz. This will require less power and produce a longer sampling time, creating a more accurate conversion. The range is 125 kHz to 1 MHz. Thus, set this to 00.
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. To set sequencer 3 to the highest priority, set SSPRI bits 12-13 to 00. Just give the other ones any priority 1 2 or 3.
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.
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.
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.
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 tot 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.



1. We enable the sample sequencer logic by writing a 1 to the corresponding **ASENn**. To enable sequencer 3, we write a1 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).

There are four steps required to perform a software-start conversion. The range is 0 to 3.3V. If the analog input is 0, the digital output will be 0, and if the analog input is 3.3V, the digital output will be 4095:

**Step 1.** The ADC is started using the software trigger The channel to sample was specified earlier in the initialization.

**Step 2.** The function waits for the ADC to complete by polling the RIS register bit 3.

**Step 3.** The 12-bit digital sample is read out of sequencer 3.

**Step 4.** The RIS bit is cleared by writing to the ISC register.

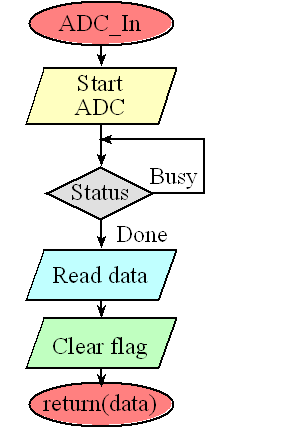


Figure 14.3. The four steps of analog to digital conversion: 1) initiate conversion, 2) wait for the ADC to finish, 3) read the digital result, and 4) clear the completion flag.

//------------ADC\_InSeq3------------  
// Busy-wait analog to digital conversion  
// Input: none  
// Output: 12-bit result of ADC conversion  
unsigned long ADC0\_InSeq3(void){  unsigned long result;  
  ADC0\_PSSI\_R = 0x0008;            // 1) initiate SS3  
  while((ADC0\_RIS\_R&0x08)==0){};   // 2) wait for conversion done  
  result = ADC0\_SSFIFO3\_R&0xFFF;   // 3) read result  
  ADC0\_ISC\_R = 0x0008;             // 4) acknowledge completion  
  return result;  
}  
  
Program 14.2. ADC sampling using software start and busy-wait (C14\_ADCSWTrigger).

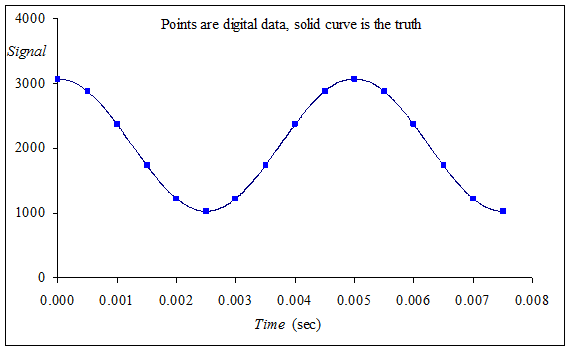
**C14.3 Nyquist Theorem**

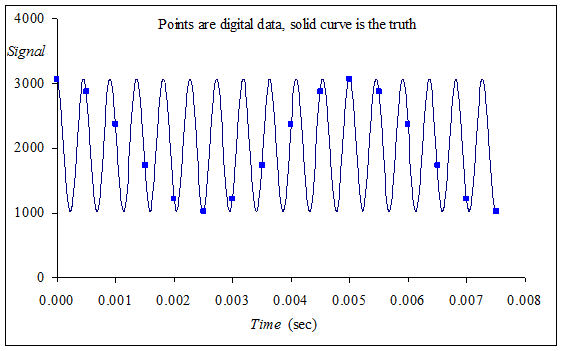
If we use Systick periodic interrupts, then the is the time between SysTick interrupts. We define the sampling rate as:

If this information oscillates at a frequency , then according to the Nyquist Theorem, we must sample that signal at

Furthermore, the **Nyquist Theorem** states that fi the signal is sampled with a frequency of , then the digital samples only contain frequency components from 0 to . Conversely, if the analog signal does contain frequency components larger than then there will be an aliasing error during the sample process (performed with a frequency of ). **Aliasing** is when the digital signal appears to have a different frequency than the analog signal.

In the plots below, the first signal (200 Hz) is sampled at 2000 Hz and we get a perfect sample. In the second signal (2200 Hz), it is sampled at 2000 Hz and thus we get aliasing.





**C14.4 Data Acquisition and Control Systems**

A **nonmonotonic** transducer is an input/output function that does not have a mathematical inverse. For example, the Sharp GP2Y0A21YK IR distance sensor has a transfer function shown below, if you read a voltage of 2 V, you cannot tell if the object is 3 cm away or 12 cm away. However, if the distance is always greater than 10 cm, then this transducer can be used.

