Lecture 14: Analog to Digital Conversion

ME/AE 6705

Introduction to Mechatronics

Dr. Jonathan Rogers

Lesson Objectives

- Understand the theory behind analog to digital conversion
  - ADC resolution, sampling rate above Nyquist frequency, Sample and Hold Circuits
- Understand operation of Successive Approximation ADC
- Be able to specify resolution and sampling rate for a given application
- Be able to configure ADC on MSPM0

Digital Inputs

- To date we have only been able to read digital inputs into the microcontroller
  - High (3.3 V, or actually > 1.4 V)
  - Low (0 V, or actually < 1.4 V)

For MSPM0, which uses TTL logic levels (runs at 3.3 V)

or

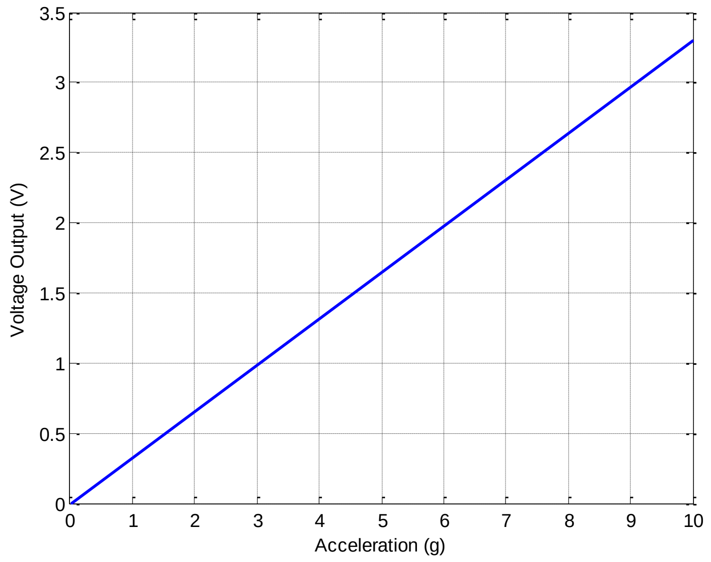
- High (5 V, or actually > 2.4 V)
- Low (0 V, or actually < 2.4 V)

For other MCU's, which use CMOS logic levels (run at 5 V)

Analog Signals

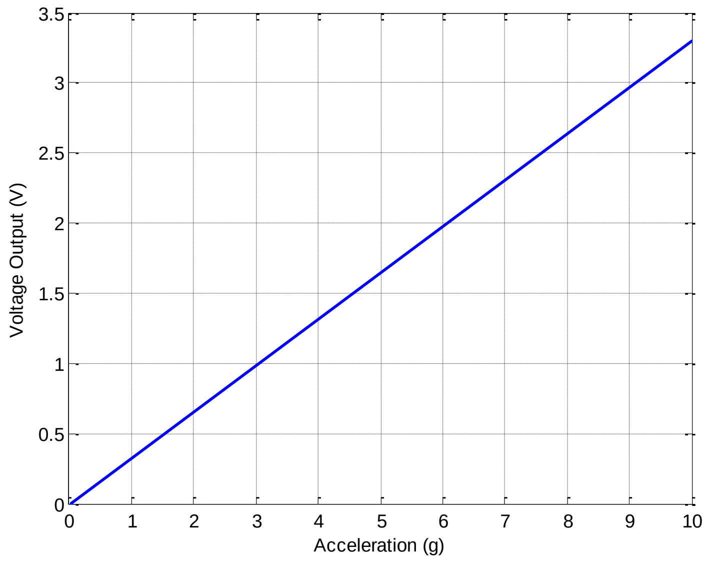
- Suppose you want to use an accelerometer to provide a feedback signal to your control law
  - Suppose accelerometer provides output voltage corresponding to acceleration measurement
  - Continuous between 0 V (for 0 g) and 3.3 V (for 10 g)

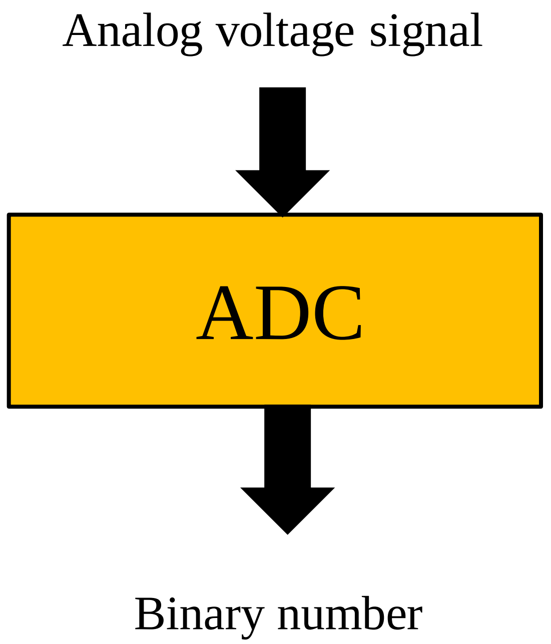
- With digital logic, we will just be able to tell whether the sensor reads above 4.25 g or below
- How do we get higher resolution measurements to our MCU?



Analog to Digital Converter (ADC)

- ADC’s are hardware device that measures a voltage with a given resolution and assigns a digital value to it

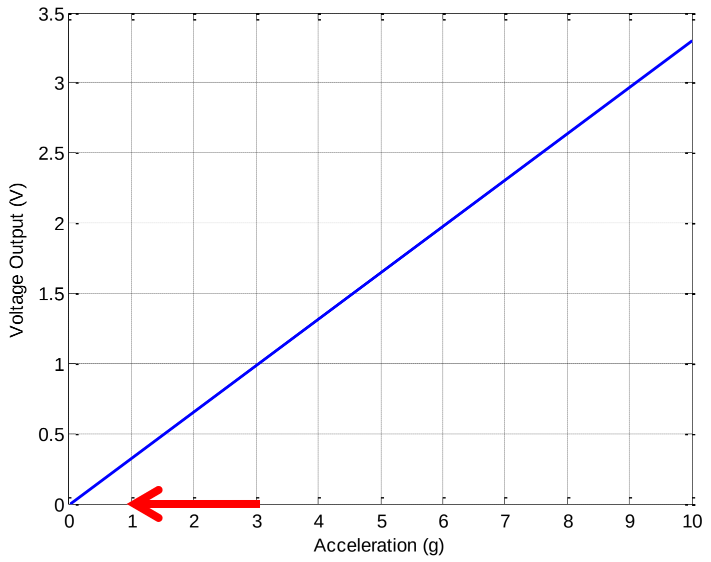


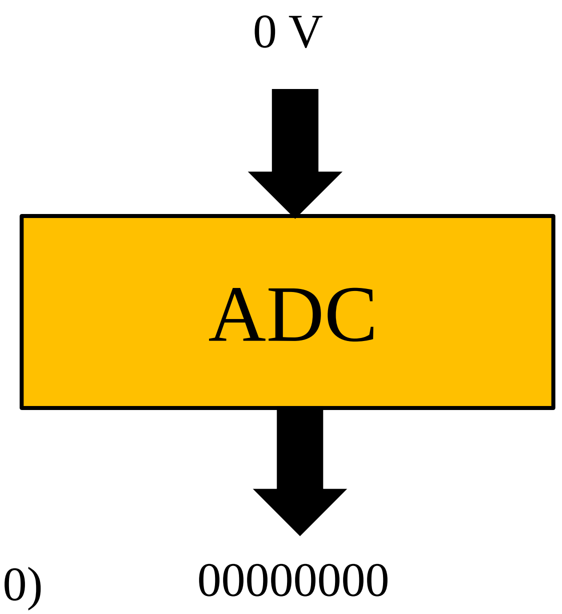


Analog to Digital Converter (ADC)

- ADC's are hardware device that measures a voltage with a given resolution and assigns a digital value to it

Consider ADC with 8 bit resolution and range of 0-3.3 V:



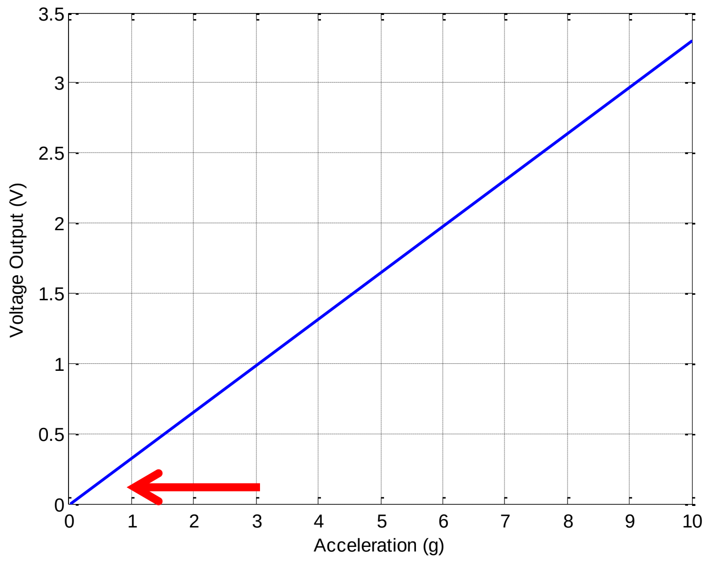


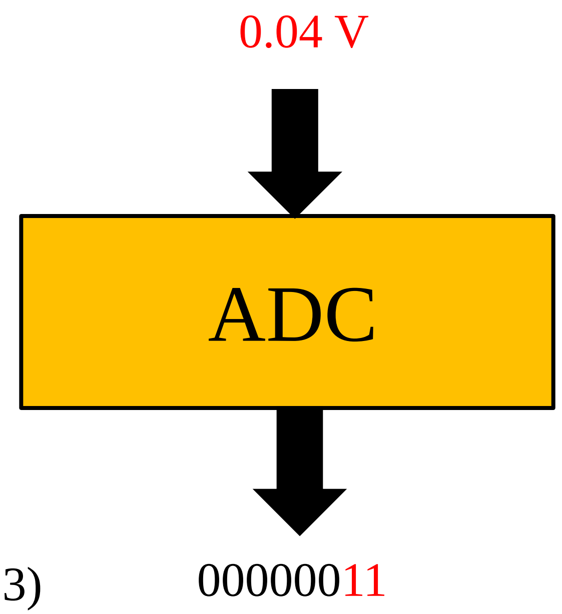
(decimal 0)

Analog to Digital Converter (ADC)

- ADC's are hardware device that measures a voltage with a given resolution and assigns a digital value to it

Consider ADC with 8 bit resolution and range of 0-3.3 V:



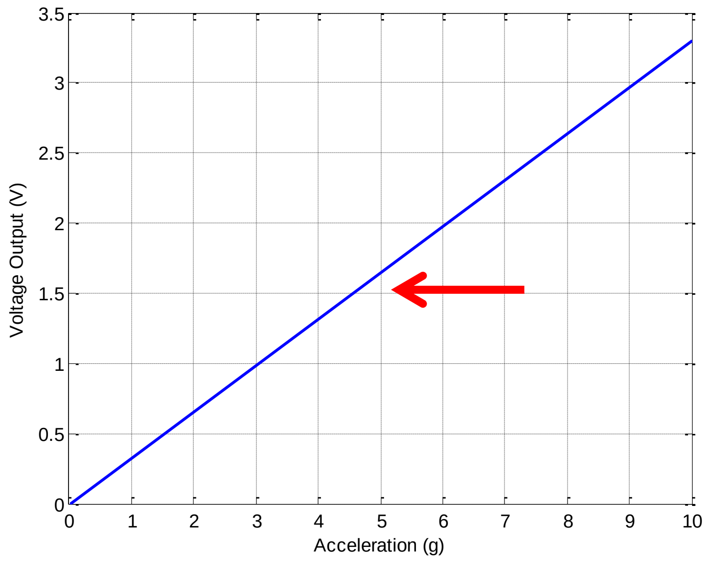


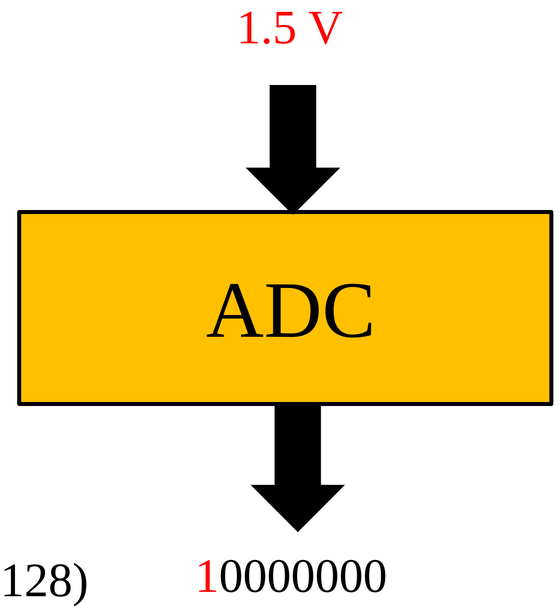
(decimal 3)

Analog to Digital Converter (ADC)

- ADC's are hardware device that measures a voltage with a given resolution and assigns a digital value to it

Consider ADC with 8 bit resolution and range of 0-3.3 V:

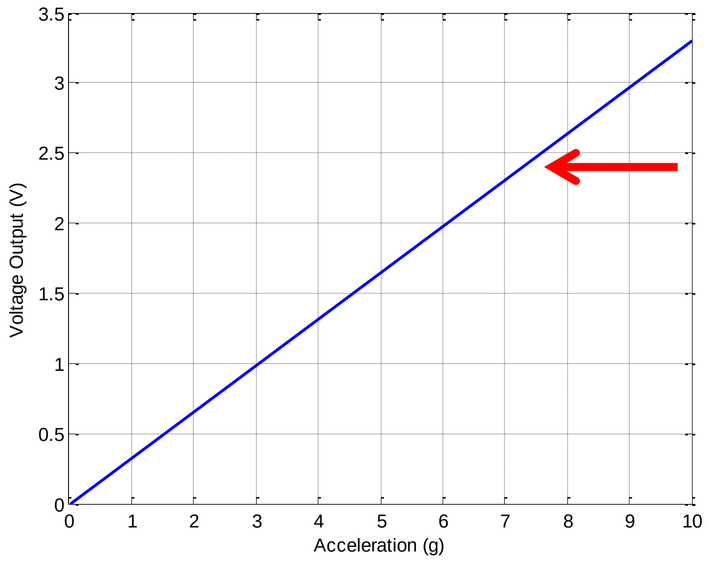




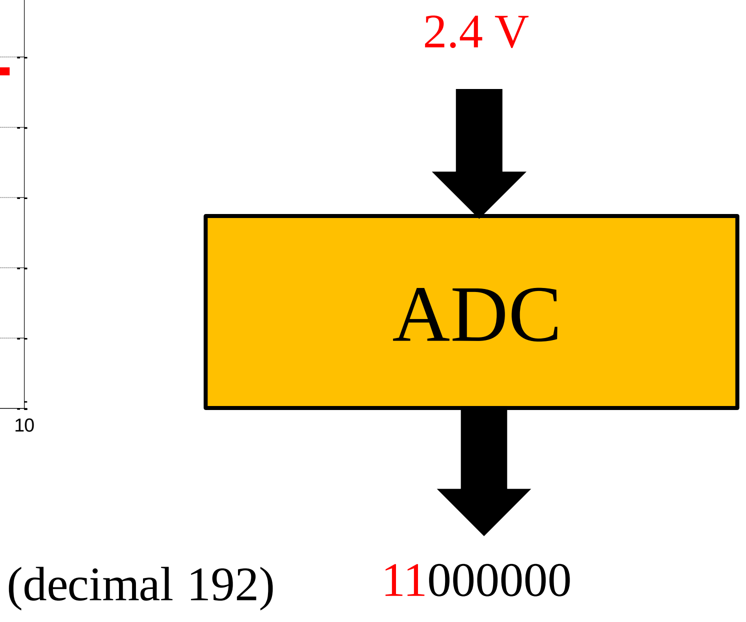
(decimal 128)

Analog to Digital Converter (ADC)

- ADC's are hardware device that measures a voltage with a given resolution and assigns a digital value to it



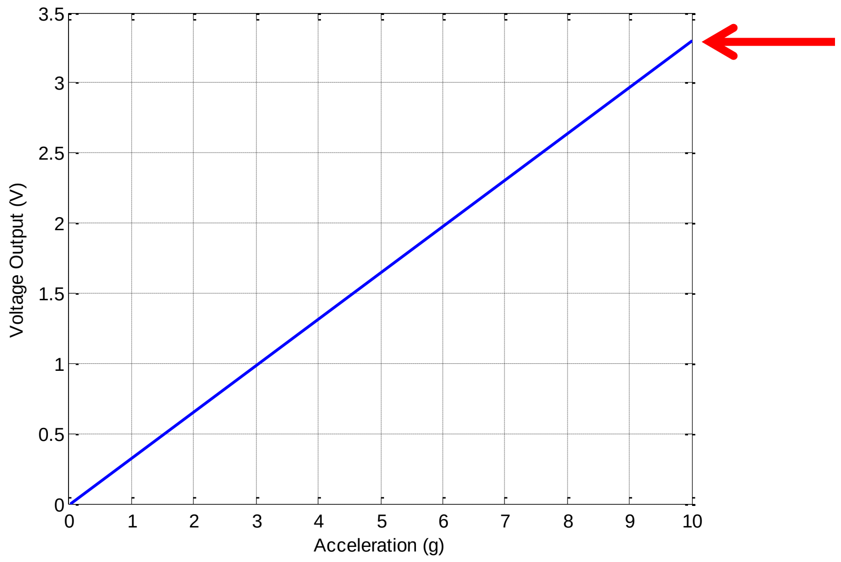
Consider ADC with 8 bit resolution and range of 0-3.3 V:

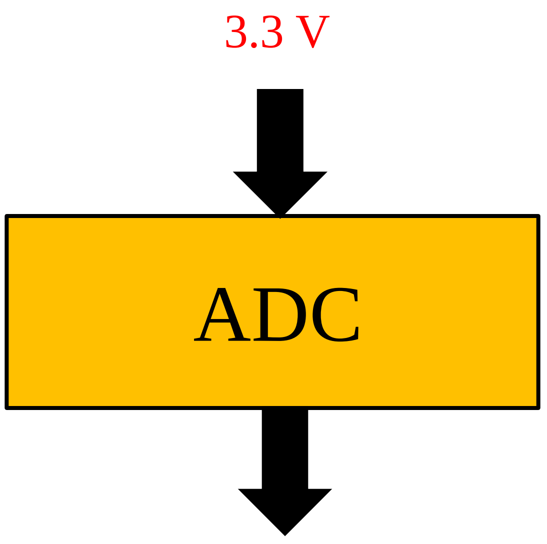


Analog to Digital Converter (ADC)

- ADC's are hardware device that measures a voltage with a given resolution and assigns a digital value to it

Consider ADC with 8 bit resolution and range of 0-3.3 V:





(decimal 255) 11111111

Analog to Digital Conversion

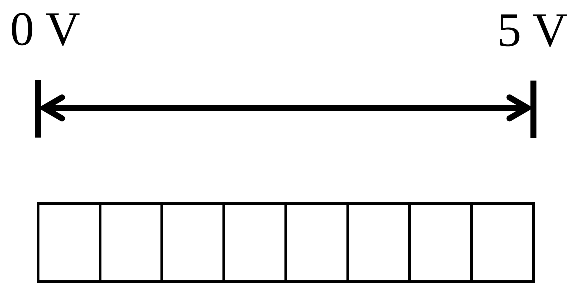
- Analog to Digital Converters work by dividing a range of voltage into equal divisions
- Terminology:
  - **Range**: Voltage range an ADC can measure (e.g., 0 - 2.5 V)
  - **Resolution**: Smallest distinguishable change in input (e.g., 0.15 mV)
  - **Precision**: Number of values ADC output can take on (e.g., 256 for 8-bit ADC, 16384 for 14-bit ADC)

Range (volts) = Precision (alternatives) x Resolution (volts)

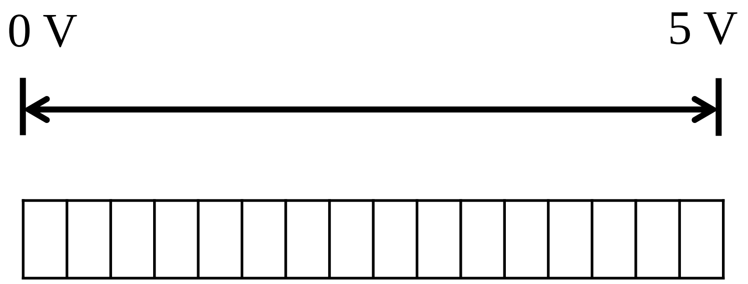
Resolution

- Resolution of ADC is determined both by range and precision
  - If we use more bits to represent given voltage range, we have higher resolution
  - If we use larger voltage range and keep number of bits constant, we have smaller resolution

3 bit ADC – yields 8 possible values



Each division represents increments of 5/8 = 625 mV



4 bit ADC – yields 16 possible values

Each division represents increments of 5/16 = 312 mV

Resolution

- Resolution is determined by number of bits *N* into which ADC divides voltage range
  - Number of divisions is 2^N-1
- Formula for computing resolution of ADC:

$$\text{Resolution} = \frac{V_{\max} - V_{\min}}{2^N}$$

Has units of V

  - Example: What is resolution of 8-bit ADC that has range of 0-3V?

$$\frac{V_{\max} - V_{\min}}{2^8} = \frac{3-0}{256} = 0.0117\ \text{V}$$

Resolution

- As voltage increases, output of ADC will truncate to nearest discrete division

Recall our resolution of 11.7 mV

Voltage In	Decimal Output	Binary Output
0 V	0	00000000
0.009 V	0	00000000
0.010 V	0	00000000
0.020 V	1	00000001
0.040 V	3	00000011
1.500 V	128	10000000
2.988 V	254	11111110
2.989 V	255	11111111

Resolution Rule of Thumb

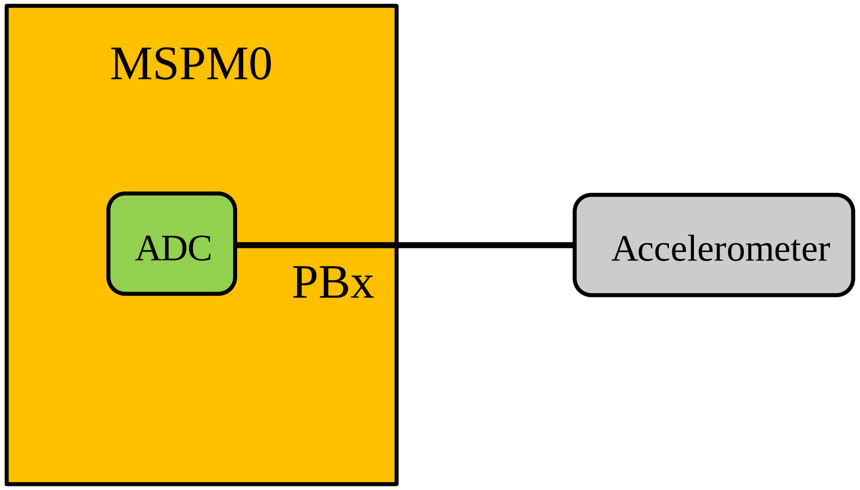
- For a given voltage range, every extra bit you use (approximately) doubles the resolution
- For 0-3V range,

8 bit	$\longrightarrow$	~ 12 mv
9 bit	$\longrightarrow$	~ 6 mv
10 bit	$\longrightarrow$	~ 3 mv
11 bit	$\longrightarrow$	~ 1.5 mV
16 bit	$\longrightarrow$	~ 46 µV

Note: In practice, ADCs are only made with even number of bits.

Selecting Resolution – Example

- Consider a +/- 4 g accelerometer. At 0 g, the output is 1.5 V, and sensitivity is 300 mV/g. We would like to be able to sense acceleration changes within 3 mg. How many bits are required from the ADC?



Selecting Resolution – Example

- Solution:
  - Voltage range: $1.5\text{V} \pm 4\text{g}\left(\dfrac{0.300\text{V}}{\text{g}}\right) = 0.3\text{V to } 2.7\text{V}$
  - Required resolution: $\dfrac{300\text{mV}}{\text{g}}\left(\dfrac{\text{g}}{1000\text{mg}}\right)(3\text{mg}) = 0.9\text{mV}$

Selecting Resolution – Example

- Now using our resolution formula:

$$\text{Resolution} = \frac{V_{\max} - V_{\min}}{2^N}$$

N = number of bits

$$0.9\text{ mV} = \frac{2.4\text{ V}}{2^N} \longrightarrow 2^N = \frac{2.4\text{ V}}{0.9\text{ mV}}$$

– Taking ln of both sides,

$$N = \frac{\ln\left(\frac{2.4\text{ V}}{0.9\text{ mV}}\right)}{\ln(2)} = 11.38$$

Round up to nearest integer

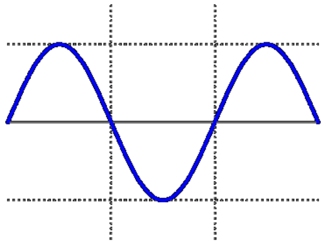
$$\longrightarrow N = 12$$

Thus we need a 12 bit ADC.

ADC Sampling Rate

- Suppose you want to record a time varying analog signal on the MCU (this is called *data acquisition*)
  - Let the analog signal be a sine wave given by:

$$s = A + B\sin\left(2\pi ft + \phi\right)$$

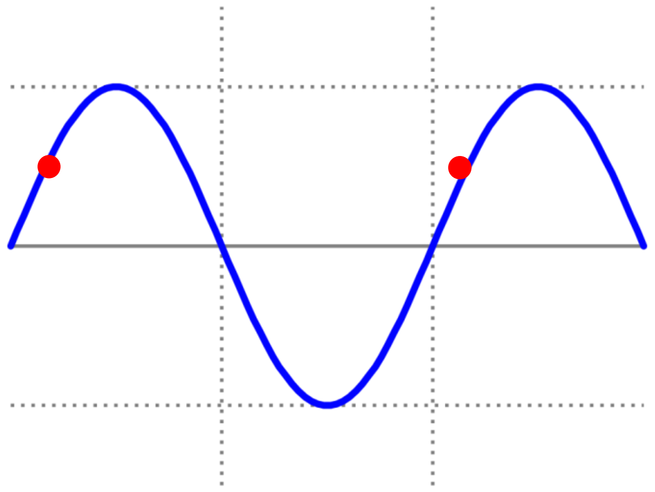


where f is frequency of signal s

- You want to sample this signal using ADC at certain frequency f_s and then determine parameters A, B, f, and ϕ from digital samples.
  - What is minimum sampling frequency f_s you can use?

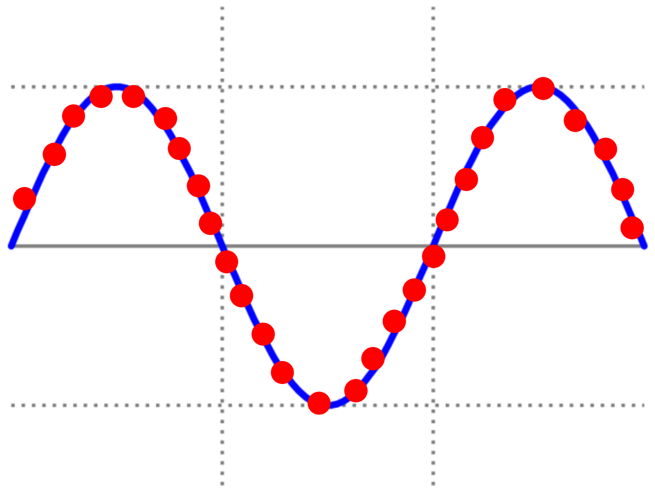
ADC Sampling Rate

Undersampling



- We undersample signal if $f_s < 2f$
- Our digitized signal does not look like a sine wave because we are not sampling fast enough
- This is called "aliasing"

Oversampling

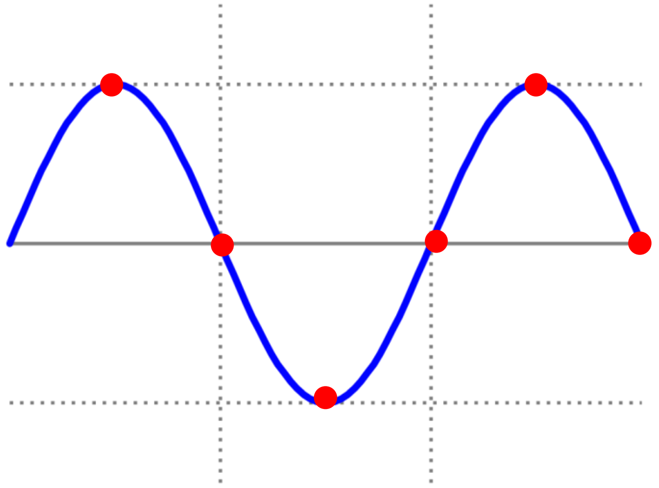


- We oversample signal if $f_s >> 2f$
- While we capture the sinusoidal profile of signal, we are using a lot more samples than needed
- Not very memory efficient

Nyquist Sampling Theorem

- If f_{max} is largest frequency component of an analog signal, you must sample at frequency greater than $2f_{max}$ to faithfully represent signal in digital samples.
  - $2f_{max}$ is called the "Nyquist frequency"
  - If we sample above the Nyquist frequency, we will be able to estimate parameters A, B, f, and ϕ

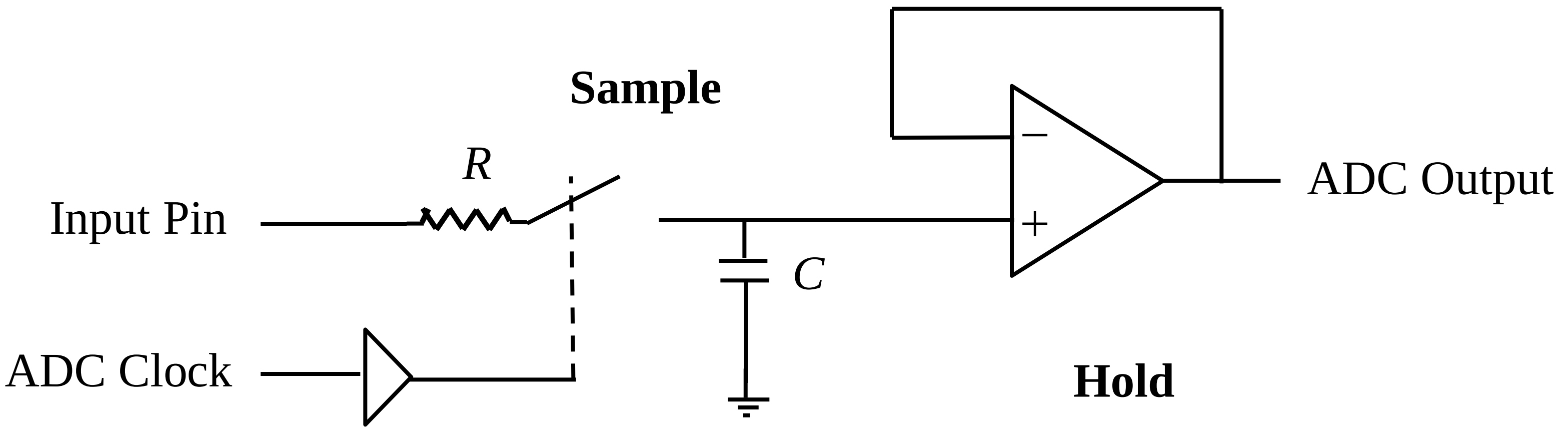
Signal sampled at $4f_{max}$



General guidance: Choose sampling rate well above Nyquist frequency, but not so large that you need huge amount of memory to store data.

Sample and Hold

- ADC operates by using a Sample and Hold (S/H) Circuit

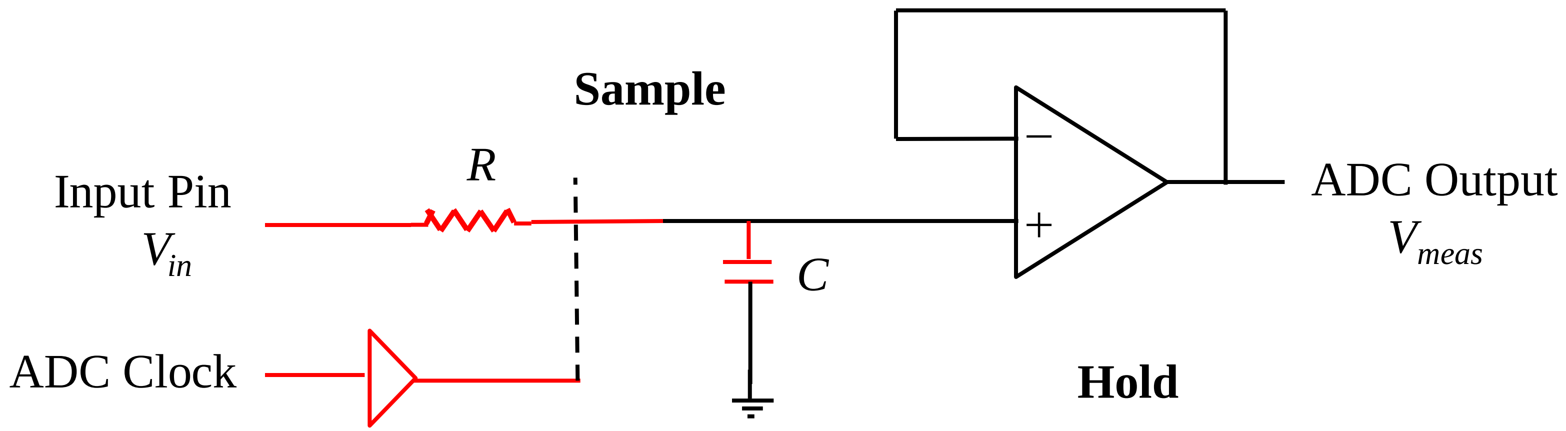


Sample component of circuit is switch that closes momentarily at each ADC clock signal (or each time ADC is read).

Hold component of circuit is "unity gain buffer amplifier" – holds capacitor voltage constant while ADC takes measurement

Sample and Hold

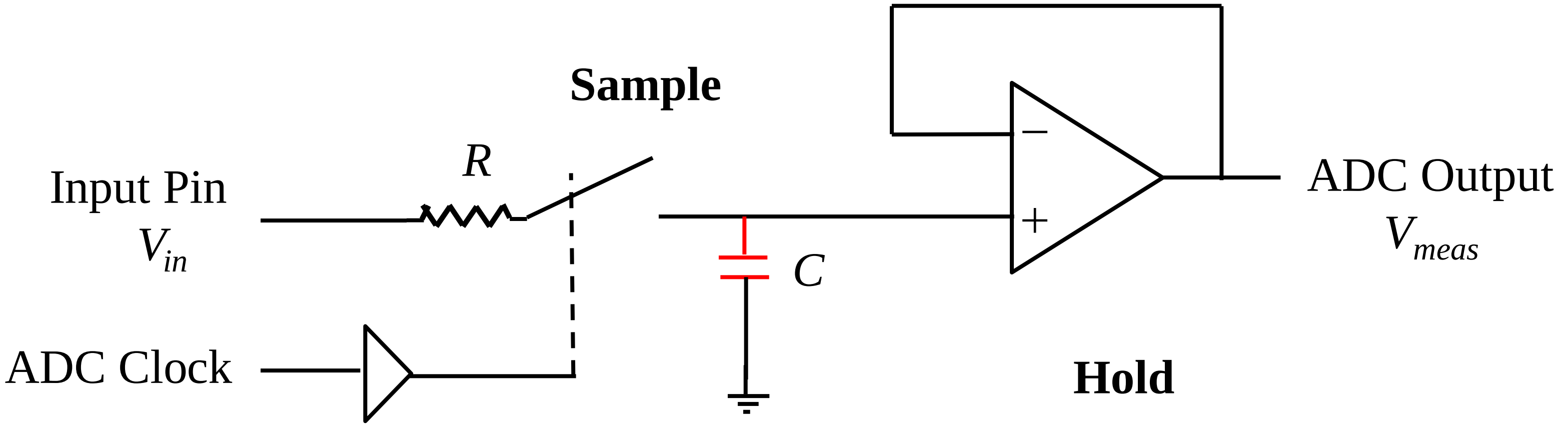
- Sample and Hold operation



Step 1: Switch closes when ADC is read by software to sample input line. Capacitor charges to voltage V_{in}.

Sample and Hold

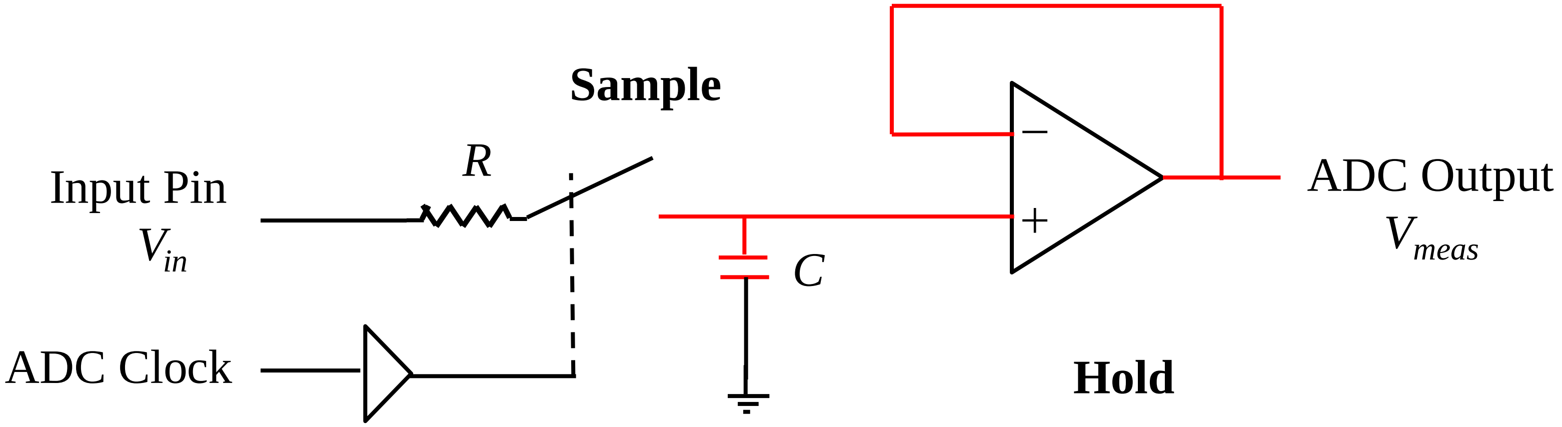
- Sample and Hold operation



Step 2: Switch opens, and hold circuit maintains capacitor at voltage V_{in}.

Sample and Hold

- Sample and Hold operation

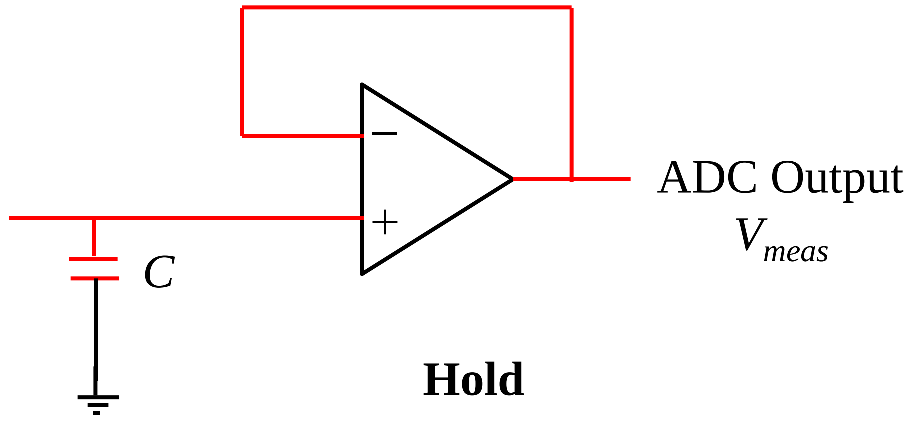


Step 3: ADC compares voltage (using successive approximation on next slides). Because buffer amplifier is used, current will not leak from capacitor while reading it so it maintains voltage.

Sample and Hold

- Importance of buffer amplifier
  - Suppose we did not use buffer amplifier and just hooked ADC output up to capacitor
    - *Current will leak out of capacitor while we read it, causing voltage droop*
  - Buffer amplifier acts as high-impedance element – allows us to read capacitor without drawing current

Essentially turns ADC into a voltmeter.



Successive Approximation ADC

- Most ADC's on MCU's are <u>successive approximation converters</u>
  - Sometimes called SAR ADC's
  - Uses trial and error approach to find voltage
  - Works by generating a voltage and comparing it to input using op-amp comparator
  - Repeats comparison for each bit from MSB to LSB
- Other types of ADC's include Flash, Counting, and Sigma-Delta

Successive Approximation ADC

- SAR ADC process:
  1. Start with k = number of bits N
  2. Set k^{th} bit to form $(N - k+1)^{\text{th}}$ approximation
  3. Generate a voltage $V_{D/A}$ equal to the $(N - k+1)^{\text{th}}$ approximation
  4. If $V_{D/A} > V_{in}$, decrease approximation by setting k^{th} bit to zero
  5. Set $k = k - 1$, go to step 2 and repeat until $k = 0$

Successive Approximation Example

- Starting with an initial guess of 0.0V, show the steps an 8-bit SAR ADC will perform to represent 2.03V (assuming voltage range of 0-3V).

Successive Approximation Example

- Starting with an initial guess of 0.0V, show the steps an 8-bit SAR ADC will perform to represent 2.03V (assuming voltage range of 0-3V).

***k* = 8** **1st Approximation**

$$\mathbf{10000000} \longrightarrow V_{D/A} = (128/255)*3V = 1.5V < 2.03V \longrightarrow \mathbf{10000000}$$

***k* = 7** **2nd Approximation**

$$\mathbf{11000000} \longrightarrow V_{D/A} = (192/255)*3V = 2.26V > 2.03V \longrightarrow \mathbf{10000000}$$

Successive Approximation Example

***k* = 6** **3rd Approximation**

10100000 $\longrightarrow$ $V_{D/A} = (160/255)*3V = 1.88V < 2.03V$ $\longrightarrow$ **10100000**

***k* = 5** **4th Approximation**

10110000 $\longrightarrow$ $V_{D/A} = (176/255)*3V = 2.07V > 2.03V$ $\longrightarrow$ **10100000**

***k* = 4** **5th Approximation**

10101000 $\longrightarrow$ $V_{D/A} = (168/255)*3V = 1.98V < 2.03V$ $\longrightarrow$ **10101000**

Successive Approximation Example

***k* = 3** **6th Approximation**

10101100 $\longrightarrow$ $V_{D/A} = (172/255)*3V = 2.02V < 2.03V$ $\longrightarrow$ **10101100**

***k* = 2** **7th Approximation**

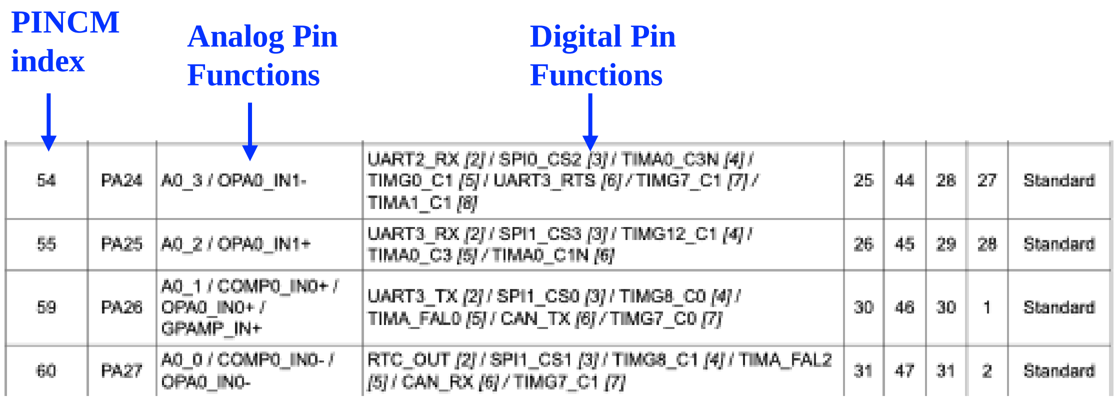
10101110 $\longrightarrow$ $V_{D/A} = (174/255)*3V = 2.04V > 2.03V$ $\longrightarrow$ **10101100**

***k* =1** **8th Approximation**

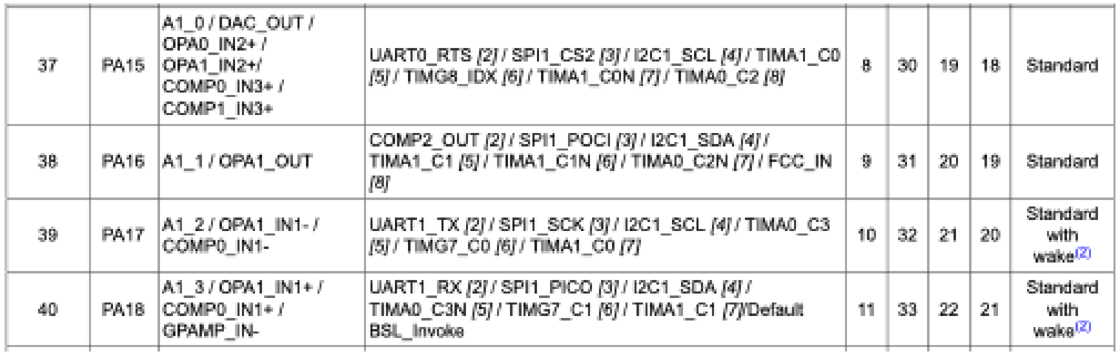
10101101 $\longrightarrow$ $V_{D/A} = (173/255)*3V = 2.035V > 2.03V$ $\longrightarrow$ **10101100**

Configuring ADC on the MSPM0

- MSPM0 has two 12-bit ADCs (ADC0 and ADC 1)
  - Each has 8 input channels
  - Means we can read 16 analog signals simultaneously


PINCM index
Analog Pin Functions
Digital Pin Functions

54	PA24	A0_3 / OPA0_IN1-	UART2_RX *[2]* / SPI0_CS2 *[3]* / TIMA0_C3N *[4]* / TIMG0_C1 *[5]* / UART3_RTS *[6]* / TIMG7_C1 *[7]* / TIMA1_C1 *[8]*	25	44	28	27	Standard
55	PA25	A0_2 / OPA0_IN1+	UART3_RX *[2]* / SPI1_CS3 *[3]* / TIMG12_C1 *[4]* / TIMA0_C3 *[5]* / TIMA0_C1N *[6]*	26	45	29	28	Standard
59	PA26	A0_1 / COMP0_IN0+ / OPA0_IN0+ / GPAMP_IN+	UART3_TX *[2]* / SPI1_CS0 *[3]* / TIMG8_C0 *[4]* / TIMA_FAL0 *[5]* / CAN_TX *[6]* / TIMG7_C0 *[7]*	30	46	30	1	Standard
60	PA27	A0_0 / COMP0_IN0- / OPA0_IN0-	RTC_OUT *[2]* / SPI1_CS1 *[3]* / TIMG8_C1 *[4]* / TIMA_FAL2 *[5]* / CAN_RX *[6]* / TIMG7_C1 *[7]*	31	47	31	2	Standard

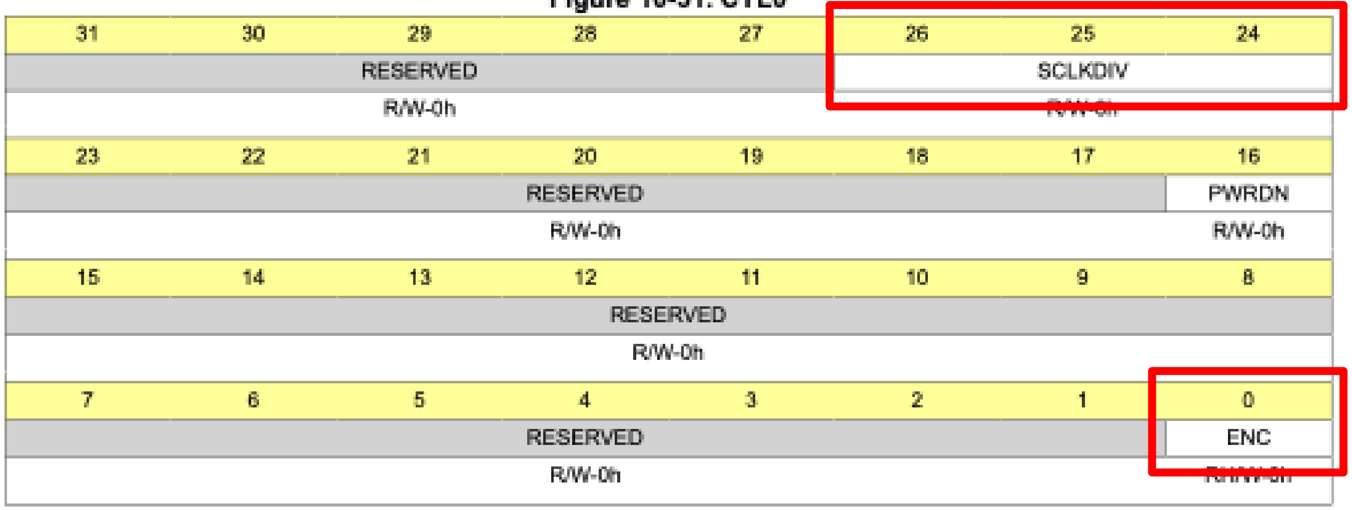


37	PA15	A1_0 / DAC_OUT / OPA0_IN2+ / OPA1_IN2+/ COMP0_IN3+ / COMP1_IN3+	UART0_RTS *[2]* / SPI1_CS2 *[3]* / I2C1_SCL *[4]* / TIMA1_C0 *[5]* / TIMG8_IDX *[6]* / TIMA1_C0N *[7]* / TIMA0_C2 *[8]*	8	30	19	18	Standard
38	PA16	A1_1 / OPA1_OUT	COMP2_OUT *[2]* / SPI1_POCI *[3]* / I2C1_SDA *[4]* / TIMA1_C1 *[5]* / TIMA1_C1N *[6]* / TIMA0_C2N *[7]* / FCC_IN *[8]*	9	31	20	19	Standard
39	PA17	A1_2 / OPA1_IN1- / COMP0_IN1-	UART1_TX *[2]* / SPI1_SCK *[3]* / I2C1_SCL *[4]* / TIMA0_C3 *[5]* / TIMG7_C0 *[6]* / TIMA1_C0 *[7]*	10	32	21	20	Standard with wake[2]
40	PA18	A1_3 / OPA1_IN1+ / COMP0_IN1+ / GPAMP_IN-	UART1_RX *[2]* / SPI1_PICO *[3]* / I2C1_SDA *[4]* / TIMA0_C3N *[5]* / TIMG7_C1 *[6]* / TIMA1_C1 *[7]*/Default BSL_Invoke	11	33	22	21	Standard with wake[2]

Configuring ADC on the MSPM0

- Key ADC configuration registers: CTL0

Figure 10-31. CTL0



31	30	29	28	27	26	25	24
RESERVED					SCLKDIV		
R/W-0h					R/W-0h		
23	22	21	20	19	18	17	16
RESERVED							PWRDN
R/W-0h							R/W-0h
15	14	13	12	11	10	9	8
RESERVED							
R/W-0h							
7	6	5	4	3	2	1	0
RESERVED							ENC
R/W-0h							R/W-0h

ENC: Enable conversion bit

SCLKDIV: Clock divider (to set ADC clock rate)

Configuring ADC on the MSPM0

- Key ADC configuration registers: MEMRES[y] (conversion result register)

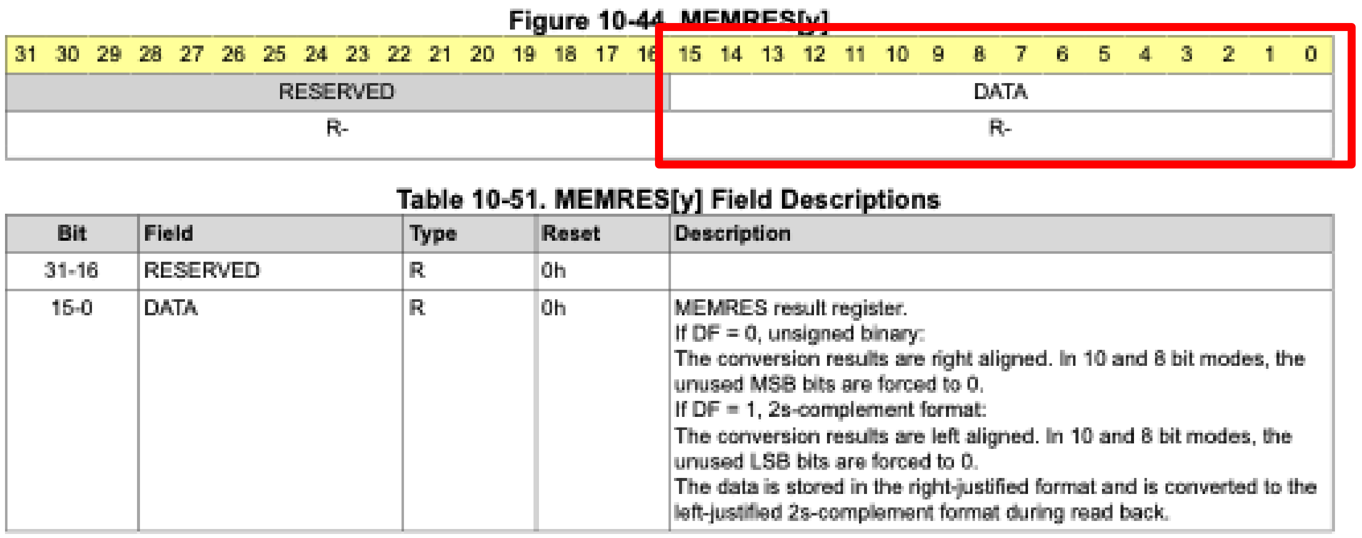


Figure 10-44. MEMRES[y]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																DATA															
R-																R-															

Table 10-51. MEMRES[y] Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	
15-0	DATA	R	0h	MEMRES result register. If DF = 0, unsigned binary: The conversion results are right aligned. In 10 and 8 bit modes, the unused MSB bits are forced to 0. If DF = 1, 2s-complement format: The conversion results are left aligned. In 10 and 8 bit modes, the unused LSB bits are forced to 0. The data is stored in the right-justified format and is converted to the left-justified 2s-complement format during read back.

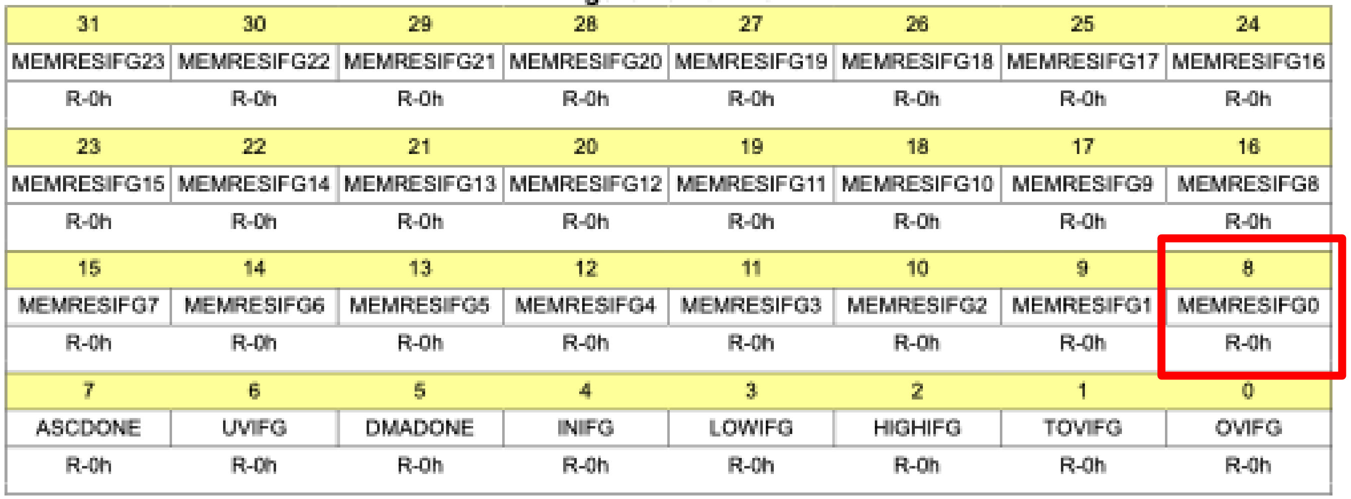
DATA (16 bits): Holds the results of each conversion

Note: There is one MEMRES[y] register for each channel

Configuring ADC on the MSPM0

- Key ADC configuration registers: RIS (raw interrupt status)

Figure 10-13. RIS

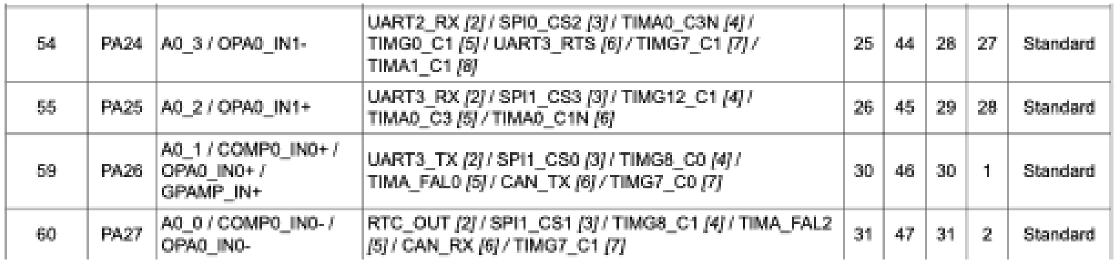


31	30	29	28	27	26	25	24
MEMRESIFG23	MEMRESIFG22	MEMRESIFG21	MEMRESIFG20	MEMRESIFG19	MEMRESIFG18	MEMRESIFG17	MEMRESIFG16
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
23	22	21	20	19	18	17	16
MEMRESIFG15	MEMRESIFG14	MEMRESIFG13	MEMRESIFG12	MEMRESIFG11	MEMRESIFG10	MEMRESIFG9	MEMRESIFG8
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
15	14	13	12	11	10	9	8
MEMRESIFG7	MEMRESIFG6	MEMRESIFG5	MEMRESIFG4	MEMRESIFG3	MEMRESIFG2	MEMRESIFG1	MEMRESIFG0
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h
7	6	5	4	3	2	1	0
ASCDONE	UVIFG	DMADONE	INIFG	LOWIFG	HIGHIFG	TOVIFG	OVIFG
R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h	R-0h

Each MEMRESIFG bit is an interrupt flag specifying that new conversion data available in MEMRES register

Configuring ADC on MSPM0

- ADC can be configured for various types of functionality
  - Auto-trigger mode
  - Manual-trigger mode
  - See reference manual for descriptions of various modes
- Today we will discuss how to configure ADC to perform single conversion (repetitively)
  - Note: When using pin as analog input, you don't need to set pin functionality in PINCM register



54	PA24	A0_3 / OPA0_IN1-	UART2_RX *[2]* / SPI0_CS2 *[3]* / TIMA0_C3N *[4]* / TIMG0_C1 *[5]* / UART3_RTS *[6]* / TIMG7_C1 *[7]* / TIMA1_C1 *[8]*	25	44	28	27	Standard
55	PA25	A0_2 / OPA0_IN1+	UART3_RX *[2]* / SPI1_CS3 *[3]* / TIMG12_C1 *[4]* / TIMA0_C3 *[5]* / TIMA0_C1N *[6]*	26	45	29	28	Standard
59	PA26	A0_1 / COMP0_IN0+ / OPA0_IN0+ / GPAMP_IN+	UART3_TX *[2]* / SPI1_CS0 *[3]* / TIMG8_C0 *[4]* / TIMA_FAL0 *[5]* / CAN_TX *[6]* / TIMG7_C0 *[7]*	30	46	30	1	Standard
60	PA27	A0_0 / COMP0_IN0- / OPA0_IN0-	RTC_OUT *[2]* / SPI1_CS1 *[3]* / TIMG8_C1 *[4]* / TIMA_FAL2 *[5]* / CAN_RX *[6]* / TIMG7_C1 *[7]*	31	47	31	2	Standard

Configuring ADC on MSPM0

- Step 1: Enable power to ADC module

```
void DL_ADC12_enablePower (ADC12_Regs *adc12)
```

- Step 2: Set clock configuration
  - Use ADC clock configuration struct

```
void DL_ADC12_setClockConfig(ADC12_Regs * adc12, DL_ADC12_ClockConfig * config)
```

```
typedef struct {
   DL_ADC12_CLOCK clockSet ;
   DL_ADC12_CLOCK_FREQ_RANGE freqRange ;
   DL_ADC12_CLOCK_DIVIDE divideRation ;
} DL_ADC12_ClockConfig ;
```

Configuring ADC on MSPM0

- Step 3: Configure conversion memory

```
void DL_ADC12_configConversionMem (ADC12_Regs *adc12, DL_ADC12_MEM_IDX idx,
             uint32_t chansel, uint32_t vref, uint32_t stime, uint32_t
             avgen, uint32_t bcsen, uint32_t trig, uint32_t wincomp)
```

  – idx is destination is register which will hold results (e.g., DL_ADC12_MEM_IDX_0, DL_ADC12_MEM_IDX_1, etc.)
  – chansel is ADC channel you are using (e.g., DL_ADC12_INPUT_CHAN_0, DL_ADC12_INPUT_CHAN_1, etc.)
  – vref is reference voltage
  – trig is triggering configuration – usually set to DL_ADC12_TRIGGER_MODE_AUTO_NEXT
  – Other options detailed in Driverlib documentation

Configuring ADC on MSPM0

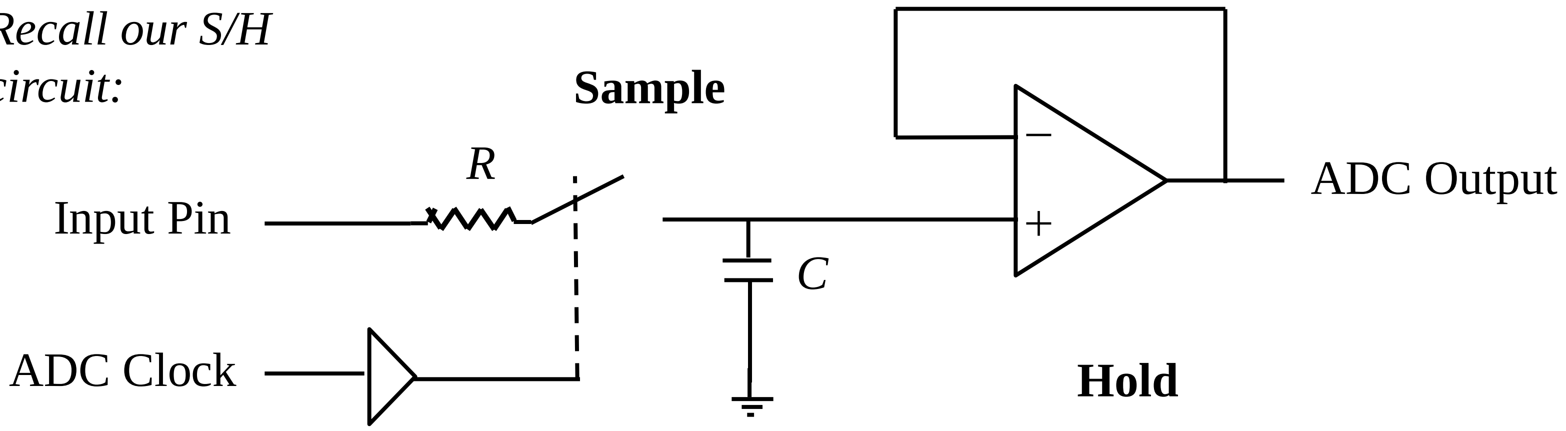
- Step 4: Set the sample time (in number of ADC clock cycles)

```
void DL_ADC12_setSampleTime0(ADC12_Regs * adc12, uint16_t adcclks)
```

Configuring ADC on MSPM0

- Note about clock rates and sampling times:
  - Running at higher clock rates allows faster sampling times, but this can decrease accuracy
  - Clock rate and sampling time can be set separately to balance desired update rate with desired accuracy

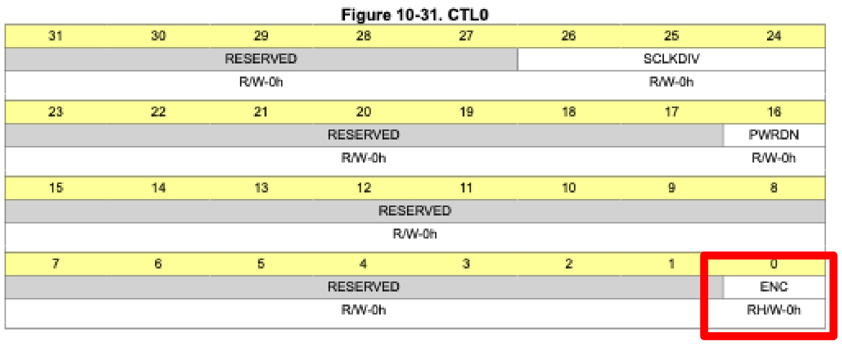
Recall our S/H circuit:



Configuring ADC on MSPM0

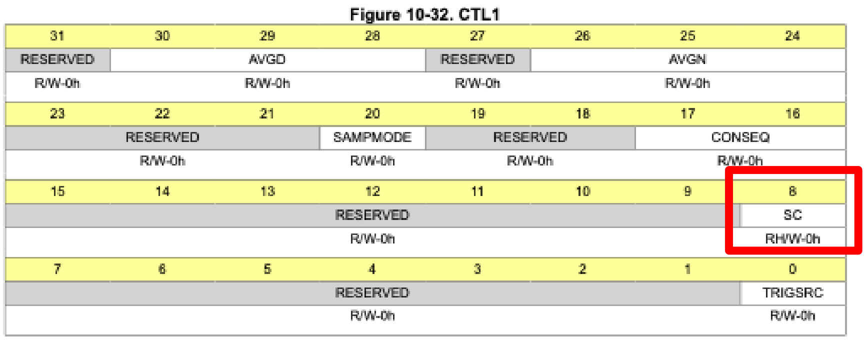
- To enable ADC we need to do two things:
  - Setting ENC bit enables ADC operation
  - Setting SC (start conversion) begins a single ADC conversion
    - *SC bit is automatically cleared by hardware – it is just a trigger bit*

Figure 10-31. CTL0



31	30	29	28	27	26	25	24
RESERVED					SCLKDIV		
R/W-0h					R/W-0h		
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED							PWRDN
R/W-0h							R/W-0h
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED							
R/W-0h							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESERVED							ENC
R/W-0h							RH/W-0h

Figure 10-32. CTL1



31	30	29	28	27	26	25	24
RESERVED	AVGD			RESERVED	AVGN		
R/W-0h	R/W-0h			R/W-0h	R/W-0h		
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
RESERVED			SAMPMODE	RESERVED		CONSEQ	
R/W-0h			R/W-0h	R/W-0h		R/W-0h	
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RESERVED							SC
R/W-0h							RH/W-0h
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESERVED							TRIGSRC
R/W-0h							R/W-0h

```
void DL_ADC12_enableConversion(ADC12_Regs *adc)
```

Sets the ENC bit

```
void DL_ADC12_startConversion(ADC12_Regs *adc12)
```

Sets the SC bit

Configuring ADC on MSPM0

- We can configure ADC to interrupt every time sample operation is complete

```
void DL_ADC12_enableInterrupt (ADC12_Regs *adc12, uint32_t interruptMask)
```

$\uparrow$

e.g., `DL_ADC12_INTERRUPT_MEM0_RESULT_LOADED`

- Alternatively, we can poll DL_ADC12_getStatus() and wait until sampling is completed

```
uint32_t DL_ADC12_getStatus(ADC12_Regs * adc12)
```

Checks BUSY *bit in* ADC STATUS *register which is set when ADC is currently performing conversion. Returns DL_ADC12_STATUS_CONVERSION_ACTIVE if BUSY bit is set.*

Retrieving ADC Results

- Use Driverlib function to get conversion results

```
Uint16_t DL_ADC12_getMemResult (ADC12_Regs *adc12, DL_ADC12_MEM_IDX idx)
```

Example ADC code

- Let’s look at an example code implementing repeated ADC conversions