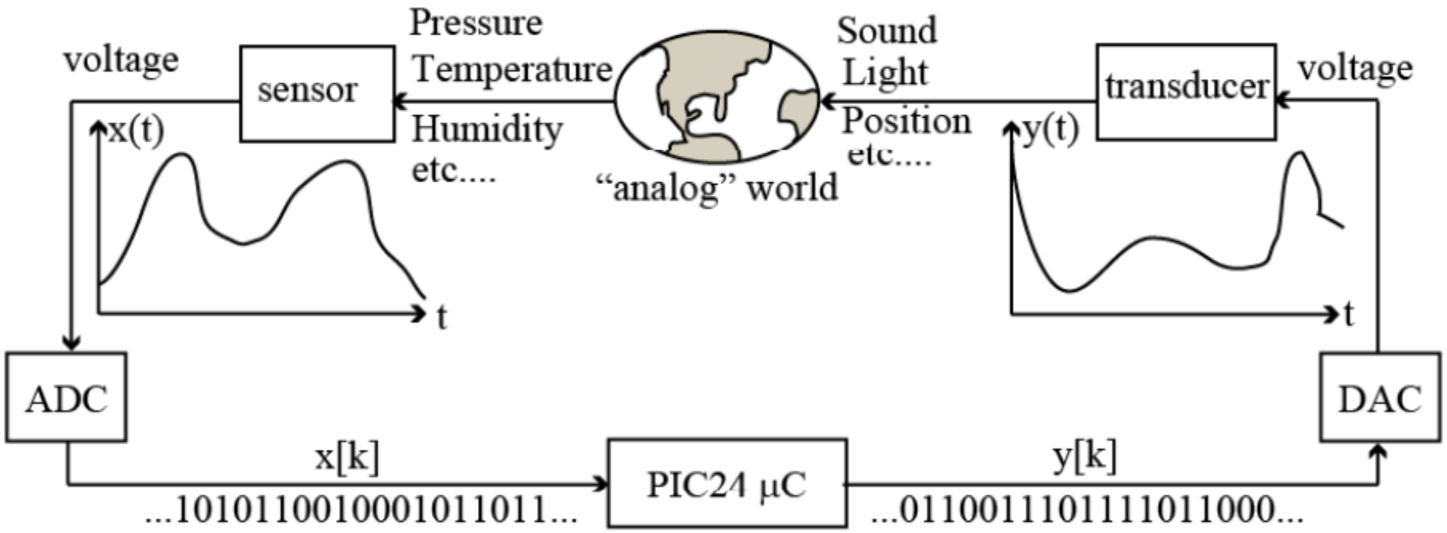
COE 115

Lecture 10

Digital Signal Processing



Analog-to-Digital Converter (ADC) converts an input analog value to an output digital representation

This digital data is processed by a microprocessor and output to a **Digital-to-Analog Converter (DAC)** that converts an input binary value to an output voltage.

Applications

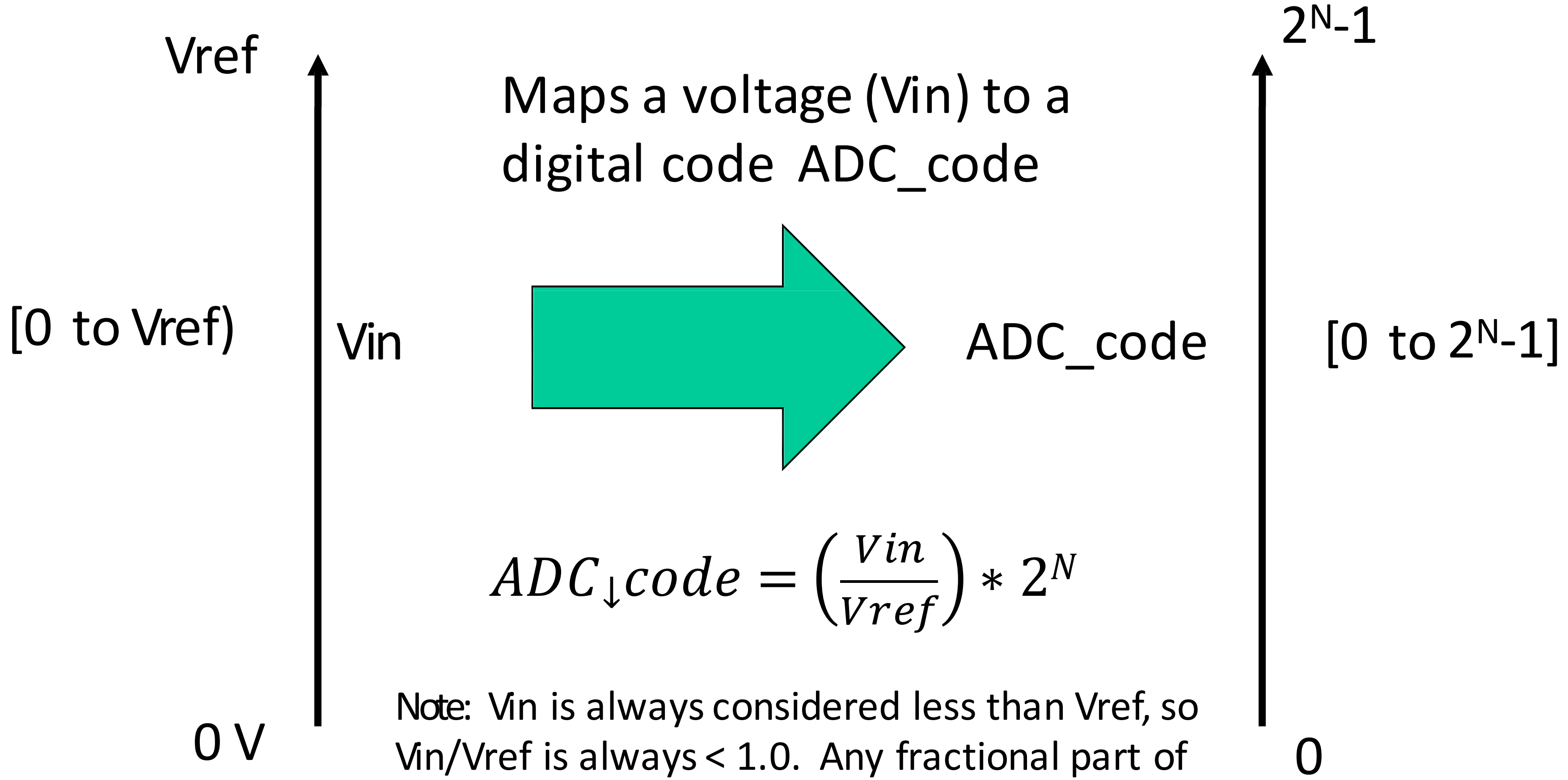
- **Audio**
  - Speech Recognition
  - Special effects (reverb, noise cancellation, etc)
- **Video**
  - Filtering
  - Special Effects
  - Compression
- **Data Logging**

Vocabulary

- **ADC (Analog-to-Digital Converter)**
  Converts an analog signal (voltage/current) to a digital code
- **DAC (Digital-to-Analog Converter)**
  Converts a digital code to an analog value (voltage/current
- **Sample Period**
  Time between each conversion. Typically, samples are taken at fixed rate
- **Reference Voltage (Vref)**
  Analog signals varies between 0V and Vref, or between -+ Vref
- **Resolution**
  Number of bits used for conversion (8 bits, 10 bits, 12 bits, 16 bits, etc.)
- **Conversion Time**
  The time it takes for an analog-to-digital conversion

N-bit ADC

Maps a voltage (Vin) to a digital code ADC_code

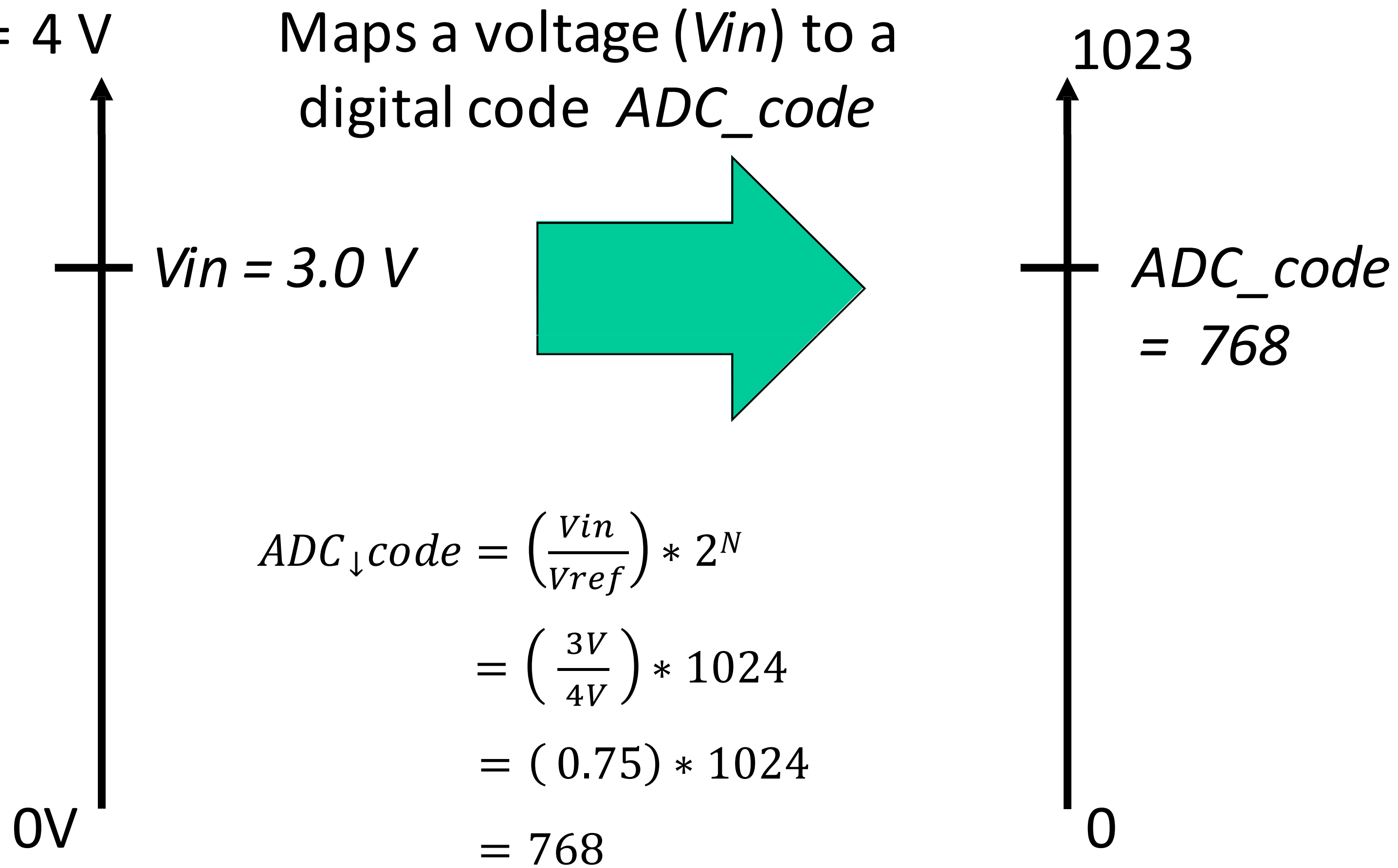


$$ADC_{\downarrow}code = \left(\frac{Vin}{Vref}\right) * 2^N$$

Note: Vin is always considered less than Vref, so Vin/Vref is always < 1.0. Any fractional part of the code is truncated.

Example 1: A 10-bit ADC

Maps a voltage (*Vin*) to a digital code *ADC_code*



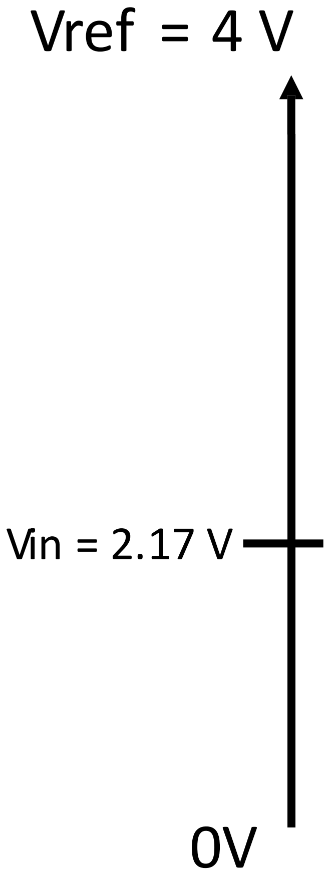
$$ADC_{\downarrow}code = \left(\frac{Vin}{Vref}\right) * 2^N$$

$$= \left(\frac{3V}{4V}\right) * 1024$$

$$= (0.75) * 1024$$

$$= 768$$

Example 2: A 10-bit ADC



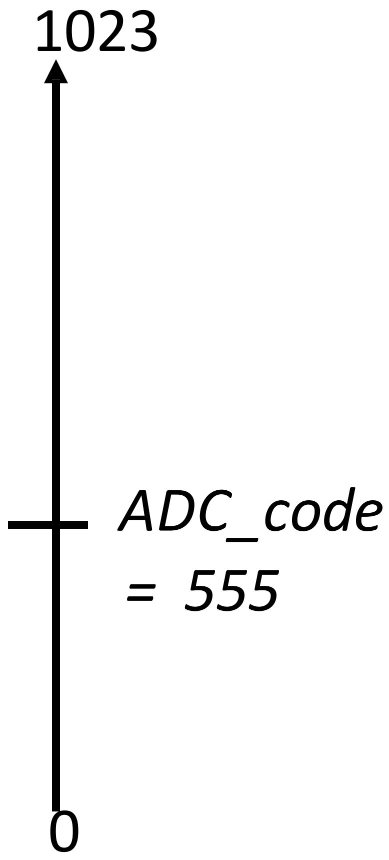
Maps a voltage (*Vin*) to a digital code *ADC_code*

$$ADC_{\downarrow}code = \left(\frac{Vin}{Vref}\right) * 2^{N}$$

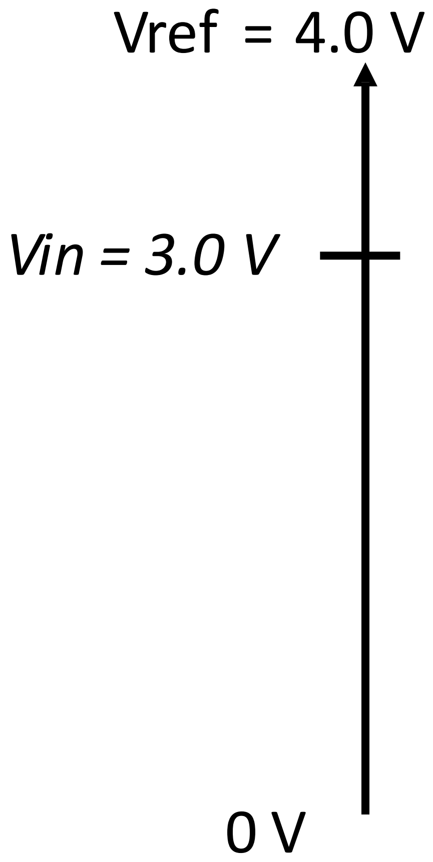
$$= \left(\frac{2.17V}{4V}\right) * 1024$$

$$= (0.5425) * 1024$$

$$= 555.52 \approx 555$$

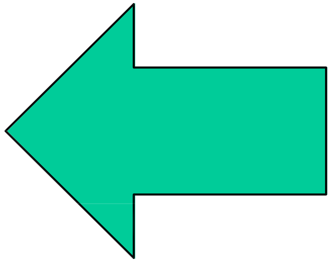


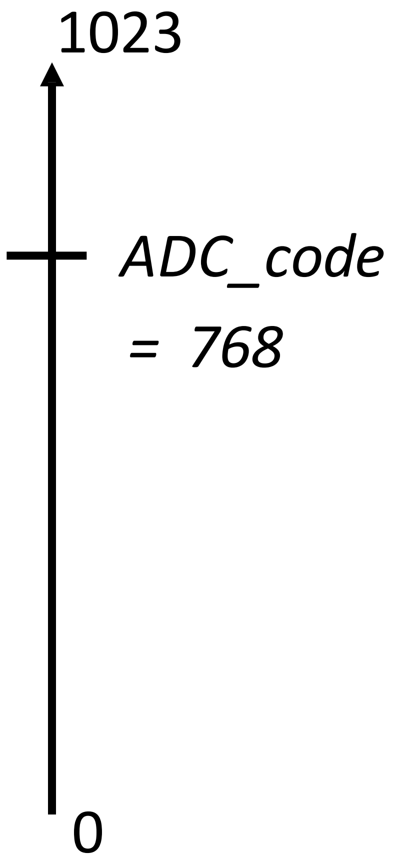
Example 3: Code to Voltage



$$ADC_{\downarrow}code = \left(\frac{Vin}{Vref}\right) * 2^{N}$$

$$\frac{ADC_{\downarrow}code}{2^{N}} * Vref = \text{Vin}$$





$$Vin = \left(\frac{ADC_{\downarrow}code}{2^{N}}\right) * Vref$$

$$= \left(\frac{768}{1024}\right) * 4\,V$$

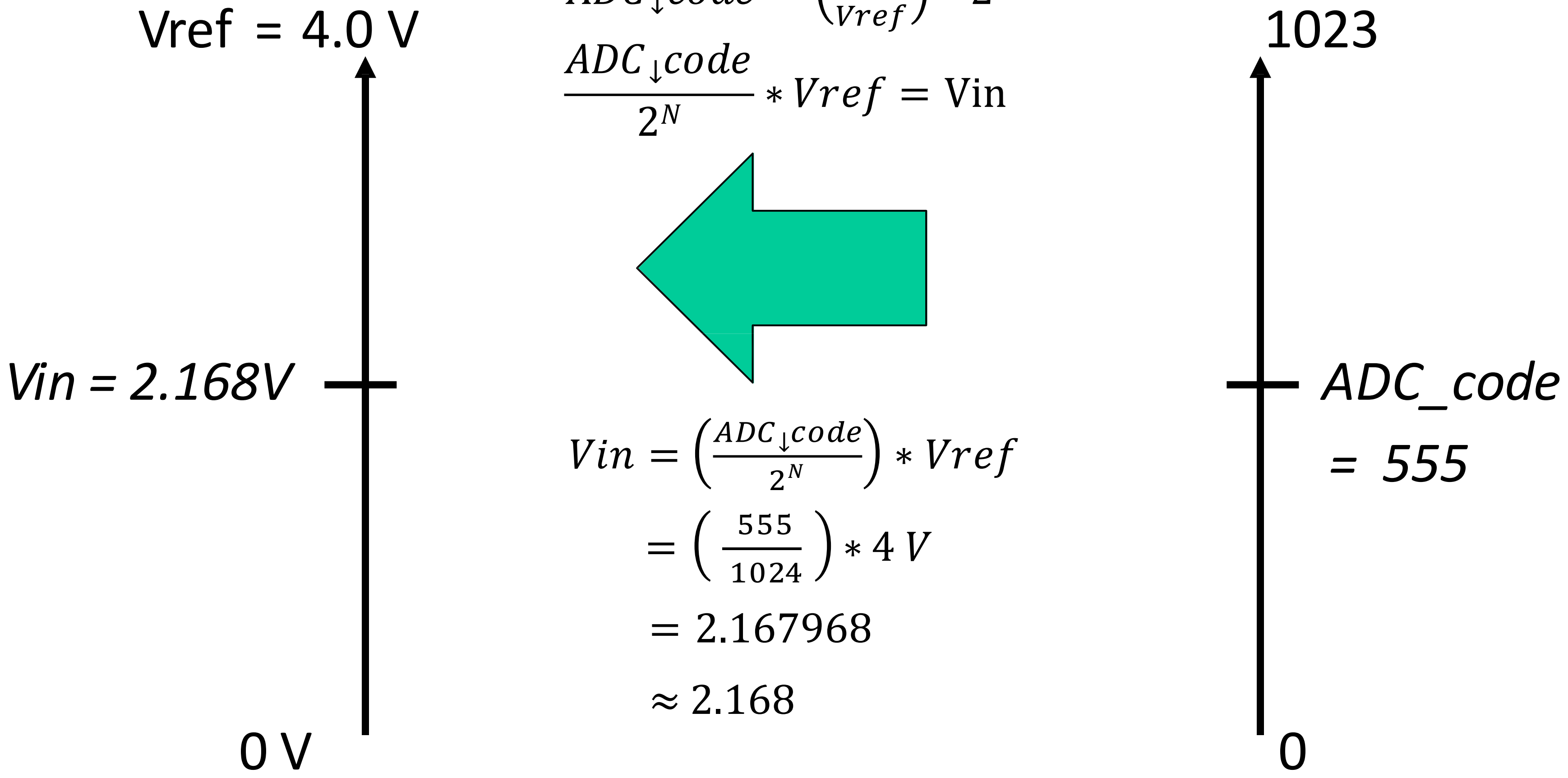
$$= (0.75) * 4\,V$$

$$= 768$$

Example 4: Code to Voltage

$$ADC_{\downarrow}code = \left(\frac{Vin}{Vref}\right) * 2^N$$

$$\frac{ADC_{\downarrow}code}{2^N} * Vref = \text{Vin}$$



$$Vin = \left(\frac{ADC_{\downarrow}code}{2^N}\right) * Vref$$

$$= \left(\frac{555}{1024}\right) * 4\,V$$

$$= 2.167968$$

$$\approx 2.168$$

ADC Resolution

For an N-bit ADC, the smallest input voltage that can resolved is 1 LSB, or:

$$1\ LSB = \frac{1}{2^N} * (Vref^+ - Vref^-)$$

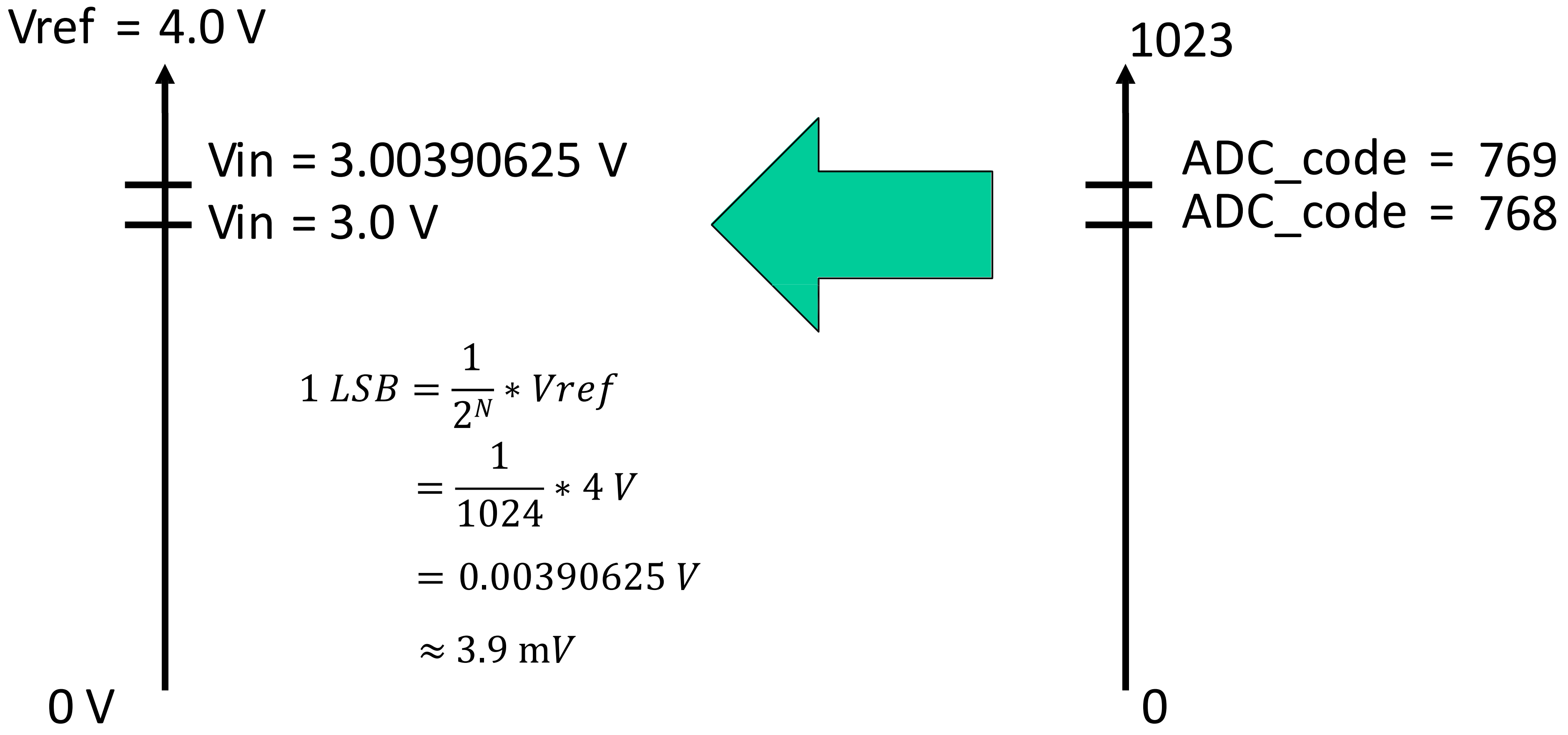
Where $Vref^+$ is the positive reference voltage and $Vref^-$ is the negative reference voltage. We will use $Vref^-$ = 0 V, and refer to $Vref^+$ as $Vref$, so this simplifies to

$$1\ LSB = \frac{1}{2^N} * Vref$$

For $Vref$ = 4V and N = 4, what is 1 LSB?

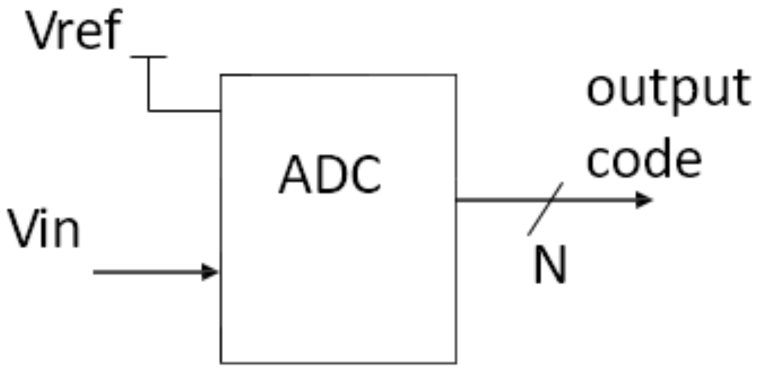
$$1\ LSB = \frac{1}{2^4} * 4\ V = \frac{1}{16} * 4\ V = \mathbf{0.25\ V}$$

Example 5: 10-bit ADC Resolution



$$1\,LSB = \frac{1}{2^N} * Vref$$
$$= \frac{1}{1024} * 4\,V$$
$$= 0.00390625\,V$$
$$\approx 3.9\ \mathrm{m}V$$

ADC and DAC Equations



ADC: Vin = input voltage

Vref^+ = reference voltage = Vref

Vref^- = 0 V

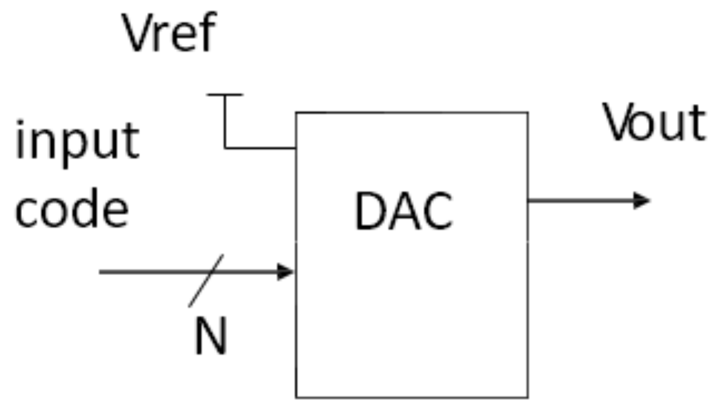
N = Number of bits of precision

$$\frac{\text{Vin}}{\text{Vref}} * 2^{\text{N}} = \text{output}_{\downarrow}\text{code}$$

$$\text{Vin} = \frac{\text{output}_{\downarrow}\text{code}}{2^{\text{N}}} * \text{Vref}$$

$$1\ \text{LSB} = \frac{Vref}{2^{\text{N}}}$$

ADC and DAC Equations



DAC: Vout = output voltage

Vref$^+$ = reference voltage = Vref

Vref$^-$ = 0 V

N = Number of bits of precision

$$\frac{\mathrm{Vout}}{\mathrm{Vref}} * 2^{\mathrm{N}} = \mathrm{input}_{\downarrow}\mathrm{code}$$

$$\mathrm{Vout} = \frac{\mathrm{input}_{\downarrow}\mathrm{code}}{2^{\mathrm{N}}} * \mathrm{Vref}$$

$$1\ \mathrm{LSB} = \frac{Vref}{2^{\mathrm{N}}}$$

Sample ADC and DAC Computations

1. If Vref = 5V, and a 10-bit A/D output code is 0x12A, what is the ADC input voltage?

$$\mathrm{Vin} = \frac{\mathrm{output}_{\downarrow}\mathrm{code}}{2^{\mathrm{N}}} * \mathrm{Vref} = \frac{\mathrm{0x12A}}{2^{10}} * 5\mathrm{V} = \frac{298}{1024} * 5\mathrm{V} = \mathbf{1.46\ V\ (ADC\ Vin)}$$

2. If Vref = 5V and an 8-bit DAC input code is 0xA9, what is the DAC output voltage?

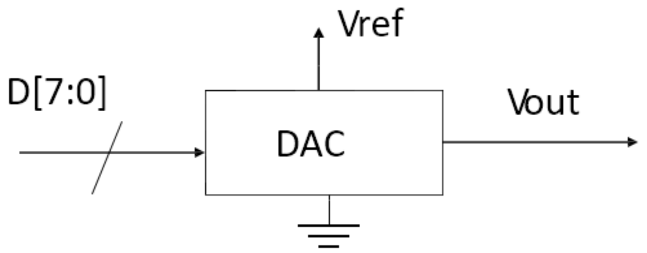
$$\mathrm{Vout} = \frac{\mathrm{input}_{\downarrow}\mathrm{code}}{2^{\mathrm{N}}} * \mathrm{Vref} = \frac{\mathrm{0xA9}}{2^{8}} * 5\mathrm{V} = \frac{169}{256} * 5\mathrm{V} = \mathbf{3.3V\ (DAC\ Vout)}$$

3. If Vref = 4V and an 8-bit A/D input voltage is 2.35V, what is the ADC output code?

$$\mathrm{output}_{\downarrow}\mathrm{code} = \frac{\mathrm{V}in}{\mathrm{Vref}} * 2^{\mathrm{N}} = \frac{2.35\ \mathrm{V}}{4\ V} * 2^{8} = 0.5875 * 256 = 150.4 \approx 150 = \mathbf{0x96\ (ADC\ output_{\downarrow}code)}$$

Digital-to-Analog Conversion

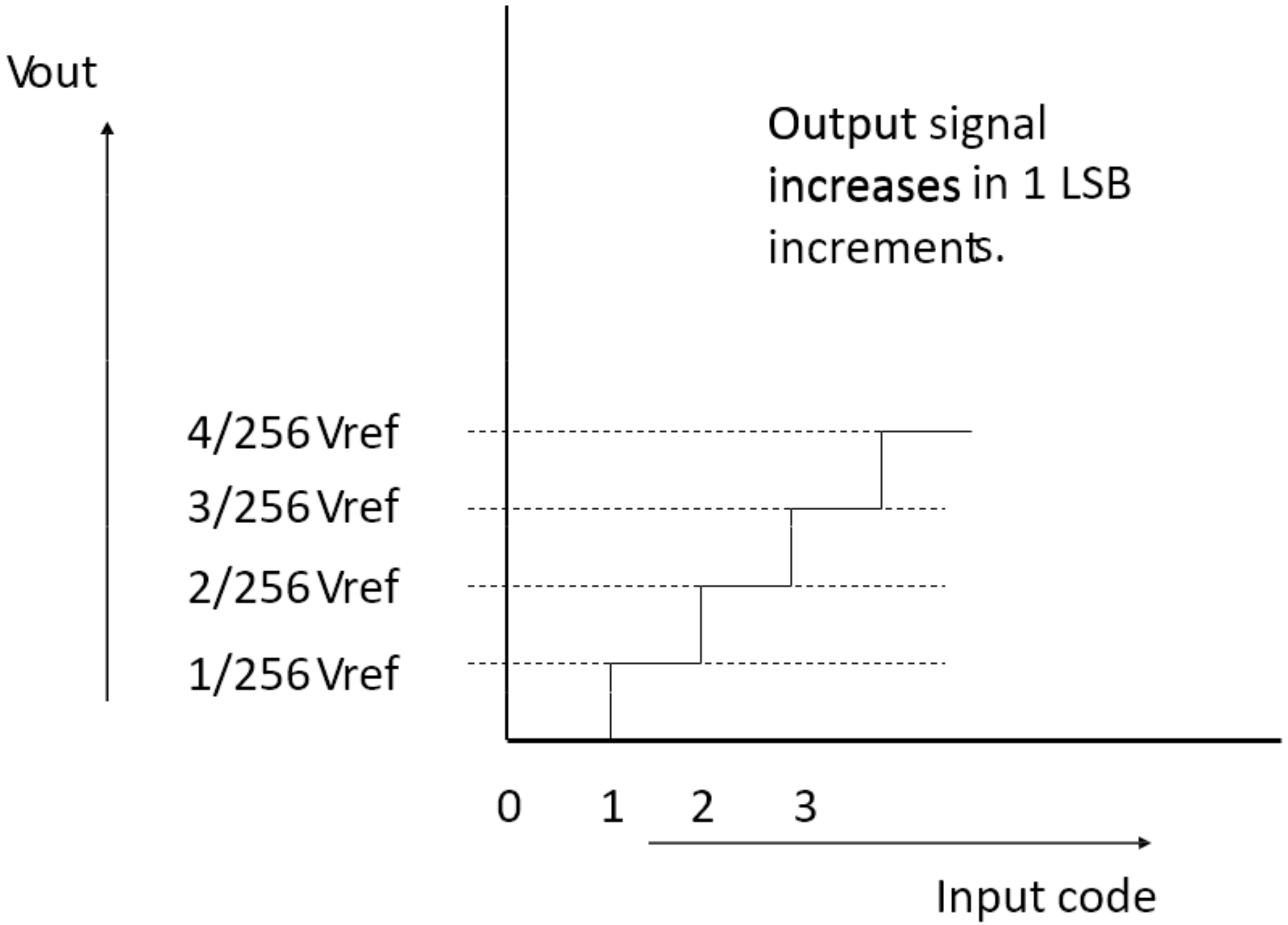
For a particular binary code, output a voltage between 0 and Vref



Assume a DAC that uses an unsigned input code, with $0 \leq Vout < Vref$. Then

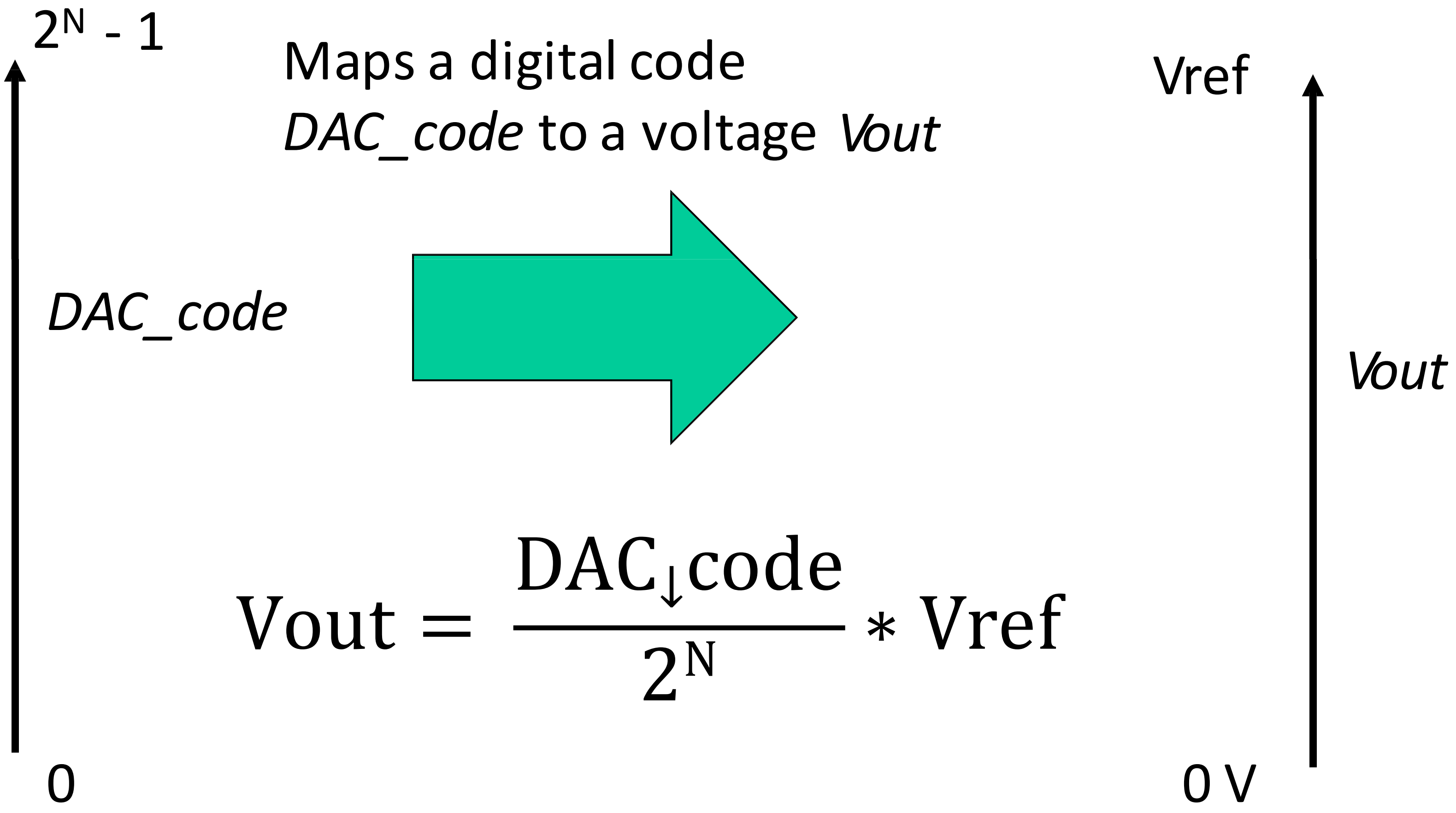
D= 0000 0000	Vout = 0V	
D= 0000 0001	Vout = Vref(1/256)	(one LSB)
D = 0000 0010	Vout = Vref(2/256)	
...		
D = 1111 1111	Vout = Vref(255/256)	(full scale)

DAC Output Plot



Output signal increases in 1 LSB increments.

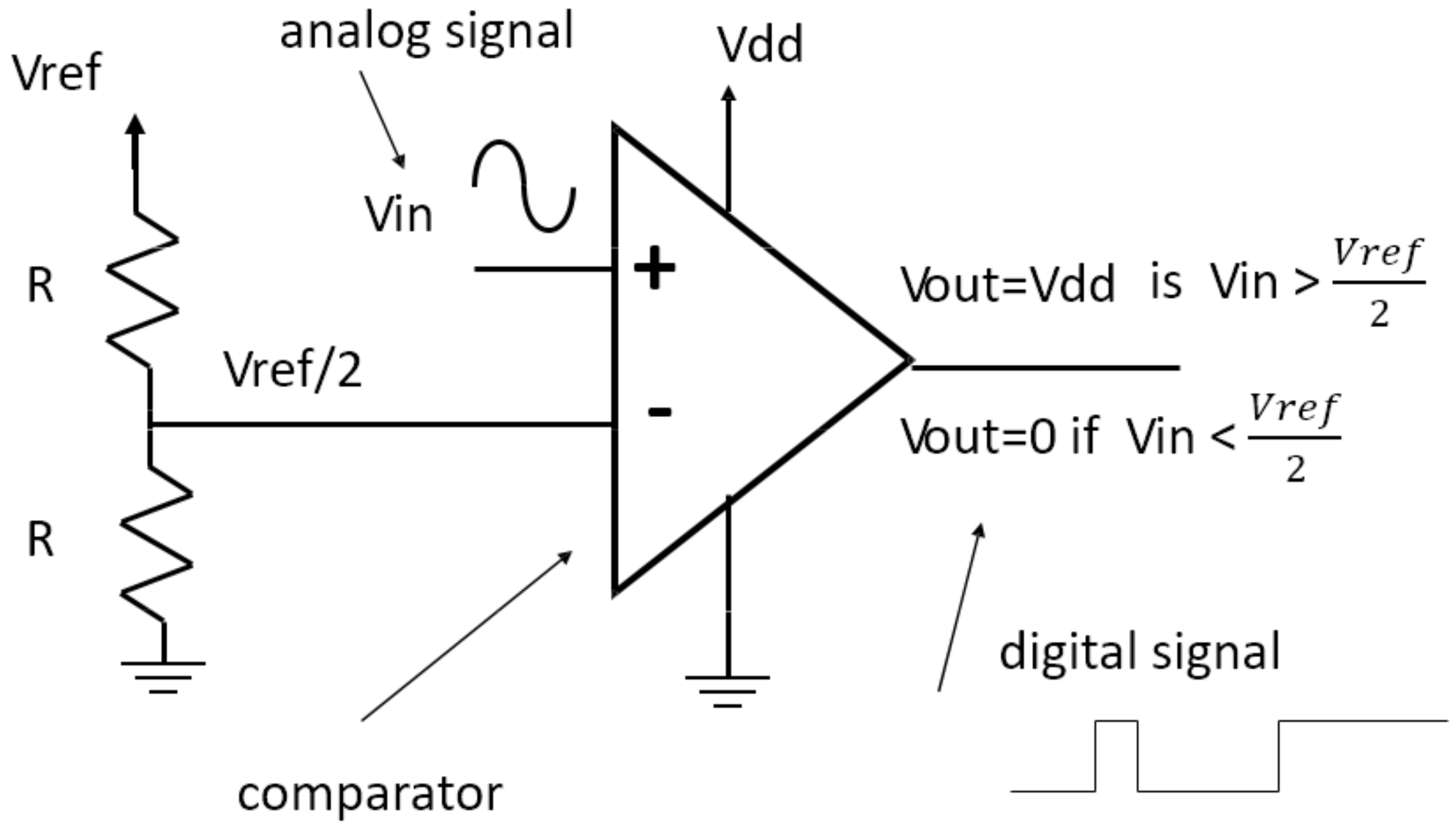
An N-bit DAC



Maps a digital code *DAC_code* to a voltage *Vout*

$$\text{Vout} = \frac{\text{DAC}_{\downarrow}\text{code}}{2^{\text{N}}} * \text{Vref}$$

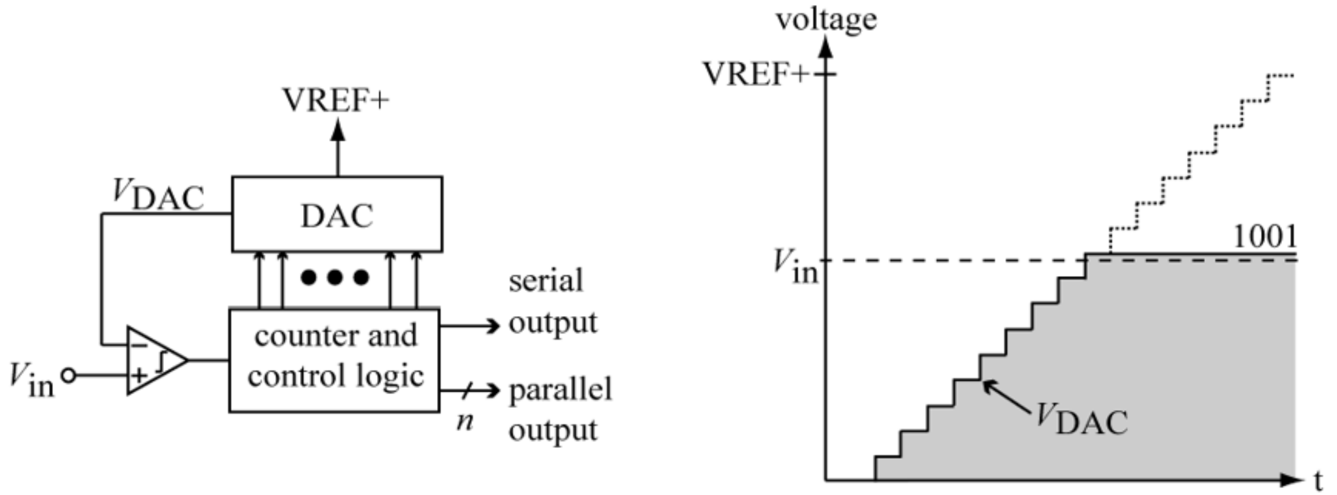
A 1-bit ADC



Vout=Vdd is Vin > $\frac{Vref}{2}$

Vout=0 if Vin < $\frac{Vref}{2}$

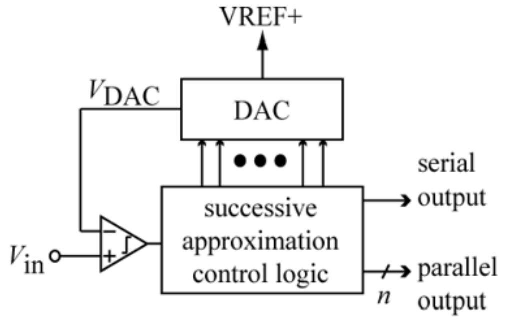
Counter Ramp ADC



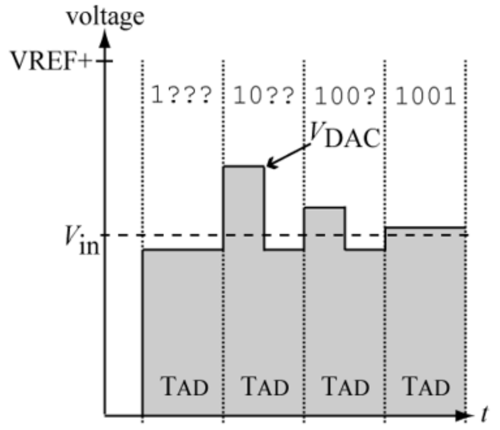
(a) Structure (b) Representative conversion cycle

Control logic use a **counter** to apply successive codes 0,1,2,3,4... to DAC (Digital-to-Analog Converter) until DAC output is greater than Vin. This is **SLOW,** and have to allocate the **worst case time** for each conversion, which is 2^N clock cycles for an N-bit ADC.

Successive Approximation ADC



(a) Structure



(b) Representative conversion cycle

- Initially set V_{DAC} to ½ Vref
- See if Vin is higher or lower
  - if Vin > ½ Vref, then next guess is between Vref and ½ Vref
  - else, next guess is between ½ Vref and GND
- Do this for each bit of the ADC, therefore, this takes N clock cycles

Successive Approximation Example

Given a 4-bit Successive Approximation ADC, and Vref = 4V.

Let Vin = 3.14159 V. Clear DAC input to 0b0000.

1. First guess, DAC input = 0b**1**000 = 8, V_{DAC} = $8/2^4 *$ 4 V = 8/16 * 4 V = 2 V

   V_{DAC} (2 V) < Vin, so guess of **'1'** for MSb of DAC was **correct**

2. Set next bit of DAC to '1', DAC input = 0b1**1**00 = 12, so V_{DAC} = 12/16*4= 3V

   V_{DAC} (3 V) < Vin, so guess of **'1'** for bit-2 of DAC was **correct**

3. Set next bit of DAC to '1', DAC input = 0b11**1**0 = 14, so V_{DAC} = 14/16*4= 3.5V.

   V_{DAC} (3.5 V) > Vin, so guess of **'1'** for bit-1 of DAC was **INCORRECT**. **Therefore reset bit-1 to 0** -> DAC input = 0b11**0**0

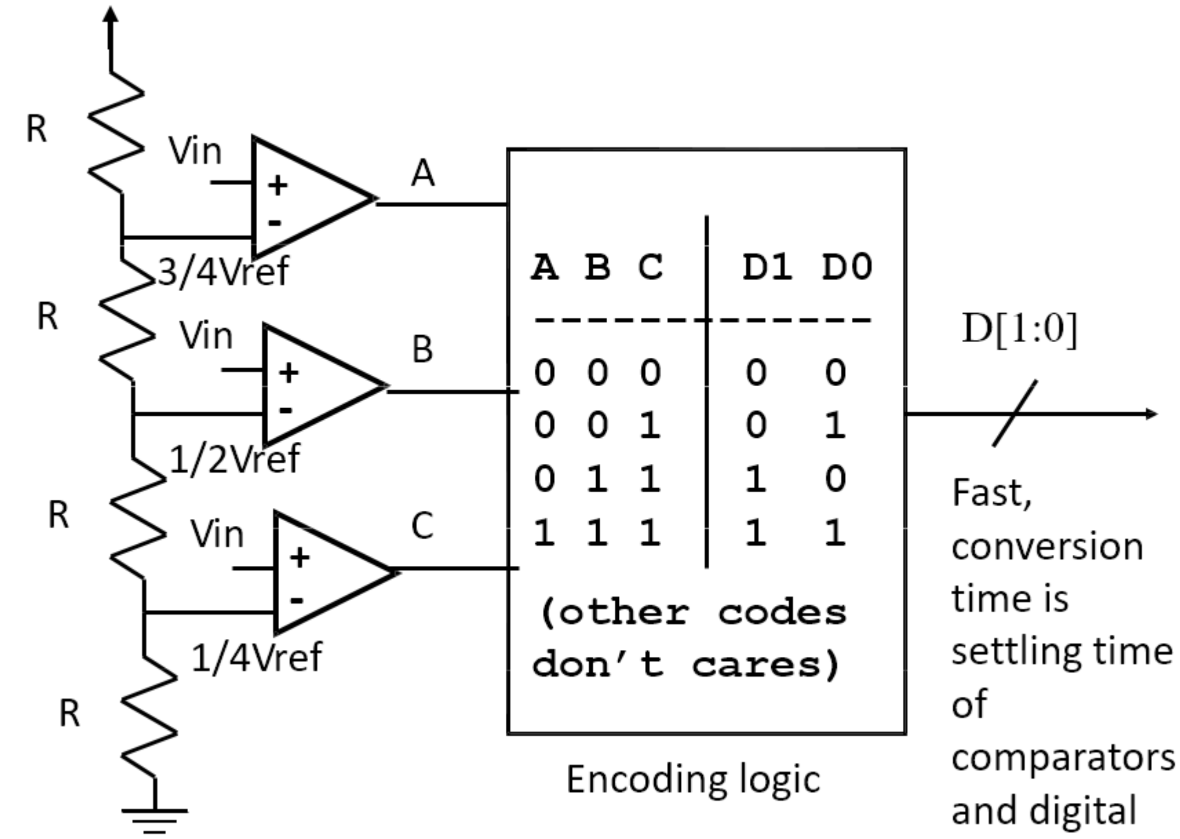
4. Set last bit of DAC to '1', DAC input = 0b110**1** = 13, so V_{DAC} = 13/16*4 =3.25V

   V_{DAC} (3.25 V) > Vin, so guess of **'1'** for last bit of DAC was **INCORRECT**. **Therefore reset last bit to 0** -> DAC input = 0b110**0**

Final ADC **output code** is **0b1100**

Check Result: output_code = Vin/Vref * 2^N = 3.14159/4 * 16 = 12.57 ≈ 12 (truncated).

A 2-bit Flash ADC



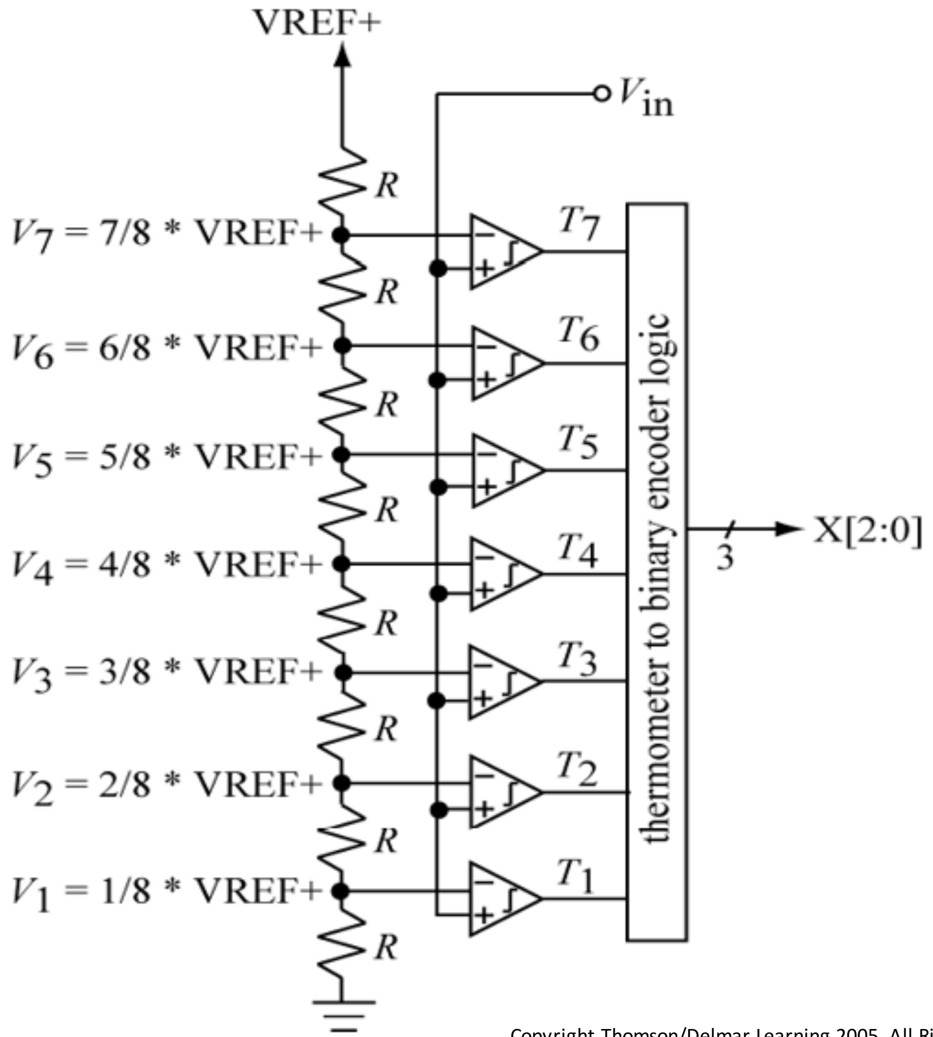
A	B	C	D1	D0
0	0	0	0	0
0	0	1	0	1
0	1	1	1	0
1	1	1	1	1

(other codes don't cares)

Encoding logic

Fast, conversion time is settling time of comparators and digital logic.

3-bit Flash ADC


VREF+
V_{in}
R
$V_7 = 7/8 * \text{VREF+}$
$V_6 = 6/8 * \text{VREF+}$
$V_5 = 5/8 * \text{VREF+}$
$V_4 = 4/8 * \text{VREF+}$
$V_3 = 3/8 * \text{VREF+}$
$V_2 = 2/8 * \text{VREF+}$
$V_1 = 1/8 * \text{VREF+}$
T_7
T_6
T_5
T_4
T_3
T_2
T_1
thermometer to binary encoder logic
3
X[2:0]

ADC Architecture Summary

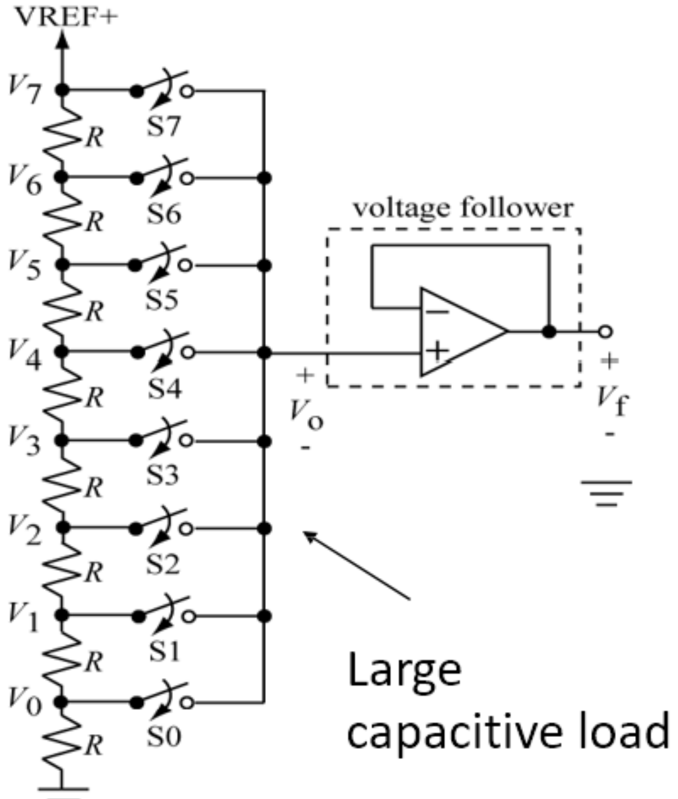
- **Flash ADCs**
  - Fastest possible conversion time
  - Requires the most transistors of any architecture
  - N-bit converter requires 2N-1 comparators
  - Commercially available flash converters up to 12 bits
  - Conversion is done in 1 clock cycle
- **Successive Approximation ADC**
  - Use only one comparator
  - Take one clock cycle per bit
  - High precision (16-bit converters are available)

Commercial ADCs

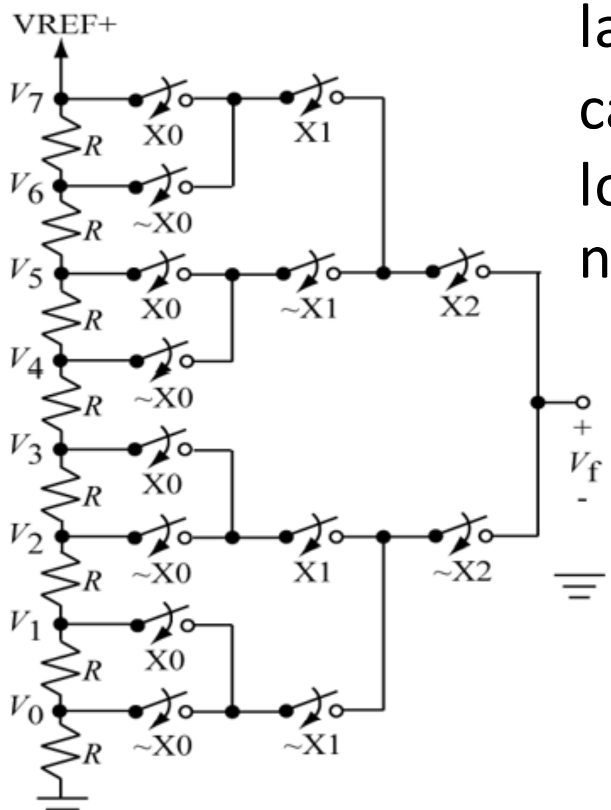
- Key timing parameter is *conversion time* – how long does it take to produce a digital output once a conversion is started
- Up to 16-bit ADCs available
- Separated into fast/medium/low speed families
  - Serial interfaces common on medium/low speed ADCs
- For high-precision ADCs, challenge is keeping system noise from affecting conversion
  - Assume a 16-bit DAC, and a 4.1V reference, then

    1 LSB = $4.1/2^{16}$ = 62 μV.

Flash DAC



(a) With Switch Decoder

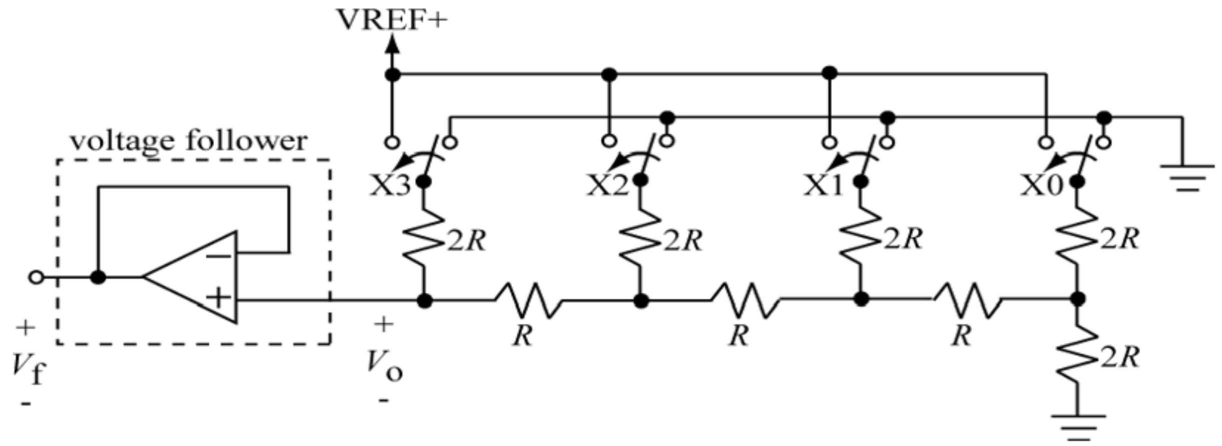


(b) Without Switch Decoder

Eliminates large capacitive load at one node.

N-bit DAC requires 2^N resistors!

R-2R Ladder DAC



Resistor ladder divides the Vref voltage to a binary weighted value 4-bit value, with the 4-bits equal to X3 X2 X1 X0

If the switch Xn is connected to Vref, then that bit value is ‘1’, if the switch Xn is not connected to Vref, then that bit value is ‘0’.

Majority of DACs use this architecture as requires far less resistors than flash DACs.

Sample DAC Computations

1. If Vref = 5V and the 8-bit input code is is 0x8A, what is the DAC output voltage?

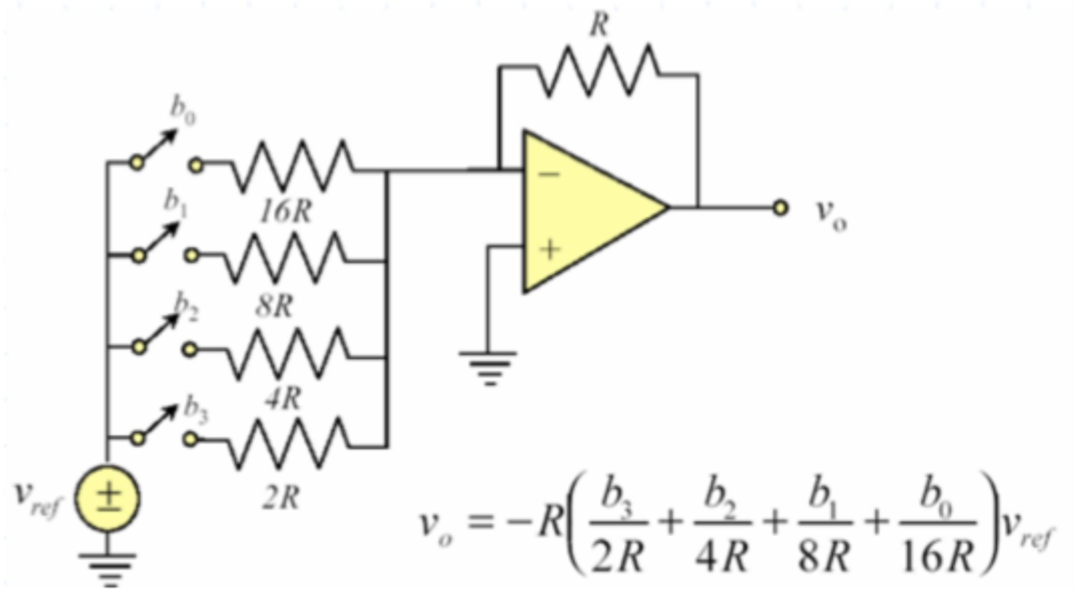
$$\frac{input_{\downarrow}code}{2^N} * Vref = \frac{0x8A}{2^8} * 5V = \frac{138}{256} * 5V = \mathbf{2.70\ V\ (Vout)}$$

2. If Vref = 4V, and the DAC output voltage is 1.25 V, what is the 8-bit input code?

$$\frac{Vout}{Vref} * 2^N = \frac{1.25\ V}{4\ V} * 28 = 0.3125\ * 256 = 80 = \mathbf{0x50\ (input_{\downarrow}code)}$$

Inverting Summer DAC

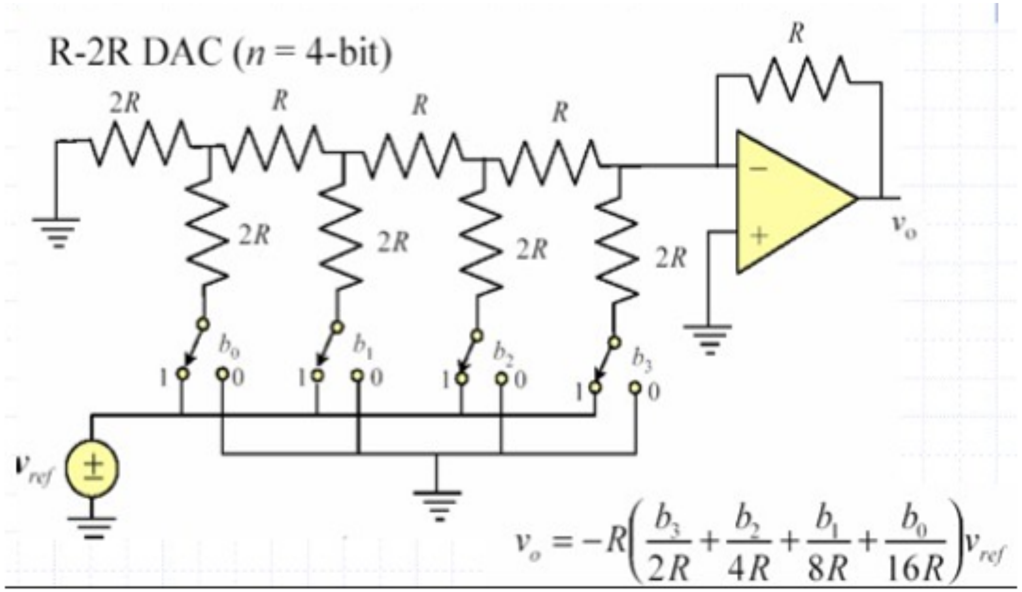
- AKA **binary-weighted input**
- Output voltage is the **inverted** sum of all input voltages
- Needs very accurate resistor values, otherwise, the input voltage would have different degrees of effect on the output and thus would not be a true sum



$$v_o = -R\left(\frac{b_3}{2R} + \frac{b_2}{4R} + \frac{b_1}{8R} + \frac{b_0}{16R}\right)v_{ref}$$

R-2R Ladder DAC

- **Required additional resistors**
- **More efficient in terms of design**


R-2R DAC (n = 4-bit)
2R
R
R
R
R
2R
2R
2R
2R
b0
b1
b2
b3
1
0
vref
vo
$v_o = -R\left(\frac{b_3}{2R}+\frac{b_2}{4R}+\frac{b_1}{8R}+\frac{b_0}{16R}\right)v_{ref}$

Commercial DACs

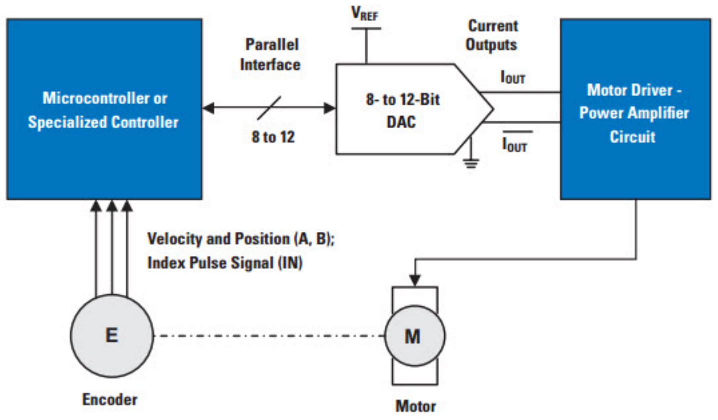
- Either voltage or current DACs
- Precision up to 16 bits
- Key timing parameter is *settling time* – amount of time it takes to produce a **stable output voltage** once the input code has changed
- We will use an 8-bit voltage DAC with a SPI interface from Maxim Semiconductor

DAC applications

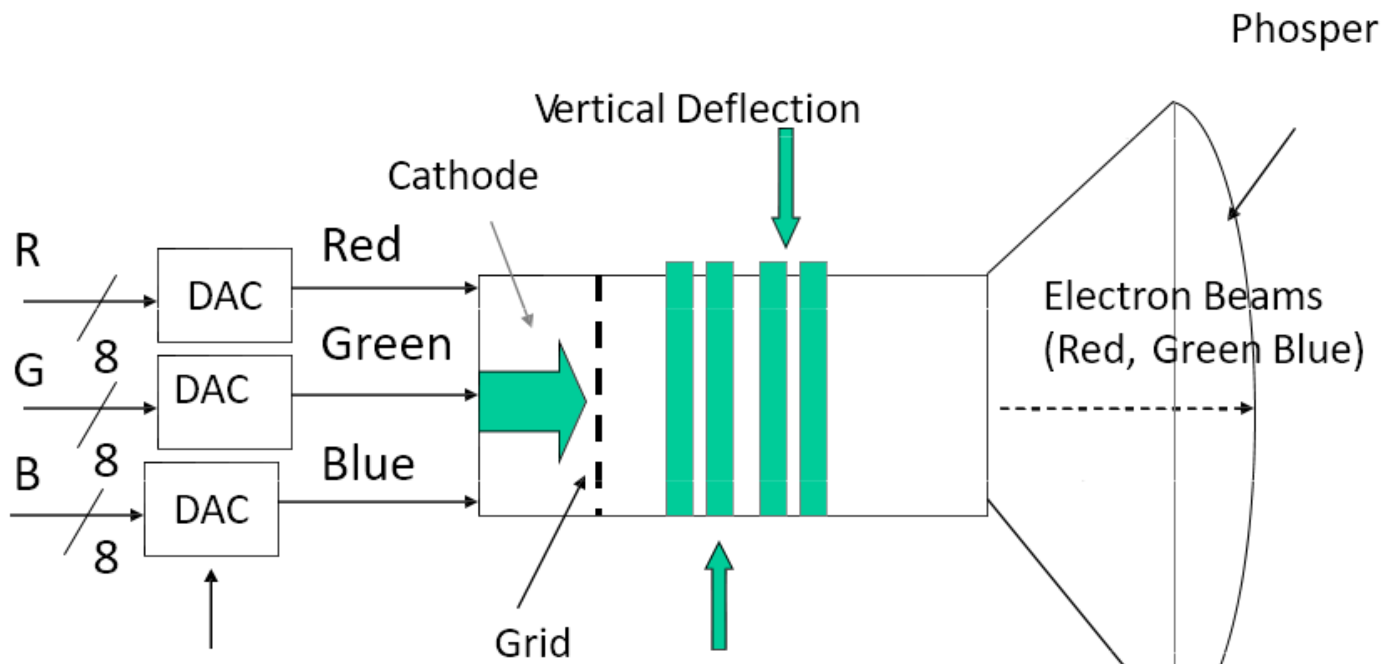
- **Audio Amplifier**
  - DACs are used to produce DC voltage gain with Microcontroller commands. Often, the DAC will be incorporated into an entire audio codec which includes signal processing features.
- **Video Encoder**
  - The video encoder system will process a video signal and send digital signals to a variety of DACs to produce analog video signals of various formats, along with optimizing of output levels. As with audio codecs, these ICs may have integrated DACs.
- **Display Electronics**
  - The graphic controller will typically use a lookup table to generate data signals sent to a video DAC for analog outputs such as Red, Green, Blue (RGB) signals to drive a display.

DAC applications

- **Data Acquisition Systems**
  - Data to be measured is digitized by an Analog-to-Digital Converter (ADC) and then sent to a processor. The data acquisition will also include a process control end, in which the processor sends feedback data to a DAC for converting to analog signals.
- **Motor Control**
  - Many motor controls require **voltage control signals**, and a DAC is ideal for this application which may be driven by a processor or controller.


Microcontroller or Specialized Controller
Parallel Interface
8 to 12
V_{REF}
8- to 12-Bit DAC
Current Outputs
I_{OUT}
$\overline{I_{OUT}}$
Motor Driver - Power Amplifier Circuit
Velocity and Position (A, B); Index Pulse Signal (IN)
E
Encoder
M
Motor

DAC Application



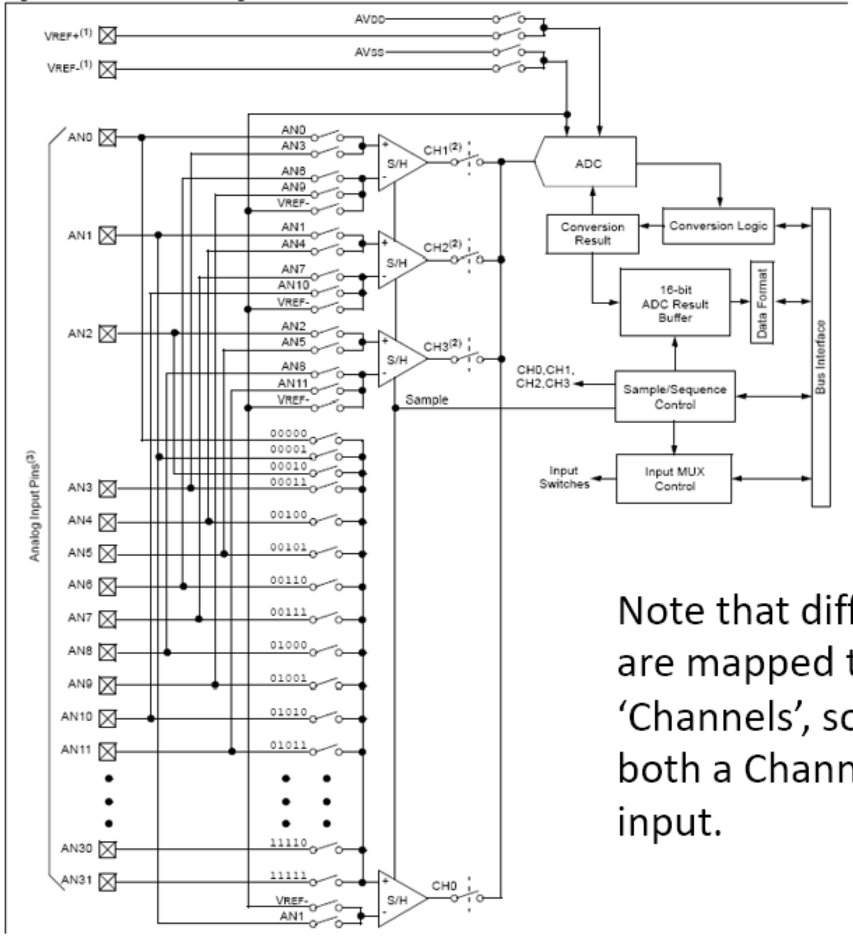
High speed video DACs produce RGB signals for color CRT

PIC24 ADC

- **The PIC24 microcontroller has an onboard ADC**
  - Successive Approximation
  - 10-bit (default) or 12-bit resolution
  - Reference voltage can be Vdd or separate voltage (min AVSS + 2.7V)
  - Multiple input (more than one input channel)
  - Clock source for ADC is either a divided Fosc, or an internally generated clock. The ADC clock period (Tad) cannot be less than 76 ns for 10-bit mode, or 118 ns for 12-bit mode. The internally generated clock has period of ~ 250ns (~ 4MHz)

Block Diagram

Figure 16-1: ADC Block Diagram

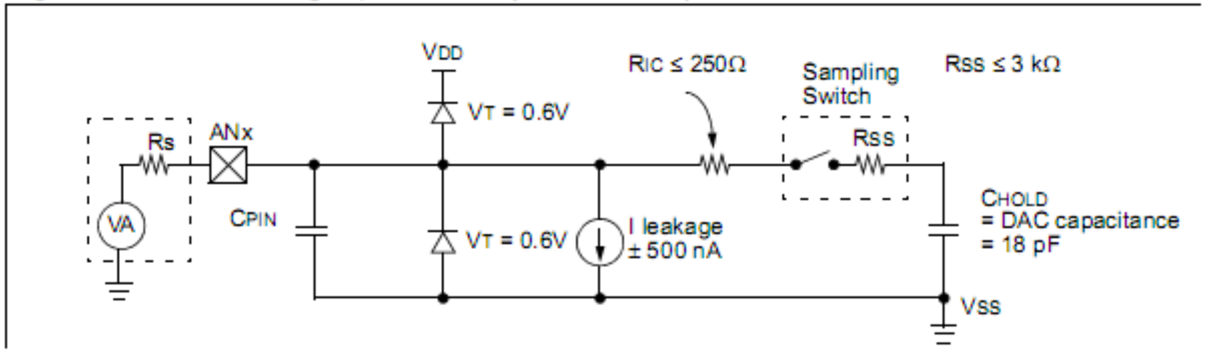


Note that different ANx inputs are mapped to different ‘Channels’, so have to select both a Channel and an ANx input.

Conversion Time

- Total conversion time = sampling time + conversion time
- Sampling looks at the input voltage and uses a storage capacitor to acquire the input.
  - This time is configurable; we will use a conservative 31 Tad periods which is the maximum for the PIC24HJGP202.
- Conversion time is Number of bits + 31 Tad periods
- So, for these settings, takes 31 (sampling) + 12 (bits) + 2 = 45 clock periods.
- Using the internal clock (250 ns), one conversion takes about 11.25 μs (88.9 kHz)

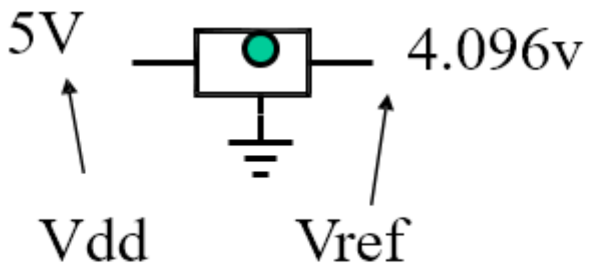
Figure 28-17: Analog Input Model (12-bit Mode)



Ric = 250 Ω
Rs = 200 Ω
Rss = 3 kΩ
Chold = 18 pF
RC = 3.45 kΩ*18 pF=61.2 n
½ LSB error = .5/4096 = 0.0122 m = 9.01 RC
9.01 RC * 61.2 n = 0.56 μs
0.56 μs/250 ns = 2.2 Tad, so round up to 3 Tad.

Voltage References

- Stability of voltage reference is critical for high precision conversions.
- We will use Vdd as our voltage reference for convenience, but we'll be throwing away at least two bits of precision due to Vdd fluctuations.
- Example commercial voltage reference: 2.048V, 2.5V, 3V, 3.3V, 4.096V, 5V (Maxim 6029). **The PIC24H can only use a voltage reference of either 3.0V or 3.3V.**



Key parameter for a voltage is stability over temperature operating range. **Need this to be less than ½ of a LSB value**.

Configuring the ADC

```
void configADC1_ManualCH0(uint16 u16_Ch0PositiveMask,
                          uint8 u8_autoSampleTime, uint8 u8_Use12bits) {

    if (u8_autoSampleTime > 31) u8_autoSampleTime = 31;
    AD1CON1bits.ADON = 0; // turn off ADC (changing setting while ADON is not allowed)

    /** Configure the internal ADC **/
    AD1CON1 = ADC_CLK_AUTO + ADC_AUTO_SAMPLING_OFF;
    if (u8_Use12bits) AD1CON1 |= ADC_12BIT;
    AD1CON3 = ADC_CONV_CLK_INTERNAL_RC + (u8_autoSampleTime << 8);
    AD1CON2 = ADC_VREF_AVDD_AVSS;
    AD1CHS0 = ADC_CH0_NEG_SAMPLEA_VREFN + u16_Ch0PositiveMask;
    AD1CON1bits.ADON = 1; //turn on the ADC

}
```

Configures for internal ADC clock, uses manual sample start/auto conversion, and u16_Ch0PositiveMask selects the ANx input from Channel 0 to convert. Uses AVDD, AVSS as references. Parameter u8_autoSampleTime sets the number of sample clocks, and u8_Use12bits determines if 12-bit or 10-bit conversion is done.

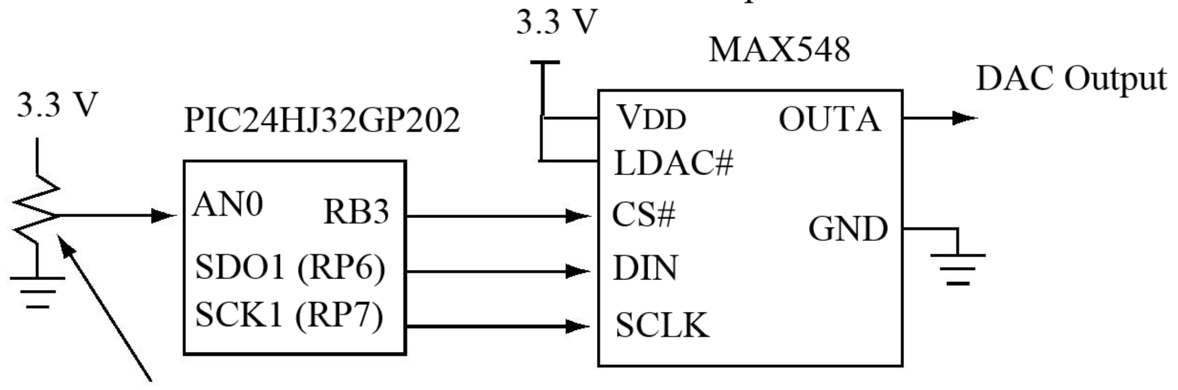
Starting a Conversion, Getting result:

```
int16 convertADC1(void) {
   SET_SAMP_BIT_AD1(); //start sampling
   WAIT_UNTIL_CONVERSION_COMPLETE_AD1(); //wait for conversion to finish
   return(ADC1BUF0);
}
```

In this mode, tell ADC to start sampling, after sampling is done the ADC conversion is started, and then a status bit is set when the conversion is finished.

Testing the ADC and DAC

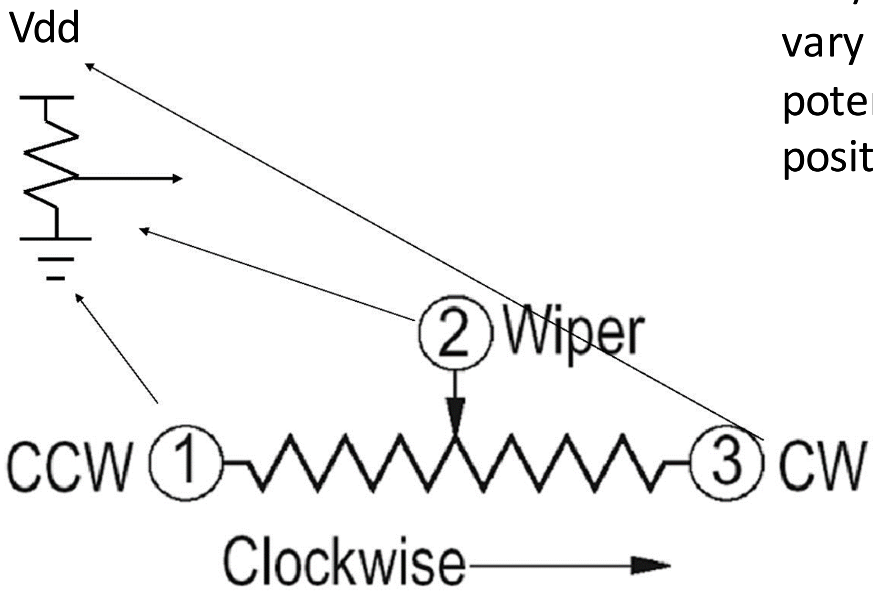
The MAX548 is an 8-bit DAC with a SPI port



Potentiometer has three pins - middle pin is the wiper, connect the end pins to Vdd/Gnd (ordering does not matter).

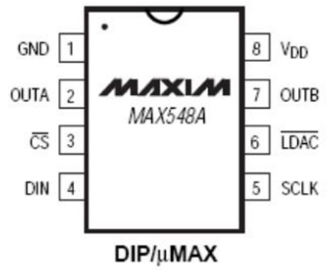
Read the voltage from the potentiometer via the PIC24 ADC, write this digital value to the DAC. The DAC output voltage should match the potentiometer voltage.

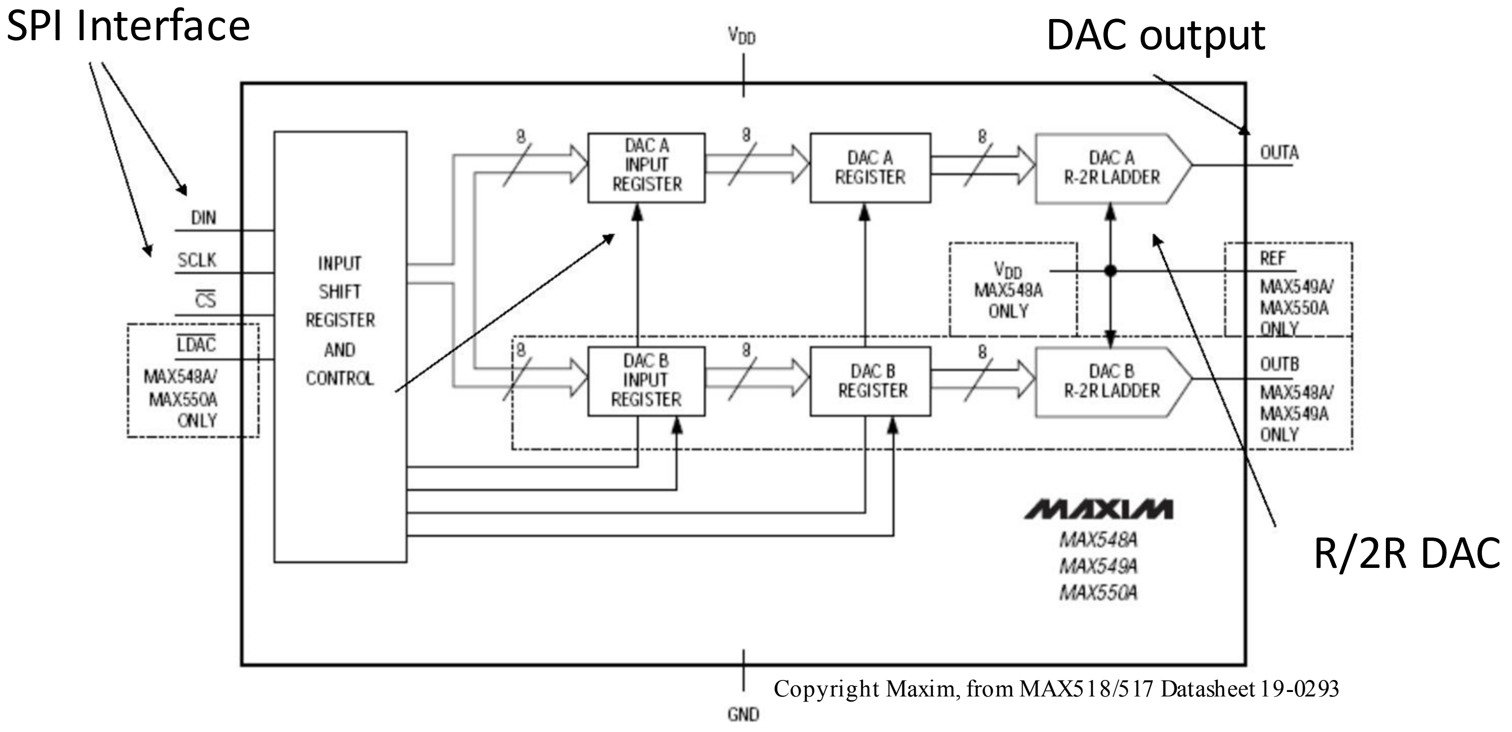
Potentiometer



A variable resistor. Tie outer two legs to Vdd/GND. Voltage on middle leg will vary between Vdd/GND as potentiometer is adjusted, changing the position of the wiper on the resistor.

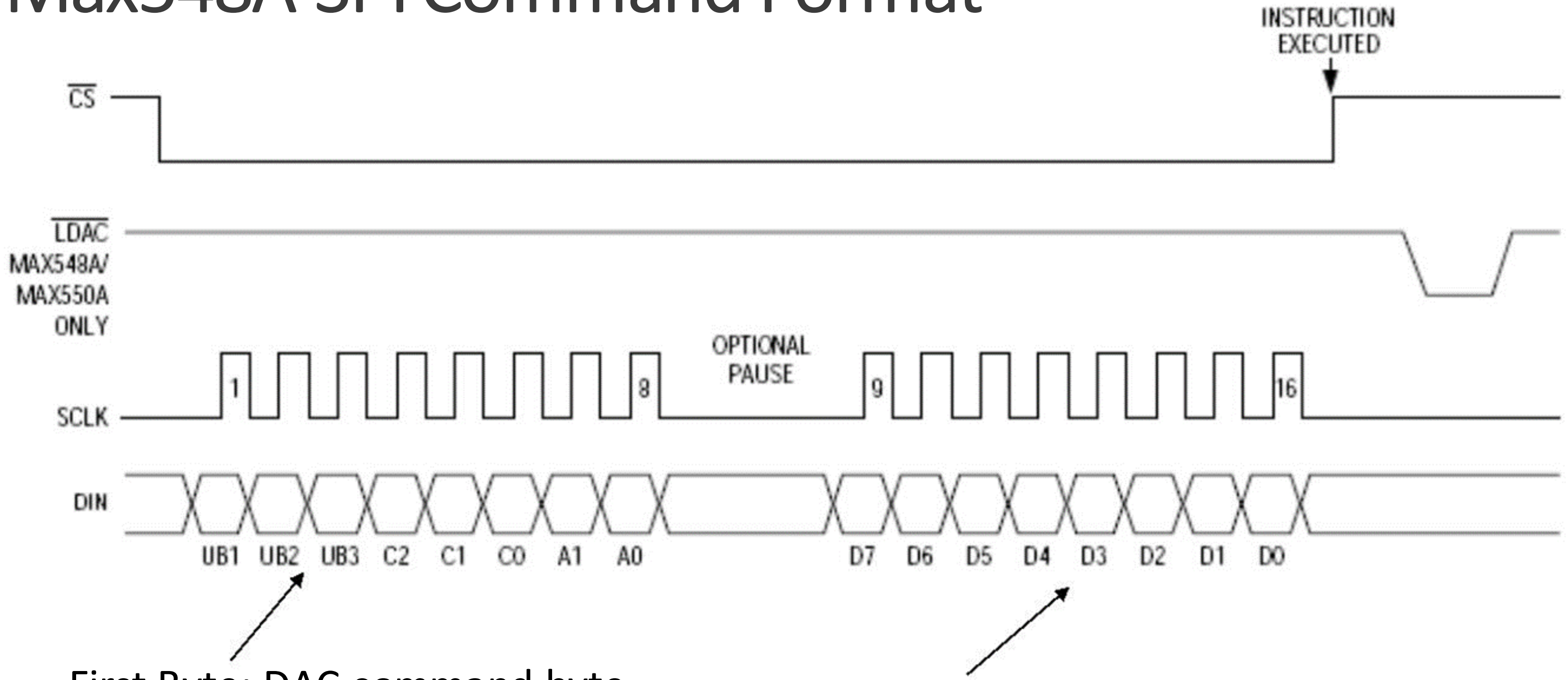
MAXIM 548 DAC





Copyright Maxim, from MAX518/517 Datasheet 19-0293

Max548A SPI Command Format



First Byte: DAC command byte

Second Byte: Command data

Command byte to do conversion: 0x09 (0b00001001)
Data is the value to convert.

Function for doing a DAC conversion:

```
#define CONFIG_SLAVE_ENABLE() CONFIG_RB8_AS_DIG_OUTPUT()
#define SLAVE_ENABLE()  _LATB8 = 0 //low true assertion
#define SLAVE_DISABLE() _LATB8 = 1

void writeDAC (uint8 dacval) {
   SLAVE_ENABLE();                         //assert Chipselect line to DAC
   ioMasterSPI1(0b00001001)     //control byte that enables DAC A
   ioMasterSPI1(dacval);        //write DAC value
   SLAVE_DISABLE();
}
```

adc_spidac_test.c

```
void configDAC() {
   CONFIG_SLAVE_ENABLE(); //chip select for DAC
   SLAVE_DISABLE();       //disable the chip select
}

int main(void) {
   uint16 u16_adcVal;
   uint8 u8_dacVal;
   uint8 u8_dacVal;
   float f_adcVal;
   float f_dacVal;

   configBasic(HELLO MSG);
   CONFIG_AN0_AS_ANALOG();
   configADC1_ManualCH0(ADC_CH0_POS_SAMPLEA_AN0, 31, 1);
   configSPI1();
   configDAC();
   while (1) {
```

Use input AN0 on Channel 0 as ADC input

Number of sampling periods, 31 is maximum for PIC24HJ32GP202

Support function, configures for manual sampling, auto conversion

Value of '1' selects 12-bit mode, '0' selects 10-bit mode.

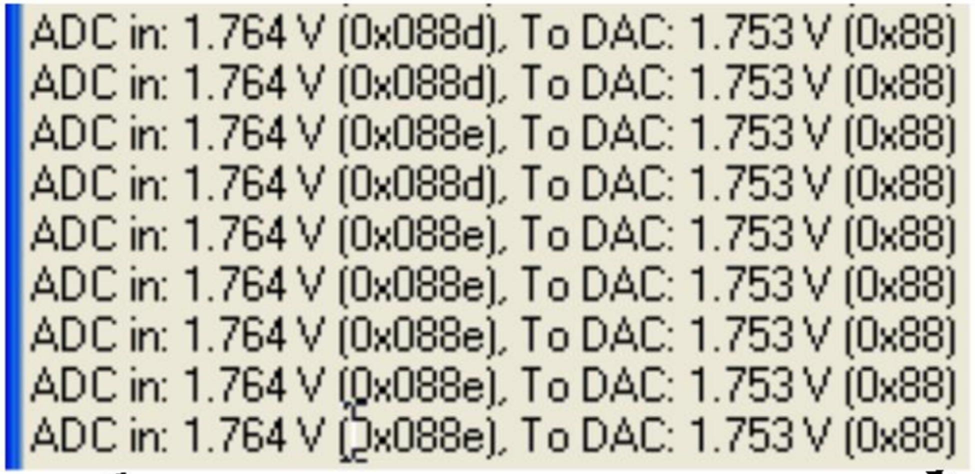
adc_spidac_test.c (cont)

```
   while (1) {
      u16_adcVal = convertADC1(); //get ADC value
      u8_dacVal = (u16_adcVal>>4) & 0x00FF; //upper 8 bits to DAC value
      writeDAC(u8_dacVal);
      f_adcVal = u16_adcVal;
      f_adcVal = f_adcVal/4096.0 * VREF; //convert to float 0.0 to VREF
      f_dacVal = u8_dacVal;
      f_dacVal = f_dacVal/256.0 * VREF;
      printf("ADC in: %4.3f V (0x%04x), To DAC: %4.3f V (0x%02x) \n",
      (double) f_adcVal, u16_adcVal, (double) f_dacVal, u8_dacVal);
      DELAY_MS(300); //delay so that we do not flood the UART.
   } //end while(1)
}
```

u16_adcVal is 12-bit ADC value.
f_adcVal is **u16_adcVal** converted to a voltage between 0 - 3.3V using a float data type.

u8_dacVal is the 8-bit value to send to the DAC (upper 8 bits of **u16_adcVal**).
f_dacVal is **u8_dacVal** converted to a voltage between 0 – 3.3V using a float data type.

Program Output



```
ADC in: 1.764 V (0x088d), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088d), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088d), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)
```

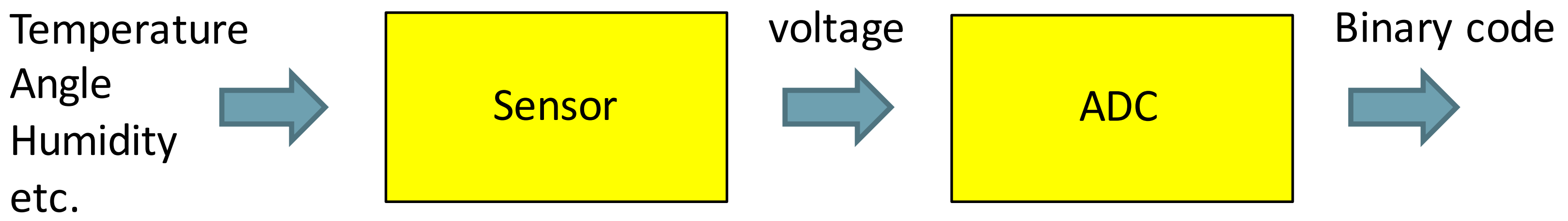
12-bit ADC code as voltage

12-bit ADC code

DAC code as 8-bit voltage

8-bit DAC code

Sensors



y = f(x)
Voltage = F (*real_world_quantity*)

- The sensor has some function that maps the real world quantity into a voltage; the function can be either linear or non-linear.
- A linear function is characterized by:

  y = m * x + b

  voltage = m * *real_world_quantity* + offset

Example 1

- An LM60 temperature sensor produces 6.25 mV for every 1°C and has a DC offset of 424 mV.
  - Voltage (mV) = 6.25 mV * Tcelsius + 424 mV
- Question: Using a 10-bit ADC with Vref = 3.3V, what is the ADC code value for a temperature of -10 Celsius?
  - Step 1: Convert temperature to voltage:
    - Voltage (mV) = 6.25 mV * (-10) + 424 mV = 361.5 mV = 0.3615 V
  - Step 2: Convert voltage to ADC code:
    - Vin/Vref * 2 = 0.3615/3.3 * 2 = 112.17 = 112

Example 1 (cont)

- Can reduce the computations needed by simplifying the equation by combining steps:
- Question: Using a 10-bit ADC with Vref = 3.3V, what is the ADC code value for a temperature of -10 Celsius?
- Step 1: Convert temperature to voltage:
  - Voltage (mV) = 6.25 mV * (Tcelsius) + 424 mV
  - ADC code = (0.00625 * Tcelsius) + 0.424)/ 3.3 V * 1024
  - ADC code = (6.4 * Tcelsius) + 434.176)/ 3.3 V
- For Tcelsius = -10
  - ADC code = ((6.4 * -10) + 434.176)/3.3 = 112.17 = 112

Example 2

- A Freescale MPX5050 pressure sensor outputs 0.2 V at 0 kPa and has a sensitivity of 90 mV/kPa.
  - Voltage (mV) = 90 mV * Pressure_kPa + 200 mV
- Question: A 10-bit ADC with a Vref = 3.3 V returns a code value of 420. What pressure is being sensed by the pressure sensor?
  - Step 1: Convert ADC code to a voltage:
    - adc_code/2^N * Vref = 420/ 2^{10} * 3.3 V = 1.354 V.
  - Step 2: Convert Volts to pressure (solve the above equation for pressure):
    - Pressure (kPa) = (voltage (mV) – 200 mV) / 90 mV

      = (1354 mV – 200 mV)/90 mV

      = 12.8 kPa

What do you have to know?

- Vocabulary
- DAC R/2R architecture
- ADC Flash, Successive approximation architectures
- PIC24 ADC
  - How to configure
  - Acquisition, Conversion time
  - How to start do conversion, read results
- MAX548A DAC usage