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 – for ADC, time between each conversion
  – Typically, samples are taken at a fixed rate
• Vref (Reference Voltage) – analog signal varies between 0 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
An N-bit ADC

Maps a voltage ($V_{in}$) to a digital code $ADC\_code$

$$ADC\_code = \lfloor (V_{in}/V_{ref}) \times 2^N \rfloor$$

$V_{in}$ is always considered less than $V_{ref}$, so $V_{in}/V_{ref}$ is always $< 1.0$. Any fractional part of the code is truncated.
Example: A 10-bit ADC

Vref = 4 V

Maps a voltage ($V_{in}$) to a digital code $ADC\_code$

$V_{in} = 3.0 \text{ V}$

$ADC\_code = 768$

$ADC\_code = \frac{V_{in}}{V_{ref}} \times 2^N$

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

$= 0.75 \times 1024 = 768$
Example: A 10-bit ADC

Vref = 4 V

Maps a voltage (Vin) to a digital code $ADC_{\text{code}}$

$Vin = 2.17 \text{ V}$

$ADC_{\text{code}} = (Vin/Vref) \times 2^N$

$= (2.17 \text{ V}/4 \text{ V}) \times 1024$

$= 555.52 \approx 555$

$ADC_{\text{code}} = 555$
Going from Code to Voltage

\[ \text{ADC\_code} = (\text{Vin}/\text{Vref}) \times 2^N \]

\[ \frac{\text{ADC\_code}}{2^N} \times \text{Vref} = \text{Vin} \]

\[ \text{Vin} = 3.0 \text{ V} \]

\[ \frac{\text{Vin}}{\text{ADC\_code}/2^N} \times \text{Vref} = \text{Vin} \]

\[ = \frac{768}{1024} \times 4 \text{ V} \]

\[ = 0.75 \times 4 \text{ V} \]

\[ = 3 \text{ V} \]
Going from Code to Voltage

\[ ADC\_code = (Vin/Vref) \times 2^N \]
\[ ADC\_code/2^N \times Vref = Vin \]

\[ Vin = 2.168 \, V \]
\[ Vin = ADC\_code/2^N \times Vref \]
\[ = 555/1024 \times 4 \, V \]
\[ = 2.167968 \]
\[ = \sim 2.168 \]
ADC Resolution

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

$$\frac{1}{2^N} \times (V_{\text{ref}^+} - V_{\text{ref}^-})$$

Where $V_{\text{ref}^+}$ is the positive reference voltage and $V_{\text{ref}^-}$ is the negative reference voltage.

We will use $V_{\text{ref}^-} = 0$ V, and refer to $V_{\text{ref}^+}$ as simply $V_{\text{ref}}$, so this simplifies to

$$\frac{1}{2^N} \times V_{\text{ref}}.$$

For $V_{\text{ref}} = 4$ V, and $N = 4$, what is 1 LSb?

$$\frac{1}{2^4} \times 4 \text{ V} = \frac{1}{16} \times 4 \text{ V} = 0.25 \text{ V}.$$
Example: 10-bit ADC Resolution

V_ref = 4.0 V

Vin = 3.00390625 V
Vin = 3.0 V

1 LSB voltage = \( \frac{1}{2^N} \times V_{ref} \)

= \( \frac{1}{1024} \times 4 \text{ V} \)

= 0.00390625 V

= \sim 3.9 \text{ mV}

ADC_code = 768
ADC_code = 769
ADC, DAC Equations

ADC: \( \text{Vin} = \text{input voltage}, \ Vref^+ = \text{reference voltage}, \ Vref^- = 0 \text{ V}. \) 
\( N = \text{number of bits of precision} \)

\[ \frac{\text{Vin}}{\text{Vref}} \times 2^N = \text{output code} \]
\[ \frac{\text{output code}}{2^N} \times \text{Vref} = \text{Vin} \]

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

DAC: \( \text{Vout} = \text{output voltage}, \ Vref = \text{reference voltage}, \)
\( N = \text{number of bits of precision} \)

\[ \frac{\text{Vout}}{\text{Vref}} \times 2^N = \text{input code} \]
\[ \frac{\text{input code}}{2^N} \times \text{Vref} = \text{Vout} \]

1 LSB = \( \frac{\text{Vref}}{2^N} \)
Sample ADC, DAC Computations

If $V_{\text{ref}} = 5\text{V}$, and a 10-bit A/D output code is $0x12A$, what is the ADC input voltage?

\[
V_{\text{in}} = \frac{\text{output}_\text{code}}{2^N} \times V_{\text{ref}} = \frac{0x12A}{2^{10}} \times 5\text{V}
\]
\[
= \frac{298}{1024} \times 5\text{V} = 1.46\text{V} \quad \text{(ADC Vin)}
\]

If $V_{\text{ref}} = 5\text{V}$, and an 8-bit DAC input code is $0xA9$, what is the DAC output voltage?

\[
V_{\text{out}} = \frac{\text{input}_\text{code}}{2^N} \times V_{\text{ref}} = \frac{0xA9}{2^8} \times 5\text{V}
\]
\[
= \frac{169}{256} \times 5\text{V} = 3.3\text{V} \quad \text{(DAC Vout)}
\]

If $V_{\text{ref}} = 4\text{V}$, and an 8-bit A/D input voltage is $2.35\text{V}$, what is the ADC output code?

\[
\text{output}_\text{code} = \frac{V_{\text{in}}}{V_{\text{ref}}} \times 2^N = \frac{2.35\text{V}}{4\text{V}} \times 2^8
\]
\[
= .5875 \times 256 = 150.4 = 150 = 0x96 \quad \text{(ADC output code)}
\]
Digital-to-Analog Conversion

For a particular binary code, output a voltage between 0 and $V_{ref}$

Assume a DAC that uses an unsigned binary input code, with $0 \leq V_{out} < V_{ref}$. Then

$D = 0000\ 0000 \quad V_{out} = 0V$
$D = 0000\ 0001 \quad V_{out} = V_{ref}(1/256) \quad \text{(one LSB)}$
$D = 0000\ 0010 \quad V_{out} = V_{ref}(2/256)$
...
$D = 1111\ 1111 \quad V_{out} = V_{ref}(255/256) \quad \text{(full scale)}$
DAC Output Plot

Output signal increases in 1 LSB increments.

Vout

4/256 Vref
3/256 Vref
2/256 Vref
1/256 Vref

0 1 2 3
Input code
An N-bit DAC

Maps a digital code ($DAC\_code$) to a voltage ($V_{out}$)

$$V_{out} = \frac{DAC\_code}{2^N} \times V_{ref}$$
A 1-bit ADC

- Analog signal: \( V_{\text{in}} \)
- Reference voltage: \( V_{\text{ref}} \)
- Comparator: \( V_{\text{out}} = V_{\text{dd}} \) if \( V_{\text{in}} > V_{\text{ref}}/2 \)
- Comparator: \( V_{\text{out}} = 0 \) if \( V_{\text{in}} < V_{\text{ref}}/2 \)
Counter Ramp ADC

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

Initially set $V_{DAC}$ to $\frac{1}{2} V_{ref}$, then see if $V_{in}$ higher or lower than $V_{DAC}$. If $> \frac{1}{2} V_{ref}$, then next guess is between $V_{ref}$ and $\frac{1}{2} V_{ref}$, else next guess is between $\frac{1}{2} V_{ref}$ and GND. Do this for each bit of the ADC. Takes $N$ clock cycles.
Successive Approximation Example

Given a 4-bit Successive Approximation ADC, and Vref = 4 V.
Let Vin = 3.14159 V. Clear DAC input to 0b0000.

1. First guess, DAC input = 0b1000 = 8, so Vdac = 8/2^4 * 4 V = 8/16 * 4 V = 2 V.
   Vdac (2 V) < Vin (3.14159 V), so guess of ‘1’ for MSb of DAC was correct.

2. Set next bit of DAC to ‘1’, DAC input = 0b1100 = 12, so Vdac = 12/16*4= 3V.
   Vdac (3 V) < Vin (3.14159 V), so guess of ‘1’ for bit2 of DAC was correct.

3. Set next bit of DAC to ‘1’, DAC input = 0b1110 = 14, so Vdac = 14/16*4= 3.5V.
   Vdac (3.5 V) > Vin (3.14159 V), so guess of ‘0’ for bit1 of DAC was incorrect.
   Reset this bit to ‘0’.

4. Set last bit of DAC to ‘1’, DAC input = 0b1101 = 13, so Vdac = 13/16*4 = 3.25V.
   Vdac (3.25 V) > Vin (3.14159 V), so guess of ‘0’ for bit0 of DAC was incorrect.
   Reset this bit to ‘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

- **Vin**
  - +
  - -
- 3/4Vref
- **Vin**
  - +
  - -
- 1/2Vref
- **Vin**
  - +
  - -
- 1/4Vref

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(Other codes don’t cares)

Encoding logic

Fast, conversion time is settling time of comparators and digital logic.
3-bit Flash ADC

\[ V_7 = 7/8 \times V_{REF+} \]
\[ V_6 = 6/8 \times V_{REF+} \]
\[ V_5 = 5/8 \times V_{REF+} \]
\[ V_4 = 4/8 \times V_{REF+} \]
\[ V_3 = 3/8 \times V_{REF+} \]
\[ V_2 = 2/8 \times V_{REF+} \]
\[ V_1 = 1/8 \times V_{REF+} \]
ADC Architecture Summary

• **Flash** ADCs
  – Fastest possible conversion time
  – Requires the most transistors of any architecture
  – N-bit converter requires $2^N-1$ comparators.
  – Commercially available flash converters up to 12 bits.
  – Conversion done in one clock cycle

• **Successive approximation** ADCs
  – 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
    \[ = \frac{4.1}{2^{16}} = 62 \mu V. \]
Flash DAC

Eliminates large capacitive load at one node.

Large capacitive load

(a) With Switch Decoder

(b) Without Switch Decoder

N-bit DAC requires $2^N$ resistors!
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

If $V_{\text{ref}} = 5\, \text{V}$, and the 8-bit input code is $0x8A$, what is the DAC output voltage?

\[
\frac{\text{input\_code}}{2^N} \times V_{\text{ref}} = \frac{0x8A}{2^8} \times 5 \, \text{V} \\
= \frac{138}{256} \times 5 \, \text{V} = 2.70 \, \text{V} \, \text{(Vout)}
\]

If $V_{\text{ref}} = 4\, \text{V}$, and the DAC output voltage is $1.25 \, \text{V}$, what is the 8-bit input code?

\[
\frac{\text{Vout}}{V_{\text{ref}}} \times 2^N = \frac{1.25}{4} \, \text{V} \times 2^8 \\
= 0.3125 \times 256 = 80 = 0x50 \, \text{(input\_code)}
\]
Commercial DACs

• Either voltage or current DACs
  – Current DACs require an external operational amplifier to convert to voltage
• 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 Application

High speed video DACs produce RGB signals for color CRT
The PIC24 µC has an onboard ADC
- Successive approximation
- 10-bit (default) or 12-bit resolution
- Reference voltage can be Vdd or separate voltage (min AVSS + 2.7 V)
- 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 a period of ~ 250 ns (~ 4 MHz).
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 is 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).

\[
\text{Ric} = 250 \ \Omega \\
\text{Rs} = 200 \ \Omega \\
\text{Rss} = 3 \ \text{k}\Omega \\
\text{Chold} = 18 \ \text{pF} \\
\text{RC} = 3.45 \ \text{k}\Omega \times 18 \ \text{pF} = 61.2 \ \text{n} \\
\frac{1}{2} \text{ LSB error} = \frac{.5}{4096} = 0.0122 \ \text{m} = 9.01 \ \text{RC} \\
9.01 \ \text{RC} \times 61.2 \ \text{n} = 0.56 \ \mu\text{s} \\
0.56 \ \mu\text{s} / 250 \ \text{ns} = 2.2 \ \text{Tad}, \text{ so round up to 3 Tad.}
\]
Voltage References

Stability of voltage reference is critical for high precision conversions.

We will use Vdd as our voltage reference for convenience, but will 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 3.0 V or 3.3 V.

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

```c
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:

```c
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

SPI interface

DAC output

R/2R DAC
Max548A SPI Command Format

First Byte: DAC command byte
Command byte to do conversion: 0x09  (0b00001001)
Data is the value to convert.
Function for doing a DAC conversion:

```c
#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();
}
```

V 0.7 39
void configDAC() {
    CONFIG_SLAVE_ENABLE();       //chip select for DAC
    SLAVE_DISABLE();             //disable the chip select
}

int main(void) {
    uint16 u16_adcVal;
    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) {
        Support function,
        configures for manual
        sampling, auto conversion

    }

    Use input AN0 on Channel 0
    as ADC input

    Number of sampling periods,
    31 is maximum for
    PIC24HJ32GP202

    Value of ‘1’ selects 12-bit
    mode, ‘0’ selects 10-bit
    mode.
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)
ADC in: 1.764 V (0x088e), To DAC: 1.753 V (0x88)

12-bit ADC code
8-bit DAC code

12-bit ADC code as voltage
8-bit DAC code as voltage
Sensors

Temperature, Angle, Humidity, Etc.

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:

\[ y = m \cdot x + b \]

\[ \text{voltage} = m \cdot \text{real}_\text{world}_\text{quantity} + \text{offset} \]
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^N = 0.3615/3.3 * 2^10 = 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:

\[
\text{adc\_code/2^N \times V_{ref}} = \frac{420}{2^{10}} \times 3.3 \text{ V} = 1.354 \text{ V}
\]

Step 2: Convert Volts to pressure (solve the above equation for pressure):

Pressure (kPa) = \left(\frac{\text{voltage (mV)} - 200 \text{ mV}}{90 \text{ mV}}\right)
\[
= \left(\frac{1354 \text{ mV} - 200 \text{ mV}}{90 \text{ mV}}\right)
\]
\[
= 12.8 \text{ kPa}
\]
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 result
• MAX548A DAC usage