DAC/ADC

E155

Sources

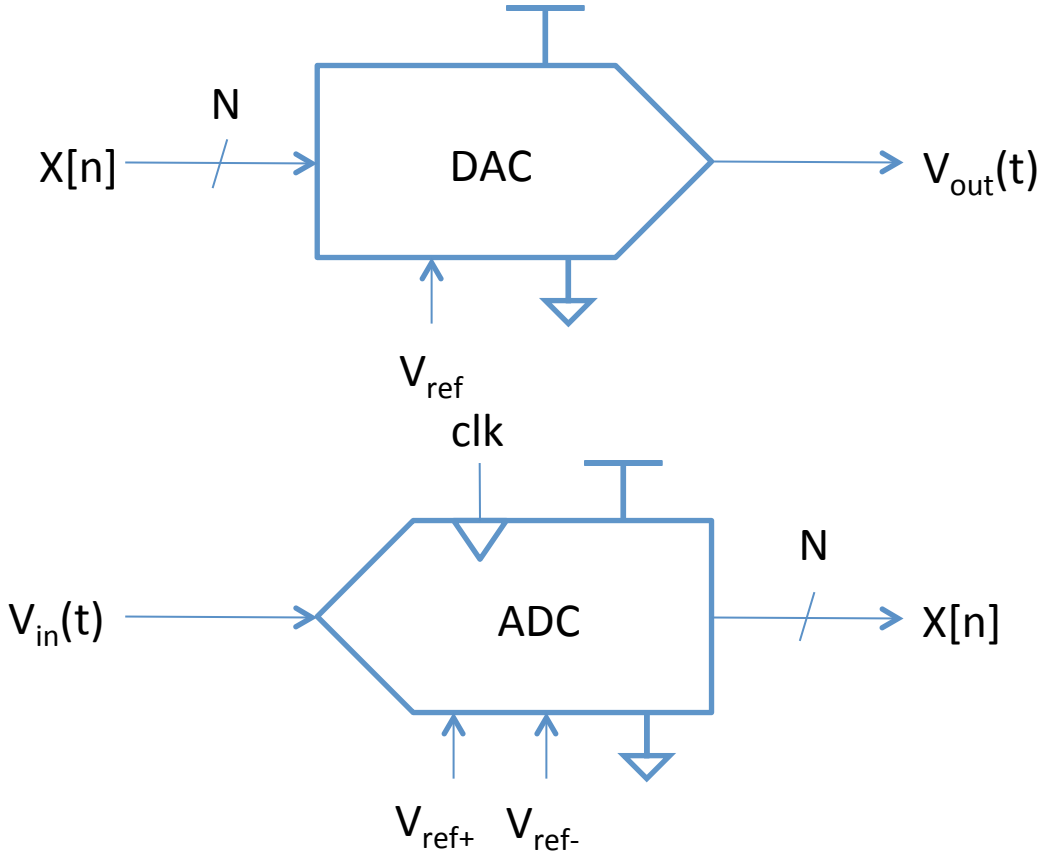
- Harris and Harris 2nd Ed. Chapter 8

Analog Input and Output

- Interface with the real world
- Analog-to-digital-converter (ADC)
- Digital-to-analog-converter (DAC)

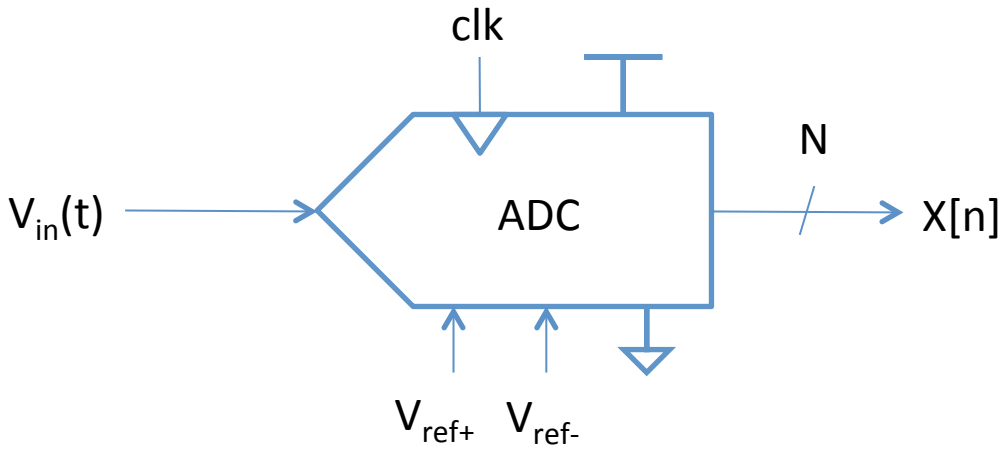
DAC/ADC Characteristics

- Resolution
- Dynamic range
- Sampling rate
- Accuracy



Example ADC

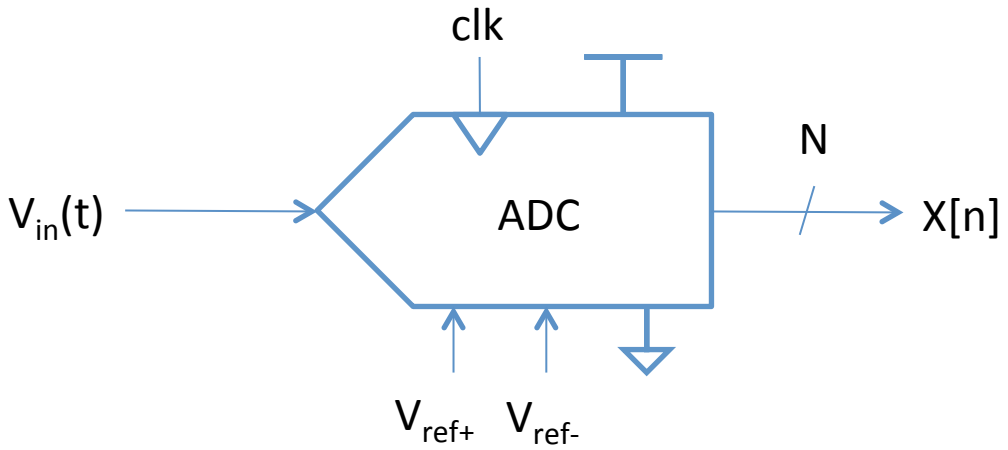
- Resolution: N = 12-bit
- Range: V_{ref-} to V_{ref+} = 0-5 V
- Sampling f_s = 44 KHz
- Accuracy: ± 3 least significant bits (lsbs)



ADC I/O Relationship

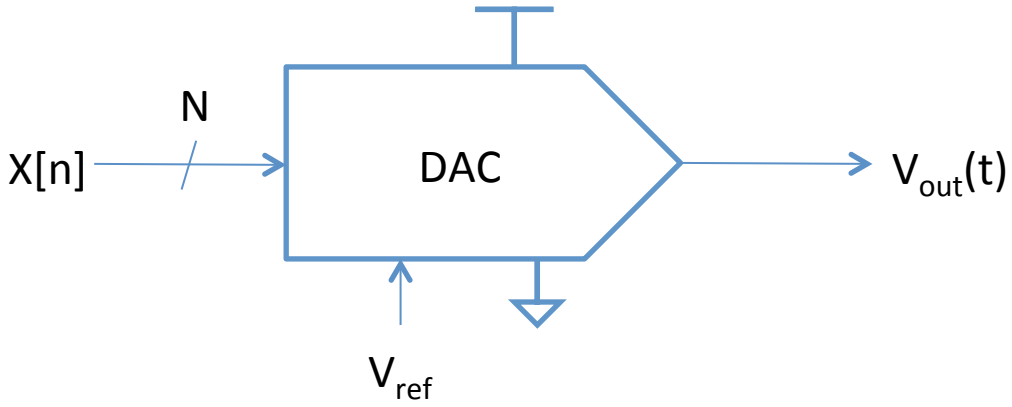
$$X[n] = 2^N \frac{V_{in}(t) - V_{ref^-}}{V_{ref^+} - V_{ref^-}}$$

$$n = \frac{t}{f_s}$$



DAC I/O Relationship

$$V_{out}(t) = \frac{X[n]}{2^N} V_{ref}$$



PIC32

- 10-bit ADC built-in
- 1 million samples/sec (Msps)
- Any of 16 analog input pins
  - AN0-15
  - Shared with digital I/O port RB
- $V_{ref+} = V_{DD}$ (3.3 V)
- V_{ref-} = GND (0 V)

Control Registers

- AD1CON1-3 – primary control
- AD1CHS – channel select
- AD1PCFG – pin configuration
- AD1CSSL – see data sheet
- ADC1BUF0-F – 10-bit conversion result

ADC Info

- ADC clock period: T_{AD} >= 65ns
- ADCS can be left at 0

$$T_{AD} = 2T_{PB}(ADCS + 1)$$

Analog Input Example

```
#include <P32xxxx.h>

void initadc(int channel) {
    AD1CHSbits.CHOSA = channel;   // select which channel
    AD1PCFGCLR = 1 << channel;    // configure input pin
    AD1CON1bits.ON = 1;           // turn ADC on
    AD1CON1bits.SAMP = 1;         // begin sampling
    AD1CON1bits.DONE = 0;         // clear DONE flag
}

int readadc(void) {
    AD1CON1bits.SAMP = 0;         // end sampling, star conversion
    while (!AD1CON1bits.DONE);    // wait until DONE
    AD1CON1bits.SAMP = 1;         // resume sampling
    AD1CON1bits.DONE = 0;         // clear DONE flag
    return ADC1BUF0;              // return result
}

int main(void) {
    int sample;
    initadc(11);
    sample = readadc();
}
```

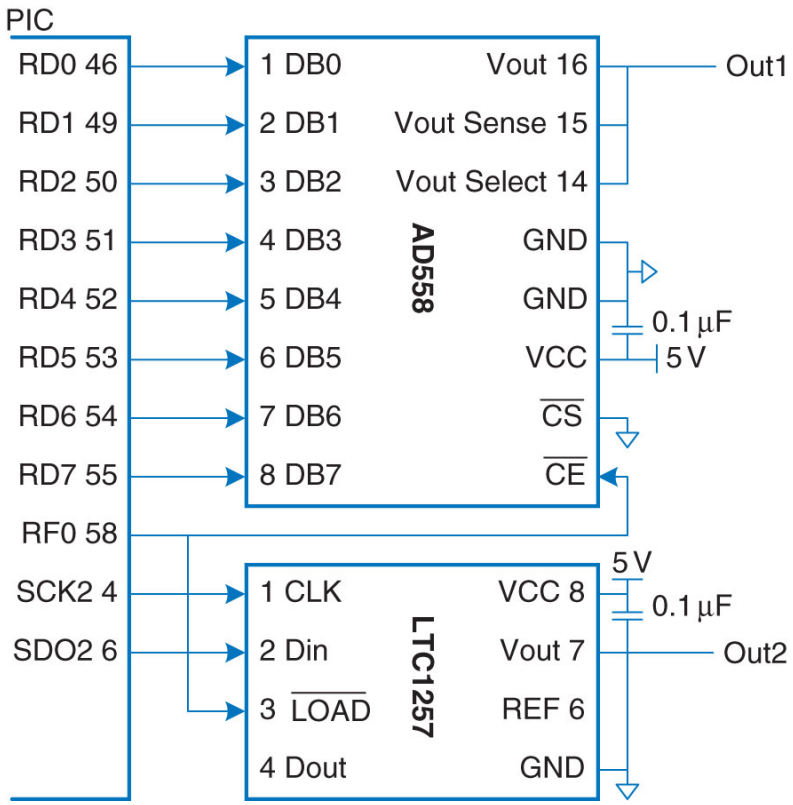
DAC Conversion

- No built-in DACs
- Some accept N parallel wires
- Some accept serial (such as SPI)
- Flexible voltage vs. not
- May need an op-amp

Examples

- AD558 8-bit parallel DAC
  - Output 0-2.56 V, 75 mW
  - 1µs settling time = 1Msample/sec
  - analog.com for datasheet
- LTC1257 12-bit serial DAC
  - Output 0-2.048 V, < 2 mW
  - 6µs settling time (SPI at 1.4 MHz)
  - linear.com for datasheet
- Texas Instruments also makes DACs/ADCs

DAC Circuit


PIC
RD0 46
RD1 49
RD2 50
RD3 51
RD4 52
RD5 53
RD6 54
RD7 55
RF0 58
SCK2 4
SDO2 6
1 DB0
2 DB1
3 DB2
4 DB3
5 DB4
6 DB5
7 DB6
8 DB7
AD558
Vout 16
Vout Sense 15
Vout Select 14
GND
GND
VCC
CS
CE
Out1
0.1 µF
5 V
1 CLK
2 Din
3 LOAD
4 Dout
LTC1257
VCC 8
Vout 7
REF 6
GND
5 V
0.1 µF
Out2

DAC sine and triangle waves

```
#include <P32xxxx.h>
#include <math.h>

#define NUMPTS 64

int sine[NUMPTS], triangle[NUMPTS];

void initio(int freq) {       // 5-605 Hz frequency
    TRISD = 0xFF00;           // PORT D outputs
    SPI2CONbits.ON = 0;       // disable to reset state
    SPI2BRG = 0;              // 1 MHz SPI clock
    SPI2CONbits.MSTEN = 1;    // enable master mode
    SPI2CONbits.CKE = 1;      // set clock-to-data timing
    SPI2CONbits.MODE16 = 1;   // activate 16-bit mode
    SPI2CONbits.ON = 1;       // turn SPI on

    TRISF = 0xFFFE;           // make RF0 an output (load and ce)
    PORTFbits.RF0 = 1;        // set RF0 = 1

    PR1 = (20e6/NUMPTS)/freq - 1; // set period register for frequency

    T1CONbits.ON = 1;         // turn Timer1 on
}
```

DAC cont.

```
void initwavetables(void) {
    int i;

    for (i=0; i<NUMPTS; i++) {
        sine[i] = 2047*(sin(2*3.14159*i/NUMPTS) + 1);  // 12-bit
        if (i<NUMPTS/2) triangle[i] = i*511/NUMPTS;    // 8-bit
        else triangle[i] = 510 - i*511/NUMPTS;
    }
}

void genwaves(void) {
    int I;
    while(1) {
        for (i=0; i<NUMPTS; i++) {
            IFS0bits.T1IF = 0;        // clear timer overflow flag
            PORTFbits.RF0 = 1;        // disable load while changing
            SPI2BUF = sine[i];        // send points to DACs
            PORTD = triangles[i];
            while(SPI2STATbits.SPIBUSY);  // wait until complete
            PORTFbits.RF0 = 0;        // load new points into DACs
            while (!IFS0bits.T1IF);   // wait until time for next
        }
    }
}
```

DAC main

```
int main(void) {
     initio(500);
     initwavetables();
     genwaves();
}
```

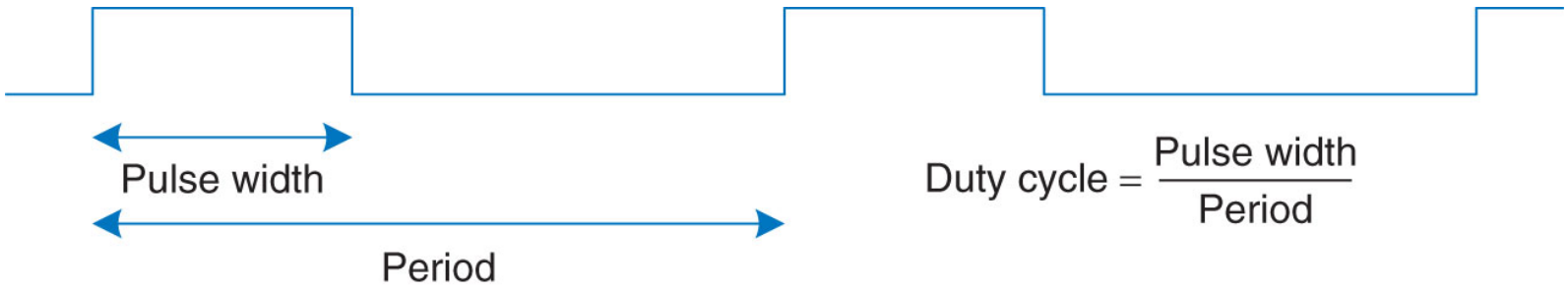
Pulse-width Modulation

- Analog through digital duty cycle
- Example:
  - Output from 0 to 3.3 V
  - 25% duty cycle

- Low-pass filter
- Use output compare (OC1-OC5)
- Use Timer 2 or 3 also

"Don't call it that"
-Michael Bluth

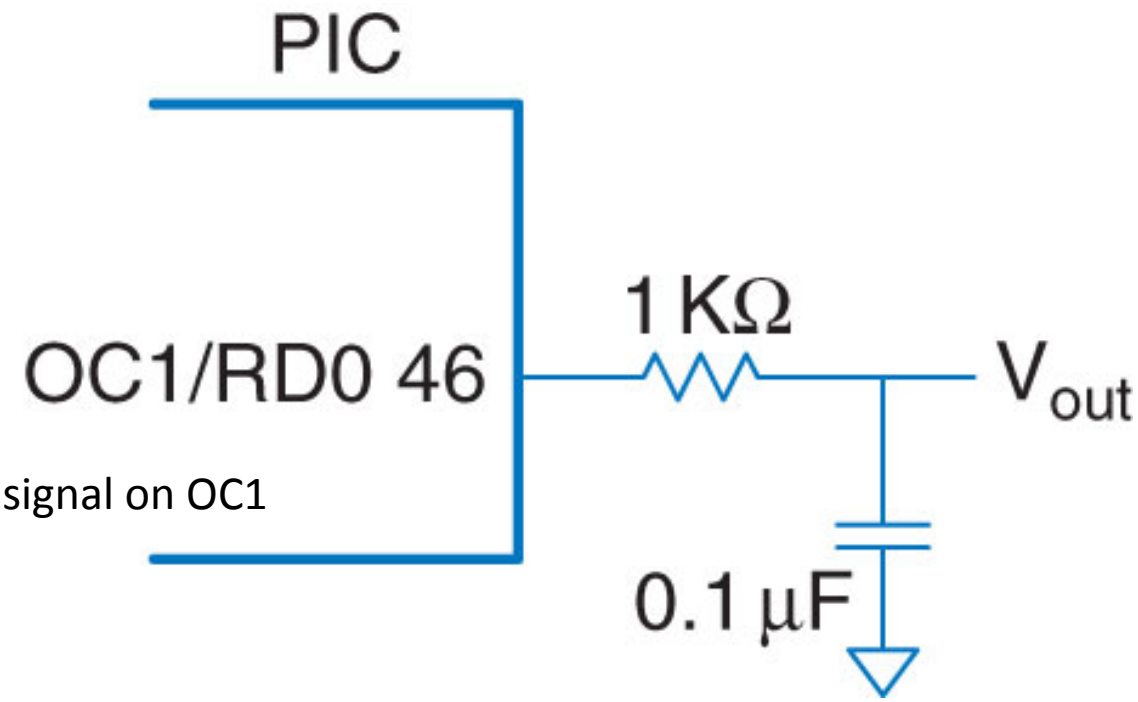
PWM



Output Compare (PIC32)

- OCxCON – control register
  - OCM bits set to 110_2 for PWM
  - ON bit enabled
  - OCTSEL (Timer2 in 16-bit by default)
- OCxR – see data sheet
- OCxRS – sets duty cycle
- Timer's period register (PR) sets period

Example



78.125 KHz signal on OC1

$$f_c = \frac{1}{2\pi RC} = 1.6KHz$$

Example Code

```
#include <P32xxx.h>

void genpwm(int dutycycle) {
    PR2 = 255;                  // set period to 255+1 ticks (78.125 KHz)
    OC1RS = dutycycle;
    OC1CONbits.OCM = 0b110;     // set OC1 to PWM mode
    T2CONbits.ON = 1;           // default mode (20 MHz, 16-bit)
    OC1CONbits.ON = 1;          // turn on OC1
}
```

Example Code Comparison

```
#include <P32xxx.h>

void genpwm(int dutycycle) {
    PR2 = 255;                  // set period to 255+1 ticks (78.125 KHz)
    OC1RS = dutycycle;
    OC1CONbits.OCM = 0b110;     // set OC1 to PWM mode
    T2CONbits.ON = 1;           // default mode (20 MHz, 16-bit)
    OC1CONbits.ON = 1;          // turn on OC1
}

// servo (for comparison)
void initservo(void) {
    T2CONbits.TCKPS = 0b111;  // prescale
    PR2 = 1561;
    OC1RS = 117;
    OC1CONbits.OCM = 0b110;
    T2CONbits.ON = 1;
    OC1CONbits.ON = 1;
}
void setservo(int angle) {
    if (angle < 0) angle =0; else if (angle > 180) angle = 180;
    OC1RS = 39 + angle * 156.1/180;
}
```

Speaker Audio

- Speakers take analog signals
- CD Audio is 44.1 kHz samples
- Can use PWM to convert
- May want an amplifier
- DAC, Filter and C design left to you!

Microphone Recording

- Analog signal
- Sample with ADC
- Design up to you!