Simple Active Filter

(See Figures 11.52, 11.53, and 11.54)

Two-Pole Butterworth Low-Pass Analog Filter

(See Figure 11.55)

1. Select the cutoff frequency $f_c$.
2. Divide the two capacitors by $2\pi f_c$.
   $$C_{1A} = \frac{14.7}{2\pi f_c} \quad C_{2A} = \frac{70.7}{2\pi f_c}$$
3. Select standard capacitors with same order of magnitude.
   $$C_{1B} = \frac{C_{1A}}{2} \quad C_{2B} = \frac{C_{2A}}{2}$$
4. Adjust resistors to maintain $f_c$ (i.e., $R = 10k\Omega \cdot x$).

Three-Pole Butterworth Low-Pass Analog Filter

(See Figure 11.56)

1. Select the cutoff frequency $f_c$.
2. Divide the three capacitors by $2\pi f_c$.
   $$C_{1A} = \frac{5.54}{2\pi f_c} \quad C_{2A} = \frac{18}{2\pi f_c} \quad C_{3A} = \frac{20}{2\pi f_c}$$
3. Select standard capacitors with same order of magnitude.
   $$C_{1B} = \frac{C_{1A}}{x} \quad C_{2B} = \frac{C_{2A}}{x} \quad C_{3B} = \frac{C_{3A}}{x}$$
4. Adjust resistors to maintain $f_c$ (i.e., $R = 10k\Omega \cdot x$).
Two-Pole Butterworth High-Pass Analog Filter

(See Figure 11.57)

1. Select the cutoff frequency $f_c$.
2. Divide the two capacitors by $2\pi f_c$.
   \[ C_A = \frac{1\mu F}{2\pi f_c} \]
3. Select standard capacitors with same order of magnitude.
   \[ C_B = \frac{C_A}{x} \]
4. Adjust resistors to maintain $f_c$.
   \[ R_1 = 107k\Omega \cdot x \quad R_2 = 1414k\Omega \cdot x \]

Three-Pole Butterworth High-Pass Analog Filter

(See Figure 11.58)

1. Select the cutoff frequency $f_c$.
2. Divide the two capacitors by $2\pi f_c$.
   \[ C_A = \frac{1\mu F}{2\pi f_c} \]
3. Select standard capacitors with same order of magnitude.
   \[ C_B = \frac{C_A}{x} \]
4. Adjust resistors to maintain $f_c$.
   \[ R_1 = 282k\Omega \cdot x \quad R_2 = 718k\Omega \cdot x \quad R_3 = 4960k\Omega \cdot x \]

Bandpass and Band-Reject Filters

(See Figures 11.59 and 11.60)

Multiple Feedback Bandpass Filter

(See Figure 11.61)

1. Select a convenience capacitance value for the two capacitors.
2. Calculate the three resistor values for $x = 1/(2\pi f_0 C)$.
   \[ R_1 = Q \cdot x \quad R_2 = x/(2Q - 1/Q) \quad R_3 = 2 \cdot Q \cdot x \]
3. Resistors should be in the $5k\Omega$ to $5M\Omega$ range. If not, repeat with different capacitance value.
Bootstrapped Twin-T Band-Reject Filter

(See Figure 11.61)

- The notch frequency $f_0$ is:
  
  $$f_0 = \frac{1}{2\pi R_1 C_1} = 60\text{Hz}$$

  where $R_1 = R_2 = 2 \cdot R_3$ and $C_1 = C_2 = 0.5 \cdot C_3$.

Digital-to-Analog Converters

(See Figures 11.62)

DAC Parameters

- Precision is number of distinguishable DAC outputs.
- Range is maximum and minimum DAC output.
- Resolution is smallest distinguishable change in output.

  $$\text{Range (volts)} = \text{Precision (alternatives)} \cdot \text{Resolution (volts)}$$

- Accuracy is (actual-ideal)/ideal.

- Two common encoding schemes:

  $$\begin{align*}
  V_{\text{out}} &= V_{fs} \left( \frac{b_7}{2} + \frac{b_6}{4} + \frac{b_5}{8} + \frac{b_4}{16} + \frac{b_3}{32} + \frac{b_2}{64} + \frac{b_1}{128} + \frac{b_0}{256} \right) + V_{os} \\
  V_{\text{out}} &= V_{fs} \left( -\frac{b_7}{2} + \frac{b_6}{4} + \frac{b_5}{8} + \frac{b_4}{16} + \frac{b_3}{32} + \frac{b_2}{64} + \frac{b_1}{128} + \frac{b_0}{256} \right) + V_{os}
  \end{align*}$$

Three-Bit DAC Examples

(See Figures 11.63 and 11.64)
Slide 13

DAC Performance Measures

(See Figures 11.65 and 11.66)

Slide 15

DAC Using a Summing Amplifier

(See Figures 11.67 and 11.68, and Table 11.12)

Slide 14

DAC Errors: Sources and Solutions

(See Table 11.11)

Slide 16

Signed DAC Using a Summing Amplifier

(See Figures 11.69 and Table 11.13)
Three-Bit DAC with an R-2R Ladder

(See Figures 11.70, 11.71, 11.72, and 11.73)

Variable-Offset and Gain Using 3-bit DACs

(See Figures 11.74 and 11.75)

Twelve-Bit DAC with a DAC8043

(See Figure 11.76 and Table 11.14)

DAC Selection: Precision, Range, and Resolution
- Affect quality of signal that can be generated.
- More bits means finer control over the waveform.
- Can be hard to specify a priori.

(See Figure 11.77)
DAC Selection: Channels, Configuration, and Speed

- Usually more efficient to implement multiple channels using a signal DAC.
- **Configuration**: can have voltage or current outputs, internal or external references, etc.
- **Speed**: specified in many ways: settling time, maximum output rate, gain/BW product, etc.

(See Figure 11.78)

DAC Selection: Power and Interface

- Three power issues: type of power required, amount of power required, and need for low-power sleep mode.
- Three approaches for interfacing exist:

(See Figure 11.79)

DAC Selection: Package and Cost

- Variety of packages exist:

(See Figure 11.80)

- Cost includes direct cost of components, power supply requirements, manufacturing costs, labor in calibration, and software development costs.

DAC Waveform Generation

(See Figure 11.81)
Periodic Interrupt Used to Generate Waveform

```c
unsigned int wave(unsigned int t){
    float result, time;
    time = 2*pi*((float)t)/1000.0;
    // t in msec into floating point time in seconds
    result = 2048.0+1000.0*cos(31.25*time)
        -500.0*sin(125.0*time);
    return (unsigned int) result;}
```

```c
#define Rate 2000
#define OC5 0x08
unsigned int Time; // Inc every 1ms
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
    TFLG1=OC5;  // Ack interrupt
    TOC5=TOC5+Rate; // Executed every 1 ms
    Time++;
    DACout(wave(Time));}
```

---

**Generated Waveform Using Linear Interpolation**

(See Figure 11.82)

---

**Periodic Interrupt Used to Generate Waveform**

```c
Periodic Interrupt Used to Generate Waveform
```

```c
int I; // incremented every 1ms
const unsigned int wave[32]= {3048,2675,2472,
    2526,2755,2931,2597,2048,1499,1165,1139,
    1341,1570,1624,1421,1048,714,624,863,1341,1846,
    2165,2206,2048,1890,1931,2250,2755,3233,3472,3382};
```

```c
#define Rate 2000
#define OC5 0x08
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
    TFLG1=OC5;  // Ack interrupt
    TOC5=TOC5+Rate; // Executed every 1 ms
    if(++I==32) I=0;
    DACout(wave[I]);}
```

---

**Periodic Interrupt Used to Generate Waveform**

```c
Periodic Interrupt Used to Generate Waveform
```

```c
int I; // incremented every 1ms
int J; // index into these two tables
const int t[10]= {0,2,6,10,14,18,22,25,30,32};
```

```c
// time in msec
const int wave[10]={3048,2472,2931,1165,1624,
    624,2165,1890,3472,3048};
```
Periodic Interrupt Used to Generate Waveform

```c
#define Rate 2000
#define OC5 0x08
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
    TFLG1=OC5;   // Ack interrupt
    TOC5=TOC5+Rate;  // Executed every 1 ms
    if((++I)==32) {I=0; J=0;}
    if(I==t[J]) DACout(wave[J]);
    else if (I==t[J+1]){
        J++;
        DACout(wave[J]);}
    else
        DACout(wave[J]+((wave[J+1]-wave[J])
        *(I-t[J]))/(t[J+1]-t[J]));}
```

Generated Waveform Using Uneven-Time

(See Figure 11.83)

Periodic Interrupt Used to Generate an Analog Waveform

```c
unsigned int I;  // incremented every sample
const unsigned int wave[32]= { 3048,2675,2526,2817,2981,2800,2337,1901,1499,1165,
    1341,1570,1597,1337,952,662,654,863,1210,1605,1950,
    2202,2141,1955,1876,2057,2366,2755,3129,3442,3382};
const unsigned int dt[32]= { // 500 ns cycles
    2000,2000,2000,2500,2500,2000,2000,1500,1500,2000,4000,
    2000,2500,2000,2000,2000,2000,1500,1500,1500,1500,2000,
    2500,2000,2000,2000,1500,1500,1500,2000,2500,2000};
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
    TFLG1=OC5;   // Ack interrupt
    if((++I)==32) I=0;
    TOC5=TOC5+dt[I];  // variable rate
    DACout(wave[I]);}
```