D/A
(Chapter 9)
and more fixed point...
D/A

- 12 bits
- 2 channels
- Channels can be synchronized
- Multiple trigger sources
- Variable gain
Range, Resolution and Format

### Output Voltage Formulas

<table>
<thead>
<tr>
<th>Resolution</th>
<th>DAC12RES</th>
<th>DAC12IR</th>
<th>Output Voltage Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 bit</td>
<td>0</td>
<td>0</td>
<td>$V_{out} = V_{ref} \times 3 \times \frac{\text{DAC12}_x\text{DAT}}{4096}$</td>
</tr>
<tr>
<td>12 bit</td>
<td>0</td>
<td>1</td>
<td>$V_{out} = V_{ref} \times \frac{\text{DAC12}_x\text{DAT}}{4096}$</td>
</tr>
<tr>
<td>8 bit</td>
<td>1</td>
<td>0</td>
<td>$V_{out} = V_{ref} \times 3 \times \frac{\text{DAC12}_x\text{DAT}}{256}$</td>
</tr>
<tr>
<td>8 bit</td>
<td>1</td>
<td>1</td>
<td>$V_{out} = V_{ref} \times \frac{\text{DAC12}_x\text{DAT}}{256}$</td>
</tr>
</tbody>
</table>
DAC shares Port 6 pins. When DAC12AMPx != 0, the pin becomes D/A output.

### DAC12 Updating and Interrupts

- **DAC12SELx=0**
  - Latch is transparent. Data written to DAC12_xDAT goes directly to DAC. DAC12ENC bit is ignored.

- **DAC12Selx=1**
  - Data written to DAC12_xDAT is latched when written.

- **DAC12Selx=2,3**
  - Data is copied from DAC12_xDAT on rising edge of TA1 or TB2

- The interrupt is enabled when DAC12SELx!=0. The DAC12IFG must be reset by software.
DAC12 Registers

DAC12_xCTL, DAC12 Control Register

<table>
<thead>
<tr>
<th>Register</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC12_0 control</td>
<td>DAC12_0CTL</td>
<td>Read/write</td>
<td>01C0h</td>
</tr>
<tr>
<td>DAC12_0 data</td>
<td>DAC12_0DAT</td>
<td>Read/write</td>
<td>01C8h</td>
</tr>
<tr>
<td>DAC12_1 control</td>
<td>DAC12_1CTL</td>
<td>Read/write</td>
<td>01C2h</td>
</tr>
<tr>
<td>DAC12_1 data</td>
<td>DAC12_1DAT</td>
<td>Read/write</td>
<td>01CAh</td>
</tr>
</tbody>
</table>

DAC12_xDAT, DAC12 Data Register

DAC12 Data Format
- 12-bit binary: The DAC12 data are right-justified. Bit 11 is the MSB.
- 12-bit 2s complement: The DAC12 data are right-justified. Bit 11 is the MSB (sign).
- 8-bit binary: The DAC12 data are right-justified. Bit 7 is the MSB. Bits 11 to 8 are don't care and do not affect the DAC12 core.
- 8-bit 2s complement: The DAC12 data are right-justified. Bit 7 is the MSB (sign). Bits 11 to 8 are don't care and do not affect the DAC12 core.
Generating a Sine Wave

```c
void main(void)
{
    const int x[]={0, 49, 90, 117, 127, 117, 90, 49, 0, -49, -90, -117, -127, -117, -90, -49};

    // Some code is missing to set clock to 8MHz, set up output...
    ADC12CTL0 = REFON;  // Internal 1.5V ref on

    DAC12_1CTL = DAC12SREF_0 | DAC12RES | DAC12LSEL_0 | DAC12IR
                 | DAC12AMP_7 | DAC12DF;

    DAC12_1CTL |= DAC12CALON;
    while (DAC12_1CTL & DAC12CALON);

    j=0;
    while (1)   {
        DAC12_1DAT = x[j++];
        if (j>15) j=0;
        for (i=0; i<96; i++) {}  
    }
}
```

*Define array in FLASH*

*Turn on DAC12_1CTL |= ENC Vref*

*Configure DAC*

*Calibrate DAC (probably not necessary)*

*Cycle through data; write to DAC*

*Delay for 8 kHz sampling (found by trial and error)*
Output
(Interrupt Driven Code)

8 KHz updates

4 KHz updates
Same time and voltage scales
Connecting the DA in lab

• We will connect the D/A to the jack labeled “MIC OUT”, which is connected to the output of OA2 (Op-Amp2).

• The output is AC coupled (you can’t just write a value to the D/A and have it appear (and stay) on the output).

Cutoff frequency for HPF filter is

\[ f_0 = \frac{1}{2\pi (R_{28} \parallel R_{33} + R_{\text{audio}}) C_{20}} \approx 0.33\text{Hz} \text{ (for scope, } R_{\text{audio}} = 1\text{M}\Omega) \]

\[ \tau = (R_{28} \parallel R_{33} + R_{\text{audio}}) C_{20} \approx 0.47\text{ sec} \]

\[ f_0 = \frac{1}{2\pi (R_{28} \parallel R_{33} + R_{\text{audio}}) C_{20}} \approx 500\text{Hz} \text{ (for headphones, } R_{\text{audio}} = 32\Omega) \]

• Mini jack
OA Module Redux

Can set input of OpAmp (either inverting or non) to output of DAC

I’m going to give you example code for lab, but I expect you to explain it.
Experimenter’s board has SK LPF Anti-imaging filter

Figure 4: MSP430 Analog Signal Chain

OA2 need not be unity gain, but OA1 must be.

Cutoff frequency for LPF filter is

\[
 f_o = \frac{1}{2\pi \sqrt{R_{24} R_{25} C_{16} C_{17}}} = 4.0\text{kHz}
\]
Output with LPF

8 KHz updates, w LPF

4 KHz updates, w LPF

8 KHz updates, w/o LPF

16 KHz updates, w LPF
Fixed Point Multiplication Redux

- Multiplying two 4-bit Q3 number results in an 8 bit Q6 number
- To convert back to a 4 bit Q3
  - mask unwanted bits
  - shift result to the right

\[
\begin{array}{c@{\times}c}
00000101_{Q3} & 0.625 \\
11111100_{Q3} & x-0.50 \\
00000000 & 000 \\
00000000 & -3125 \\
00000101 & -.31250 \\
00000101 \\
00000101 \\
00000101 \\
00000101 \\
00000101 \\
00000101 \\
\hline
10011.101100 \\
\end{array}
\]

(= -2 + 1 + 0.5 + 0.125 + 0.0625 = -0.3125)

11.101100 becomes 1.101 (= -1 + 0.5 + 0.125 = 0.375)
Generating Sine Waves

• Method 1: Table look up.
  – This is of limited usefulness, because
    • only certain frequencies can be generated,
    • it is memory intensive

• Recursive equations:
  – Given \(\sin(\theta_{i-1})\) and \(\sin(\theta_i)\), with \(\theta_i = \theta_{i-1} + \delta\) (\(\delta\) is update interval or increment)

\[
\begin{align*}
\sin(\theta_{i+1}) &= \sin(\theta_i + \delta) = \sin(\theta_i)\cos(\delta) + \cos(\theta_i)\sin(\delta) \\
\sin(\theta_{i+1}) &= \sin(\theta_i)\cos(\delta) + 0.5(\sin(\theta_i + \delta) - \sin(\theta_i - \delta)) \\
0.5\sin(\theta_{i+1}) &= \sin(\theta_i)\cos(\delta) - 0.5\sin(\theta_i - \delta) = \sin(\theta_i)\cos(\delta) - 0.5\sin(\theta_{i-1}) \\
\sin(\theta_{i+1}) &= 2\sin(\theta_i)\cos(\delta) - \sin(\theta_{i-1}) \\
\text{let } a &= 2\cos(\delta), \\
\text{let } b &= -\sin(-\delta) = \sin(\delta) \\
\sin(\theta_{i+1}) &= a\sin(\theta_i) - \sin(\theta_{i-1})
\end{align*}
\]
Completing the Recursion

\[ a = 2 \cos(\delta), \]
\[ b = - \sin(-\delta) = \sin(\delta) \]

\[
\sin(\theta_{i+1}) = 2 \sin(\theta_i) \cos(\delta) - \sin(\theta_{i-1}) \\
= a \sin(\theta_i) - \sin(\theta_{i-1})
\]

\[
\sin(0 + \delta) = a \sin(0) - \sin(-\delta) = a \sin(0) + b = b
\]

\[
\sin(\delta) = b
\]

\[
\sin(2\delta) = a \sin(\delta) - \sin(0)
\]

\[
\sin(3\delta) = a \sin(2\delta) - \sin(\delta)
\]

\[
\sin(4\delta) = a \sin(3\delta) - \sin(2\delta)
\]

\[
\sin(\theta_{i+1}) = a \sin(\theta_i) - \sin(\theta_{i-1})
\]

or in terms of a sequence \( y_i \) : \( y_{i+1} = ay_i - y_{i-1} \), with \( y_{-1} = -b \), \( y_0 = 0 \)
Notes about Recursion

\[ y_{i+1} = ay_i - y_{i-1}, \quad \text{with } y_{-1} = -b, \quad y_0 = 0 \]
\[ a = 2\cos(\delta), \]
\[ b = -\sin(-\delta) = \sin(\delta) \]

• As an example, if you are updating a sine wave at 8 kHz, and want the sine wave to be 1 kHz, this gives an update angle of \( \delta = \frac{2\pi}{8}, \) or \( 45^0 \) (so there are 8 points per cycle).

• Note that you don’t need an integer number of points per cycle. Changing the sine wave frequency to 330 Hz yields an update angle of \( \delta = \frac{2\pi*0.33}{8} = 14.85^0 \) (so there are 24.24 points per cycle). This is a substantial advantage of the recursion method over the table-lookup method.

• If you took E71 (or really paid attention in E12) this is simply the zero-input (i.e., there are non-zero initial conditions) response of a system with poles on the unit circle (that is, there is no decay or growth; like poles on \( j\omega \) axis in continuous time systems).

• It is very closely related to the impulse response of the system.
In Lab

• You’ll generate a sine wave using recursion with 8-bit variables (either Q6 or Q7).
• Beware of overflow (the largest number in Q7 is just less than 1). Solutions:
  – start recursion with a value a little less than sin(-T)
  – use Q6 (less precision, but maximum is just less than 2)
References

- Mini-jack image: [http://www.techpowerup.com/reviews/Razer/m100/images/minijack.jpg](http://www.techpowerup.com/reviews/Razer/m100/images/minijack.jpg)
- MSP430FG4618/F2013 Experimenter’s Board User's Guide