The LPC2138 has two on-board analog-to-digital converters each of which provides 10-bits of accuracy with a conversion time of about 2.44 μsec. Each converter has an 8-channel analog front end so that there are 16-channels of A/D available. The converters can be programmed to do single or multiple conversions in a burst mode on multiple channels. Be sure to program the pin select register to change the appropriate bits to analog input before doing a conversion. See Lecture 3 for a pin select register map.

There are three registers that control the A to D converter:

- **ADCR** – A to D Control register. Controls the A to D mode.
  - Selects channels to be converted
  - Sets up the Peripheral clock divider
  - Sets the mode of the converter
  - Determines the number of bits in the conversion (up to 10)
  - Sets the power down mode to save power
  - Has start bits for the burst mode

- **ADDR** – A to D Data register
  - Has the done bit
  - Holds the converted data

- **ADGST** – A to D Global Start Register
  - Has the start bit for the converter

For the ARM7 board that we are using in this class there are only six A to D channels brought out to pins as shown in the table below.

<table>
<thead>
<tr>
<th>A/D Channel</th>
<th>CPU Pin</th>
<th>CN2 Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD0.0</td>
<td>P0.27</td>
<td>4</td>
</tr>
<tr>
<td>AD0.1</td>
<td>P0.28</td>
<td>7</td>
</tr>
<tr>
<td>AD0.2</td>
<td>P0.29</td>
<td>8</td>
</tr>
<tr>
<td>AD0.3</td>
<td>P0.30</td>
<td>9</td>
</tr>
<tr>
<td>AD1.0</td>
<td>P0.6</td>
<td>13</td>
</tr>
<tr>
<td>AD1.1</td>
<td>P0.8</td>
<td>10</td>
</tr>
</tbody>
</table>

A to D Channels

Only AD0.0 and AD1.0 have protection diodes that keep the input voltage from going more than 1 diode drop below ground or one diode drop above the 3.3 volt supply. The other channels which are brought out to CN 2 have no such protection and their voltage values should be kept in the range 0 to 3 volts.

For this class we will be using the A to D converter in a non-burst mode doing a single conversion on a single channel. Most of the set up for the A to D converter is done in the control register (ADCR).
The A to D Converter Control Register (ADCR)

The first 8-bits of ADCR control the analog channel being converted with 1-bit per channel. To operate on channel 0 these 8 bits should be 0000 0001.

The second set of 8-bits on ADCR control the A to D clock which is derived from the peripheral clock (PCLK). The PCLK is divided by these 8 bits plus 1 and must result in an A to D clock of 4.5MHz or less. For example, if the PCLK is running at 30MHz then we need to divide 30MHz/4.5MHz ≤ (number + 1). Since we need an A to D clock less than or equal to 4.5MHz we take number + 1 = 7 bits. So we need to set bits 8 through 15 to 6 or 0000 0110 to get an A to D clock of 30MHz/7 = 4.28MHz.

We want single channel conversions one at a time (not in the burst mode) so we set bit-16 to 0.
Bits 17, 18, and 19 determine how fast the burst mode runs and since we are not using the burst mode we can set these three bits to 000.

Bit-20 is not used and bit-21 is the power down mode bit. Set it to 1.

Bits 22 and 23 are used for device testing and should be set to 00 for normal operation.

Bits 24, 25, and 26 are used to start the converter. We will set these bits to 001 when we are ready to do the conversion. For now set them to 000.

Bits 27 to 31 are not being used so set them to 0000 0.

Putting all of these settings together gives a value of 0x00200601 in hex for the ADCR register. For example,

```
VPBDIV = 0x02;       // Set the PCLK to 30 MHz
AD0CR  = 0x00200601; // Setup A/D: 10-bit AIN0 @ 4.28MHz software controlled
```

Sets the Peripheral clock divisor to divide the 60MHz CCLK by 2 to get 30 MHz. The A to D Control register uses bits 15:8 to act as an A to D clock prescaler. AD0CR = 0x00200601 = 0000 0000 0010 0000 0000 0110 0000 0001 so that bits 15:8 = 00000110 = 6. 30MHz/(6 + 1) = 4.2857MHz. Bits 7:0 select the channel number (in this case channel 0 is selected). Bits 21 = 1 which makes the A to D converter operational (not in power down mode).

The conversion can be started by setting bits 26:24 to 001. For example

```
AD0CR  |= 0x01000000;   // Start A/D Conversion
```

Once the conversion is started we can read the data out of the A to D Data Register (ADDR).

<table>
<thead>
<tr>
<th>ADDR</th>
<th>Name</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:0</td>
<td>Reserved</td>
<td>Reserved. These bits always read as zeroes. User should not write ones to reserved bits.</td>
<td>D</td>
</tr>
<tr>
<td>15:6</td>
<td>V/VSA</td>
<td>When DONE is 1, this field contains a binary fraction representing the voltage on the Ain pin selected by the SEL field, divided by the voltage on the VDD pin. Zero in the field indicates that the voltage on the Ain pin was less than, equal to, or close to that on VDD, while 0x3F indicates that the voltage on Ain was close to, equal to, or greater than that on VDD. For testing, data written to this field is captured in a shift register that is clocked by the A/D converter clock. The MS bit of this register sources the DINSERI input of the A/D converter which is used only when TEST1:0 are 10.</td>
<td>X</td>
</tr>
<tr>
<td>23:16</td>
<td>Reserved</td>
<td>Reserved. These bits always read as zeroes. User should not write ones to reserved bits.</td>
<td>D</td>
</tr>
<tr>
<td>26:24</td>
<td>CHN</td>
<td>These bits contain the channel from which the LS bits were converted.</td>
<td>X</td>
</tr>
<tr>
<td>29:27</td>
<td>Reserved</td>
<td>Reserved. These bits always read as zeroes. User should not write ones to reserved bits.</td>
<td>D</td>
</tr>
<tr>
<td>30</td>
<td>OVERUN</td>
<td>This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the LS bits. In non-RIFO operation, this bit is cleared by reading this register.</td>
<td>D</td>
</tr>
<tr>
<td>31</td>
<td>DONE</td>
<td>This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.</td>
<td>D</td>
</tr>
</tbody>
</table>

The A to D Converter Data Register (ADDR)
The most significant bit in the A to D Data register is the done bit. This bit changes to 0 when a conversion is started and goes to one when the conversion is done. The converted 10-bits of data is stored in bits 15 to 6. We can ignore the other bits in this register.

For example, we can run the following loop after starting a conversion.

```c
val = AD0DR;
while ((val & 0x80000000) == 0) // Wait for the conversion to complete
    val = AD0DR;
DACR = (val & 0x0000FFC0) << 6;
```

where val is an unsigned int. Note that the data comes in to AD0DR as bits 15:6. They are thus shifted over 6 places (exactly as they are for the D to A converter output).

For single conversions the A to D Global Start register (ADGSR) needs to be set to all zeros and since this is its reset value it can be ignored.

### Table 139: A/D Global Start Register (ADGSR - 0x0034008)

<table>
<thead>
<tr>
<th>ADCR</th>
<th>Name</th>
<th>Description</th>
<th>Reset/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>Reserved</td>
<td>User software should not write ones to reserved bits. The value read from a reserved bit is not defined.</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>BURST</td>
<td>If this bit is 0, conversions are software controlled and require 11 clocks. If this bit is 1, the A/D converters do repeated conversions at the rate selected by their CLKS fields, scanning (if necessary) through the pins selected by 1s in the SEL fields. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that’s in progress when this bit is cleared will be completed.</td>
<td>0</td>
</tr>
</tbody>
</table>

### A to D Global Start Register (ADGSTR)

The program AtoDTest.c does an A to D conversion on AD0.0 and sends its value to the D to A converter. No clock is used to determine when conversions should take place so they run as fast as possible.

/*AtoDTest.c                                  July 6, 2007
   This program inputs from the A to D converter and outputs
   directly to the D to A converter.
*/
#include <LPC213X.H>

int main(void)
{
    unsigned int val;
    PINSEL1 = 0x00480000; // P0.25 set to DA Out, P0.27 set to input AD0.0
    VPBDIV = 0x02;        // Set the Pclk to 30 MHz
    AD0CR  = 0x00200601;  // Setup A/D: 10-bit AIN0 @ 4.28MHz
    AD0CR  |= 0x01000000; // Start A/D Conversion
    val = 0;
    while(1)
    {
        val = AD0DR;          // Read A/D Data Register
        while ((val & 0x80000000) == 0) // Wait for conversion to complete
            val = AD0DR;
        DACR = val;            // Output to D/A
        AD0CR  |= 0x01000000;  // Restart A/D Converter
    }
}
Class Exercise 9-1: It is difficult to measure how fast the A to D is running using AtoDTest.c since it outputs what it inputs. A better way to do this is to allow all the conversion to take place but to output a square wave to the D to A instead of the A to D input value. You can do this by creating a variable and assigning it a value and complementing it each time you go through a cycle. For example,

```c
int dtoaOut = 0;
and
replace DACR = val;
with
dtoaOut = ~dtoaOut & 0x00000FFC0;
DACP = dtoaOut;
```

The high time (or the low time) of the resulting square wave will be the time it takes the processor to go through one cycle of A to D and D to A conversion (plus a few instructions). This will be approximately the same as the A to D conversion time.

Class Exercise 9-2:
Exercise 9-1 establishes the fastest possible sampling frequency for your ARM system. Using the same program as in AtoDTest.c input sinusoids that go beyond this sample frequency and explain the output.