There is one 10-bit A/D converter on the LPC1114 ARM Cortex M0 processor. It is multiplexed out to 5 analog inputs on P0.11, P1.0, P1.1, P1.2, and P1.3. The converter has the following features:

- 10-bit successive approximation Analog-to-Digital Converter (ADC).
- Input multiplexing among 5 pins.
- Power-down mode.
- Measurement range 0 to 3.6 V. Do not exceed the VDD voltage level.
- 10-bit conversion time ≥ 2.44 μs.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.
- Individual result registers for each A/D channel to reduce interrupt overhead.

### Table 1

**ADC Registers**

<table>
<thead>
<tr>
<th>Name</th>
<th>Access</th>
<th>Address offset</th>
<th>Description</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD0CR</td>
<td>R/W</td>
<td>0x000</td>
<td>A/D Control Register. The AD0CR register must be written to select the operating mode before A/D conversion can occur.</td>
<td>0x0000 0000</td>
</tr>
<tr>
<td>AD0GDR</td>
<td>R/W</td>
<td>0x004</td>
<td>A/D Global Data Register. Contains the result of the most recent A/D conversion.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0INTEN</td>
<td>R/W</td>
<td>0x008</td>
<td>Reserved.</td>
<td>-</td>
</tr>
<tr>
<td>AD0DR0</td>
<td>R/W</td>
<td>0x010</td>
<td>A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR1</td>
<td>R/W</td>
<td>0x014</td>
<td>A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR2</td>
<td>R/W</td>
<td>0x018</td>
<td>A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR3</td>
<td>R/W</td>
<td>0x01C</td>
<td>A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR4</td>
<td>R/W</td>
<td>0x020</td>
<td>A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR5</td>
<td>R/W</td>
<td>0x024</td>
<td>A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR6</td>
<td>R/W</td>
<td>0x028</td>
<td>A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0DR7</td>
<td>R/W</td>
<td>0x02C</td>
<td>A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.</td>
<td>NA</td>
</tr>
<tr>
<td>AD0STAT</td>
<td>RO</td>
<td>0x030</td>
<td>A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt flag.</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that in the digital mode the A/D pins are 5 volt tolerant but in the analog mode they are not and the analog input should not exceed 3.6 volts.
The ADC is powered off by default so as to conserve energy since many applications don't use it. Bit 4 in the power-down configuration register, PDRUNCFG, must be set to 0 to power up the ADC.

The clock for the ADC must be set to something less than 4.5 MHz. The clock divider must divide the system clock to get this clock rate. A typical full conversion takes eleven clock cycles making a conversion take $11/4.5 \text{ MHz} = 2.44 \mu\text{sec}$. 

To configure the I/O pins for analog input the appropriate mode must be selected in the I/O Configuration Register (IOCON). Likewise, the configuration register should set the hysteresis and mode function to zero.

The ADC has two modes of operation: burst mode and software controlled mode. In burst mode the ADC automatically does a conversion on the channels selected and stores the result in memory. We will not be using burst mode in this class. In the software controlled mode the program starts the conversion on a single channel and the conversion requires 11 clock cycles to complete.

The user can determine when a conversion is complete by looking at bit-31 in the A/D data register. There is one such register for each A/D channel. The result of the conversion is stored in this register as well.

To do a complete conversion in the software mode the following steps are required:

- Power up the A/D by setting bit 4 in PFRUNCFG to 0.
- Enable the A/D clock in the SYSAHBCLKCTRL register.
- In the IO Configuration register set bit 7 to 0 to select the Analog input mode and set the lower three bits to the lower three bits to 2 to get the A/D function.
- Set up the AD0CR register:
  - Clock divisor = 11 makes A/D clock = $48 \text{ MHz}/11 = 4.36 \text{ MHz}$
  - Select A/D channel – say channel 0
- Start the conversion by setting bit 24 in AD0CR to 1.
- Wait for the conversion to end. This sets bit 31 in the AD0 data register to 1.

It is also possible to get the end of conversion to trigger an interrupt (IRQ24) by enabling the interrupt in the appropriate register and writing the interrupt service routine.
Example 1
This example does an A/D conversion on channel 0 and sends the result to the PWM channel.

```c
//AtoD.c
//Inputs from AD converter channel 0 on pin P0.11 and outputs 10 bits to
// PWM channel on P1.9
#define PDRUNCFG (*((volatile unsigned long *) 0x40048238))   //Power-down configuration register
#define SYSAHBCLKCTRL (*((volatile unsigned long *) 0x40048080))   //System AHB clock control
#define IOCON_R_PIO0_11 (*((volatile unsigned long *) 0x40044074))   //Pin control register
#define AD0CR (*((volatile unsigned long *) 0x4001C000))   //A/D Control
#define AD0DR0 (*((volatile unsigned long *) 0x4001C010))   //A/D Ch 0 Data
#define IOCON_PIO1_9 (*((volatile unsigned long *) 0x40044038))   //Pin control register
#define TMR16B1TCR (*((volatile unsigned long *) 0x40010004))   //Timer 1 Timer control register
#define TMR16B1PR (*((volatile unsigned long *) 0x4001000C))   //Timer 1 Prescale register
#define TMR16B1MR3 (*((volatile unsigned long *) 0x40010024))   //Timer 1 Match register 3
#define TMR16B1PWMC (*((volatile unsigned long *) 0x40010074))   //Timer 1 PWM Control register
#define TMR16B1MCR (*((volatile unsigned long *) 0x40010014))   //Timer 1 Match Control register
#define TMR16B1MR0 (*((volatile unsigned long *) 0x40010018))   //Timer 1 Match register 0

int main()
{
    PDRUNCFG &= ~(1 << 4);                   //Power up A/D
    SYSAHBCLKCTRL |= (1 << 13);              //ADC Clock enable; channel 0 selected
    IOCON_R_PIO0_11 &= 0xFFFFFF78;
    IOCON_R_PIO0_11 |= (1 << 1);
    AD0CR = 0x0B01;                  //Clock divider is 0x8
    //************** Set up PWM
    SYSAHBCLKCTRL |= (1 << 8);  //Enable clock for 16-bit timer 1
    IOCON_PIO1_9 |= 1;       //Selects match function so Pin1_9 is PWM
    TMR16B1PR = 1;           //Prescale register. Divide P Clock by 2 = 24 MHz
    TMR16B1MR3 = 1024;       //Match register 3. TR and Pin1_9 Reset
    TMR16B1MCR |= (1 << 10); //Causes TimerCounter to be reset if match on MR3
    TMR16B1PWMC |= 1;        //Enable PWM on Channel 1
    TMR16B1TICR |= 1;       //Enable TimerCounter to run
    while (1)
    {
        AD0DR0 |= (1 << 24);   //Start conversion
        while(AD0DR0 < 0x7FFFFFFF); //Wait for done bit
        TMR16B1MR0 = ((AD0DR0 & 0xFFC0) >> 6); //Data to PWM
    }
}
```
Example 2
This program inputs from the A/D converter, converts the raw data to ASCII and sends it to the UART.

//AtoDtoUART.c
//Converts A to D input to float and sends result to UART
#define U0RBR (*((volatile unsigned long *) 0x40008000))   //Receive Buffer
#define U0THR (*((volatile unsigned long *) 0x40008000))   //Transmit Buffer
#define U0DLL (*((volatile unsigned long *) 0x40008000))   //Divisor Latch LSByte
#define U0DLM (*((volatile unsigned long *) 0x40008004))   //Divisor Latch MSByte
#define U0IER (*((volatile unsigned long *) 0x40008004))   //Interrupt Enable
#define U0IIR (*((volatile unsigned long *) 0x40008008))   //Interrupt ID
#define U0FCCR (*((volatile unsigned long *) 0x40008008))   //FIFO control
#define U0LCR (*((volatile unsigned long *) 0x4000800C))   //Line control
#define U0MCR (*((volatile unsigned long *) 0x40008010))   //Modem control
#define U0LSR (*((volatile unsigned long *) 0x40008014))   //Line Status
#define U0MSR (*((volatile unsigned long *) 0x40008018))   //Modem status
#define U0SCR (*((volatile unsigned long *) 0x4000801C))   //Scratch pad register
#define U0ACR (*((volatile unsigned long *) 0x40008020))   //Autobaud control
#define U0FDR (*((volatile unsigned long *) 0x40008028))   //Fractional divide register
#define U0TER (*((volatile unsigned long *) 0x40008030))   //Transmit enable
//Enable TxD to output
#define IOCON_PIO1_7 (*((volatile unsigned long *) 0x400440A8))   //Pin control register
#define SYSAHBCLKCTRL (*((volatile unsigned long *) 0x40048080))   //System AHB clock control
#define UARTCLKDIV (*((volatile unsigned long *) 0x40048098))   //UART clock divider
//
#define PDRUNCFG (*((volatile unsigned long *) 0x40048238))   //Power-down configuration register
#define IOCON_R_PIO0_11 (*((volatile unsigned long *) 0x40044074))   //Pin control register
#define AD0CR (*((volatile unsigned long *) 0x4001C000))   //A/D Control
#define AD0DR0 (*((volatile unsigned long *) 0x4001C010))   //A/D Ch 0 Data
#define IOCON_PIO1_9 (*((volatile unsigned long *) 0x40044038))   //Pin control register
#include<math.h>
int IntToASCII(char buffer[], int ADInt);
int FloatToASCII(char buffer[], float ADFloat);
const char toASCII[] = "0123456789";

int main()
{
    int i, n, ADIn;
    char buffer[10];
    PDRUNCFG &= ~(1 << 4);                   //Power up A/D
    SYSAHBCLKCTRL |= (1 << 13);              //ADC Clock enable
    IOCON_R_PIO0_11 &= 0xFFFFFFF78;
    IOCON_R_PIO0_11 |= (1 << 1);
    IOCON_PIO1_7 |= 0x01;                     //TxD enable to output
    SYSAHBCLKCTRL |= (1 << 12);              //Clock to UART
    UARTCLKDIV |= 2;                        //48000000/2 = 24 MHz for UARTPCLK
    U0FCR |= 0x01;                           //Enable FIFO buffer
    U0LCR |= 1 << 7;                         //DLAB must be 1 to write to U0DLL, and U0FDR
    //divAdd=1 mulVal=4 dll=250
    U0FDR = 0x41;                            //DIVADDDVAL is in bits 0-3
    //MULVAL is in bits 4-7
    U0DLL = 250;
    U0LCR &= ~(1 << 7);                      //DLAB back to 0 to transmit
    U0LCR |= 0x03;                           //8-bit data width
    U0TER |= 0x80;                           //Enable transmit

    AD0CR = 0x80A1;
    while (1)
    {
        AD0CR |= (1 << 24);                  //Start conversion
    }
}

8-4
while(AD0DR0 < 0x7FFFFFFF);  //Wait for done bit
ADIn = ((AD0DR0 & 0xFFC0) >> 6);  //ADIn is raw 10-bit data
n = IntToASCII(buffer, ADIn);   //Convert to ASCII
buffer[n] = 0x0d;  //Add CRLF to buffer
buffer[n+1] = 0x0a;
n = n + 2;
for(i=0;i<n;i++)               //Print n-characters to UART
  {while(!(U0LSR & 0x20));
    U0THR = buffer[i];
  }
}

//Converts ADInt to ascii characters and stores result in buffer
//  The number of characters in the buffer is returned.
int IntToASCII(char buffer[], int ADInt)
{
  int n;        //total number of characters
  int m = 5;    //largest number of whole number places
  int i, index, digit;
  if((int)ADInt > 0)       //If input is not 0 find out how many
    {i = m;            //  digits there are.
      while(pow(10, i) > (int)ADInt)
        i--;
      m = i+1;
    }
  else
    m = 1;               //If ADInt is 0 we will return 1 digit
  index = m;             //top index in buffer
  n = m;                 //Return m digits
  i = 0;
  index--;              //Find the digits and store them in ASCII in the buffer
  while(i < m)
    {digit = (int)(ADInt/(pow(10, i))) % 10;
     buffer[index] = toASCII[digit];  //Convert digit to ASCII
     index--;
     i++;
    }
  return n;
}