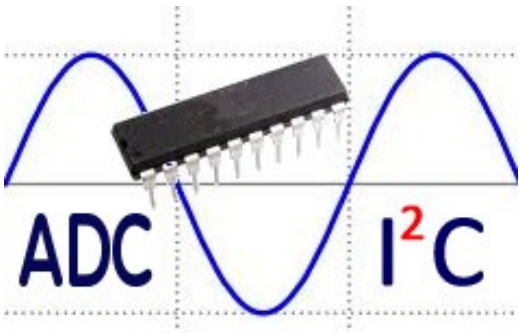
ADC to I²C

Data Sheet

10 Channel Analog to Digital Converter

with output via I²C



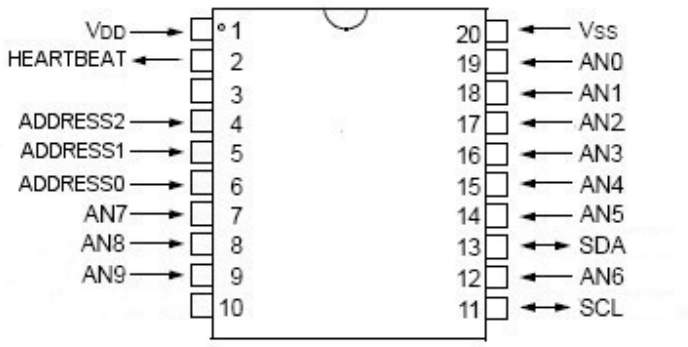
Introduction

Many microcontroller projects involve the use of sensors like Accelerometers, Gyroscopes, Temperature, Compass, Barometric, Current, Proximity and others. Some sensors have I²C or SPI interfaces but there are still a great many which produce an analogue output voltage. These are measured with the use of the built-in Analogue to Digital Converters (ADC) included in microcontrollers. However, if you are using many sensors and don't have enough ADC inputs, or have used those pins for other functions, you either have to upgrade to a larger microcontroller with more inputs or resort to using an external (and expensive) ADC chip.

With this in mind we have programmed our own microcontroller to provide you with 10 additional ADC inputs available via I²C as a slave device.

No additional components are required as it has its own built in oscillator clock circuit.

Pinout



Specifications

- 10 ADC input channels, each with 10 bit resolution (AN0 through AN9)
- I²C interface
- Configurable Slave Address between 40 and 47 allows up to 8 units to be added to your project
- 64MHz Clock Speed
- ADC values are measured 1000 times per second (1 kHz)
- Selectable Complementary Filter function
- Selectable 10-bit (2 bytes per ADC) or 8-bit (1 byte per ADC) output
- Heartbeat output to show unit is functioning
- 5V or 3.3V operation
- Each ADC input is read 3 times and the average value taken to reduce spurious values

Hardware Configuration

Slave Address

Pins 4, 5 and 6 are used to configure the slave address of the device. The device has a default slave address of 40, but by tying these pins high or low, the address can be altered. For the default address tie all the pins low.

Slave Address	Address2 (pin4)	Address1 (pin5)	Address0 (pin 6)
40	0	0	0
41	0	0	1
42	0	1	0
43	0	1	1
44	1	0	0
45	1	0	1
46	1	1	0
47	1	1	1

Software Configuration

There are some configuration options that are *optionally* configured from the Master device by writing to memory location 0.

Configuration Byte

7	6	5	4	3	2	1	0
NA	NA	NA	NA	NA	FIL1	FIL0	RES

RES Output Resolution (default 1)
Determines whether the output is the full 10 bit resolution or 8 bit resolution. 10 bit resolution requires 2 output bytes per channel, whereas 8 bit only requires 1 byte. 8 bit resolution is the 10 bit value divide by 4.
0 = 10bit (0 to 1024)
1 = 8bit (0 to 255) - **default**

FIL1, FIL0 Filter Output (default 0, 0)
Determines whether the output is the direct ADC value last measured or a filtered output

Output	FIL1	FIL0
ADC Value	0	0
Complementary Filter	0	1
Future Use	1	0
Future Use	1	1

Filters

Because a large number of sensors produce noisy output, it is often required to filter the voltage output from these sensors. This can be done in hardware via Low Pass RC Filters but can also be done in software. Using any filter has the effect of reducing the response rate of a sensor.

ADC Value

This is simply the most recent ADC value recorded for each channel

Complementary Filter

The complementary filter is a useful filter function which will reduce spurious values by only applying a percentage of the new ADC reading to the stored value. In this function, 10% of the new reading is added to 90% of the stored value. This simple filter is based on the following function

value = (value * 0.9) + (New ADC reading * 0.1)

Heartbeat

A heartbeat output is available to show the unit is functioning.
The heartbeat is a short pulse of about 100uS duration each time the Analog voltages are read, so the output is a 1kHz pulse. This can be checked on an oscilloscope or can be fed to an LED (via appropriate resistor) to produce a (dim) output.

Example Code

Here is an Arduino Sketch showing how the device can be read.
Note that the device is not limited to being used with An Arduino board, any I²C capable microcontroller can be used.

```
/*
** Wire Master Reader of Hobbytronics 10 channel ADC
** Created 06 Feb 2010
**
** This example code is in the public domain.
** www.hobbytronics.co.uk
*/

#include <Wire.h>

const int  adc_address=40;   // I2C Address of ADC Chip
const int  adc_config=B00000011;  // 8-bit, complementary filter on (see datasheet)

unsigned int adc_data[10];   // Array to store ADC values

void setup()
{
  //Send settings to ADC_I2C device (optional)
  Wire.begin();              // join i2c bus (address optional for master)
  Wire.beginTransmission(adc_address); // transmit to device
  Wire.write(adc_config);     // send config options
  Wire.endTransmission();    // stop transmitting

  //Start Serial port
  Serial.begin(9600);        // start serial for output
}

void get_adc_8()
{
  // Get 8-bit ADC data
  // We only want single byte values (0 to 255)
  unsigned char data_values_rcvd=0;  // keep track of how many characters received
  Wire.requestFrom(adc_address, 10);    // request 10 bytes from slave device
  data_values_rcvd=0;
  while(Wire.available())
  {
    if(data_values_rcvd < 10) adc_data[data_values_rcvd] = Wire.read(); // receive a byte as
character
    data_values_rcvd++;
  }
}

void get_adc_10()
{
  // Get 10-bit ADC data
  // We want double byte values (0 to 1024)
  // We will reconstruct an unsigned int from the two bytes received
  unsigned char hi_lo_byte;
  unsigned char data_values_rcvd=0;  // keep track of how many characters received

  Wire.requestFrom(adc_address, 20);    // request 20 bytes from slave device
  data_values_rcvd=0;
  hi_lo_byte=1;                         // First byte will be a high byte
  while(Wire.available())
  {
    if(data_values_rcvd < 10)
    {
      if(hi_lo_byte)
      {
        // Its a high byte
        adc_data[data_values_rcvd] = Wire.read() << 8; // receive high byte and shift left 8
```

```
        hi_lo_byte=0;    // next byte will be a low byte
      }
      else
      {
        adc_data[data_values_rcvd] += Wire.read(); // receive low byte and add to existing value
        data_values_rcvd++;
        hi_lo_byte=1;    // next byte will be a high byte
      }
    }
  }
}

void loop()
{
  unsigned char i;

  if(bitRead(adc_config, 0)) get_adc_8();  // get 8-bit data
  else get_adc_10();                       // get 10-bit data

  // Print out our data
  for(i=0;i<10;i++)
  {
    Serial.print(adc_data[i], DEC);     // print the character
    Serial.print(",");                  // print comma
  }
  Serial.println("");                   // print carriage return
  delay(500);
}
```