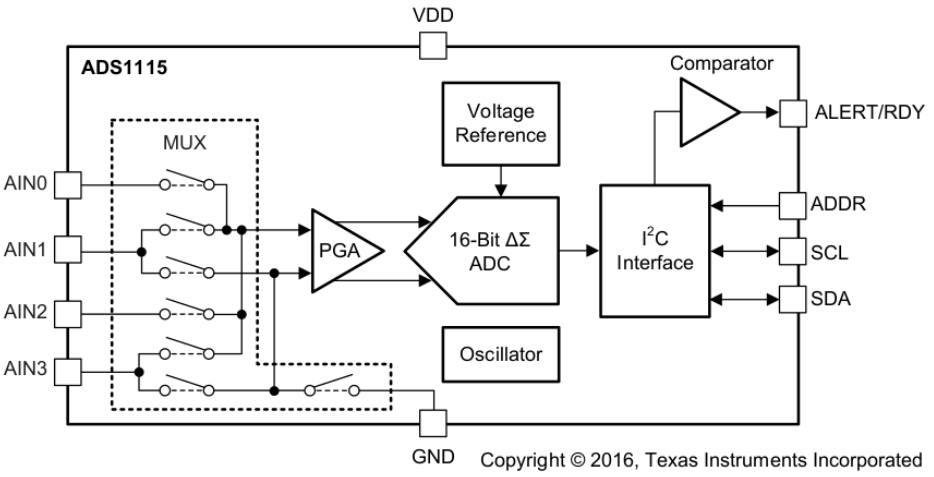
Analog-to-Digital Converter Module

Operations

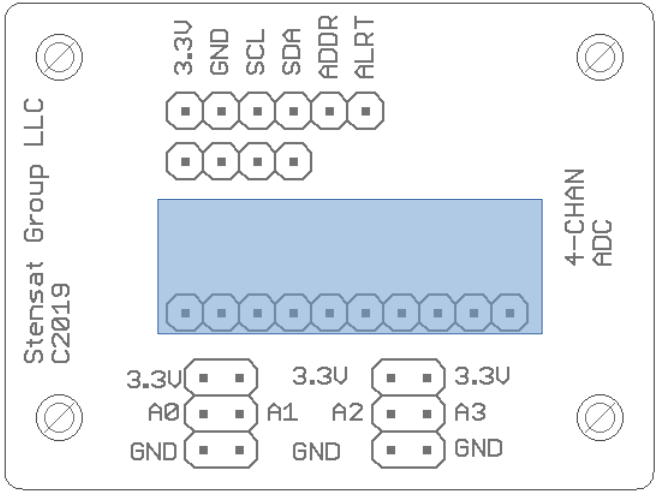
Introduction

- The Analog-to-Digital Converter or ADC is a module that converts voltages to a digital value.
- The module uses the Texas Instruments ADS1115 which can measure four channels with a resolution of 16-bits.
- The ADC has a programmable gain amplifier which allows different ranges of voltages to be measured.
- Communications is through the I2C bus.



ADC Module

- The ADC module provides easy connections.
- Power and I2C bus are broken out along with the ADDR pin and ALERT pin from the ADC. A spare set of four pins aligned with power and the I2C signals provide the ability to daisy-chain multiple I2C based modules.
- On the bottom side are the connections to the analog inputs. Each input includes a ground and 3.3V power to support voltage divider circuits or sensors that require power. A0 is channel 0, A1 is channel 1, A2 is channel 2 and A3 is channel 3.



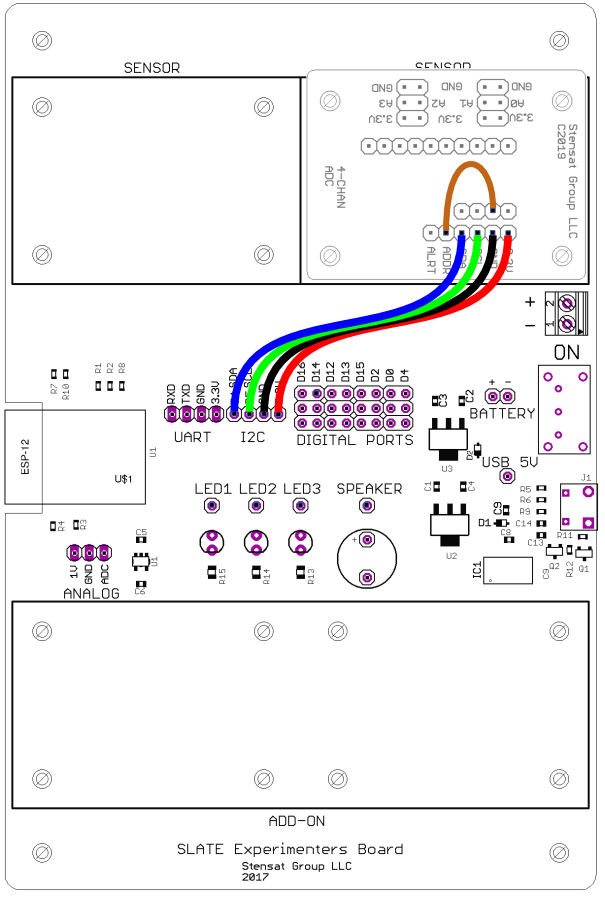
ADC Interface

- The ADC can be configured to operate at one of four available device addresses. This allows up to four ADC modules to be used on an I2C interface.
- The address is selected based on where the ADDR signal is connected, 3.3V, GND, SDA signal or SCL signal.

ADDR Conncetion	Address
GND	0x48
3.3V	0x49
SDA	0x4a
SCL	0x4b

ADC Connection

- Connections are shown to the right using the SLATE board.
- A jumper from ADDR to GND sets the address of the ADC to 0x48.



ADC Configuration

- There is one 16-bit configuration register that needs to be set before using the ADC.
- The bits in the configuration register are used to set the channel input to measure the voltage range and the data rate for continuous measurements.
- Below is the configuration register showing all the bit assignments.

15	14	13	12	11	10	9	8
OS	MUX[2:0]			PGA[2:0]			MODE
R/W-1h	R/W-0h			R/W-2h			R/W-1h
7	6	5	4	3	2	1	0
DR[2:0]			COMP_MODE	COMP_POL	COMP_LAT	COMP_QUE[1:0]	
R/W-4h			R/W-0h	R/W-0h	R/W-0h	R/W-3h	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

ADC Configuration

- Bit 15 – OS, This is the operational status bit. Logic 1 indicates conversion in progress.
- Bits 14-12 – MUX[2:0], these three bits select the input configuration. Both differential input configuration or single ended configuration can be selected. We will use single ended configuration.
- Bits 11-9 – PGA[2:0], these three bits set the voltage range on the input. Up to 6.144 volts can be selected but the input range can only be up to the power input which is 3.3 volts. By default, +/- 2.048 volts is the range. Higher ranges means that the input will not reach full scale of the ADC.
- Bit 8 – MODE, This sets the ADC to operate in single conversion of continuous mode. Setting it to 1 is single-shot mode and power down.

ADC Configuration

- Bits 7-5 – DR[2:0], These three bits set the data rate. The range is from 8 samples/sec to 860 samples/sec.
- The rest of the bits are for comparator operations. These bits will not be used. Read the datasheet if you are interested in using the camparator functions.

ADC Operation

- The following code will set sample rate of the ADC to 128 samples/sec. The ADC will convert the voltage to a value in 7.8 ms after initiating the conversion.
- The following example will operate the ADC in single conversion mode.
- The configuration register will be written to configure the device and initiate a conversion. The program will wait 10 ms and read the results. The 10 ms gives the ADC time to perform the conversion.
- The code to the right is a function to get data from the ADC.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
  unsigned char b;
  b = (addr << 4) | 0xc3;
  Wire.beginTransmission(ADC);
  Wire.write(0x01);
  Wire.write(b);
  Wire.write(0x83);
  Wire.endTransmission();
  delay(10);
  Wire.beginTransmission(ADC);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(ADC,2);
  while(Wire.available() < 2) delay(1);
  int c = Wire.read();
  int d = Wire.read();
  short res = (c << 8) + d;
  return res;
}
```

ADC Operation

- ADC is defined as the address with the ADDR pin connected to a GND pin.
- The function **readADC()** will configure the ADC and set the multiplexer input based on the parameter addr.
- **b** is used to combine the input selection with the other configuration bits. The input selection is shifted 4 bits in the direction of the most significant bit to the MUX[] bit locations.
- **addr** is to have a value range of zero to three to select one of the four ADC inputs.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
  unsigned char b;
  b = (addr << 4) | 0xc3;
  Wire.beginTransmission(ADC);
  Wire.write(0x01);
  Wire.write(b);
  Wire.write(0x83);
  Wire.endTransmission();
  delay(10);
  Wire.beginTransmission(ADC);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(ADC,2);
  while(Wire.available() < 2) delay(1);
  int c = Wire.read();
  int d = Wire.read();
  short res = (c << 8) + d;
  return res;
}
```

ADC Operation

- The first I2C operation is to set the ADC configuration register by sending two configuration bytes.
- First, the address 0x01 is sent to select the configuration register.
- The first configuration byte is then sent. OS bit is set to start the conversion. The gain is set to +/- 4.096 volts, the input is selected and MODE is set to single conversion.
- The second configuration byte sets the sample rate to 128 samples/sec and the default values for the comparator functions.
- After the write to the configuration register, the program waits 10 ms.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
  unsigned char b;
  b = (addr << 4) | 0xc3;
  Wire.beginTransmission(ADC);
  Wire.write(0x01);
  Wire.write(b);
  Wire.write(0x83);
  Wire.endTransmission();
  delay(10);
  Wire.beginTransmission(ADC);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(ADC,2);
  while(Wire.available() < 2) delay(1);
  int c = Wire.read();
  int d = Wire.read();
  short res = (c << 8) + d;
  return res;
}
```

ADC Operation

- After the short wait, the address of the data conversion is then sent to the ADC. The **endTransmission()** is sent with a false flag indicating to the I2C to restart.
- A read request for 2 bytes is then sent.
- The program waits for 2 bytes to become available and the values are then read.
- The most significant byte is read first followed by the least significant.
- The two bytes read are assembled and put into the **res** variable.
- The results are then returned.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
  unsigned char b;
  b = (addr << 4) | 0xc3;
  Wire.beginTransmission(ADC);
  Wire.write(0x01);
  Wire.write(b);
  Wire.write(0x83);
  Wire.endTransmission();
  delay(10);
  Wire.beginTransmission(ADC);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(ADC,2);
  while(Wire.available() < 2) delay(1);
  int c = Wire.read();
  int d = Wire.read();
  short res = (c << 8) + d;
  return res;
}
```

Part 1 of 2 Code. See next page for the rest of the code

ADC Operation

- With the **readADC()** function completed, the main code can be written.
- The **setup()** function initializes the serial interface and the I2C interface.
- The **loop()** function loops through the four inputs and displays the value from each.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
 /* code removed for space */
}

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
}

void loop() {
  for(int c = 0;c < 4;c++) {
    short val = readADC(c);
    Serial.print(c);
    Serial.print(" ");
    float volt = (float)val/32767.0*4.096;
    Serial.println(volt);
    delay(100);
  }
}
```

Part 2 of 2 Code. Include function in previous page where shown.

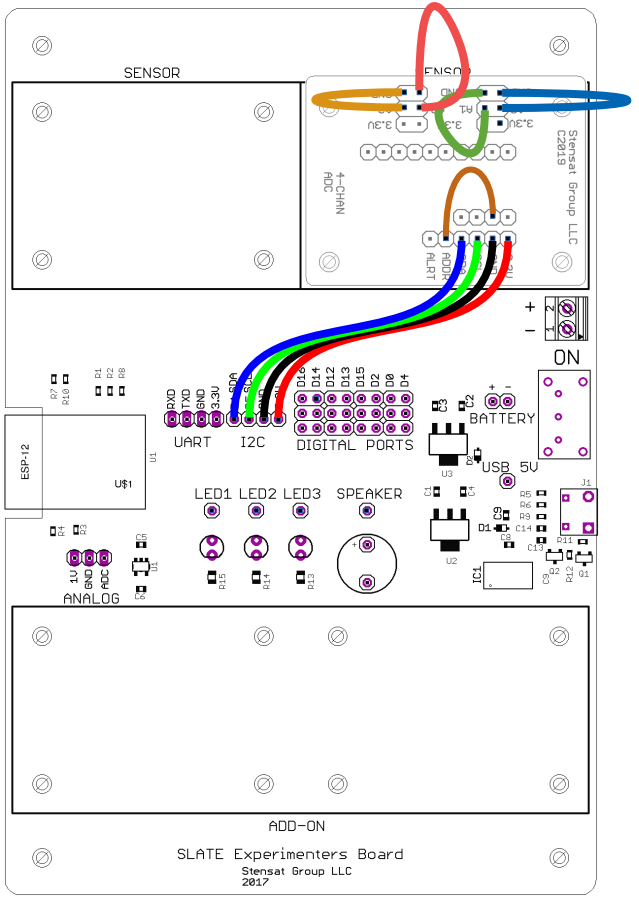
ADC Operation

- As the **for()** loop cycles through each input, the measurement is made for the input selected by variable **c**.
- A 16-bit value is returned into the short variable **val**. Since the input could be negative, the short variable val is signed.
- The ADC value is converted to a voltage. The 4.096 is the range that the ADC is set to. The 32767 is the ADC range which is -32767 to 32767.

```
#include <Wire.h>

#define ADC 0x48

short readADC(unsigned char addr) {
 /* code removed for space */
}

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
}

void loop() {
  for(int c = 0;c < 4;c++) {
    short val = readADC(c);
    Serial.print(c);
    Serial.print(" ");
    float volt = (float)val/32767.0*4.096;
    Serial.println(volt);
    delay(100);
  }
}
```

ADC Test

- Upload the program.
- Connect all four ADC inputs (A0,A1,A2,A3) to the GND pins on the ADC module.
- Open the Serial Monitor window and check the results of the ADC conversion. All the values should be 0 or very close to it.
- Move one of the ADC inputs from GND to 3.3V connection on the SLATE.
- Check the measurement on the serial monitor window. It should be reading near 3.3 volts.
- Test each input.



Using the ADC

- Install the Ultrasonic sensor board that includes the thermistor and light sensor. Connect the sensors to the first two channels on the ADC module. 1V on the sensor board will connect to 3.3V. ADC connects to the A0 and A1. Put the light sensor on A0 and the thermistor on A1.
- GND pins on the sensor board connect to GND on the ADC module.
- Monitor the voltages and play with the sensors to make them change. This is how analog sensors can be connected to the ADC board.