Mayhew Labs

Extended ADC Shield
User Manual

Table of Contents:

Introduction	1
Hardware Description	1
Pin Descriptions	2
Setting the SPI communication level	2
Setting User Defined pin usage	2
Freeing Up Pin 9 (BUSY)	2
Installing Input Filtering Capacitors	3
Extended ADC Shield Schematic	3
ExtendedADCShield Library for Arduino	4

Introduction

The Extended ADC Shield gives your Arduino the ability to measure extended voltage ranges with higher resolution and faster speed than the built in analog-to-digital converters (ADCs). Each of the eight input channels can be configured as a single-ended (ground referenced) or differential input and can be set for an input range of 0V to 5V, 0V to 10V, ±5V and ±10V. Analog signals are connected to the shield using robust screw-terminal connections and are safe for input voltages up to ±25V. The shield is available in 12-bit, 14-bit, and 16-bit versions and has a maximum sample rate of 100ksps regardless of which version is used. The shield interfaces with any Arduino hardware platform over SPI along with a few additional control signals and has selectable pin usage for these signals. The shield can be set to communicate at 5V or 3.3V levels, making it safe for 3.3V tolerant Arduinos.

This manual will help you get up and running and also explain how the shield works. If you want to skip the details and get down to business, install the library and run the demo examples (see the **ExtendedADCShield Library for Arduino** section).

Hardware Description

The Extended ADC Shield uses the Linear Technology LTC185x series of 8-channel analog-to-digital converter ICs, which include the LTC1857 (12-bit), LTC1858 (14-bit), and LTC1859 (16-bit). These are unique ADCs because each channel may be software-configured for different input ranges, and be programmed for single-ended or differential inputs (and combinations of both), while being powered from a 5V supply. The IC's are also 5V and 3.3V communication level compatible, making them ideal ADCs for all Arduino hardware platforms.

The shield is set up to have flexible configuration for what Arduino pins it occupies. It uses the Arduino's ICSP header for SPI communication so that it can communicate with all Arduino variants. The shield uses isolated analog and digital ground planes so that noise from switching communication lines can be kept to a minimum on the analog inputs. It also contains positions for 0603-sized capacitors that the user can populate to help filter out incoming noise both on single-ended and differential inputs.

Contact: CustomerService@mayhewlabs.com

Pin Descriptions

Name	Pin	Description
CONVST	IO8 or user defined	Conversion start: Pull this pin high for at least 40ns to start an analog to digital conversion on the last configured channel.
RD (SS)	IO10 or user defined	Read (Slave select): Pull this pin low when you want to communicate to the shield. This pin doesn't need to be low to initiate a conversion via the CONVST pin.
SCK	ICSP pin 3 or user defined	SPI clock: Data clock to the shield from the Arduino. Data is clocked in or out on the rising edge. On Uno or Duemilanove this is connected to IO13. On Mega1280 or Mega2560 this is connected to IO52. On Leonardo or Due it is not connected to any IO pins.
SDI	ICSP pin 4 or user defined	SPI data input: Data input to the shield from the Arduino. On Uno or Duemilanove this is connected to IO11. On Mega1280 or Mega2560 this is connected to IO51. On Leonardo or Due it is not connected to any IO pins.
SDO	ICSP pin 1 or user defined	SPI data output: Data output from the shield to the Arduino. On Uno or Duemilanove this is connected to IO12. On Mega1280 or Mega2560 this is connected to IO50. On Leonardo or Due it is not connected to any IO pins.
BUSY	IO9 or user defined	Busy: Pulled low when an analog to digital conversion is in progress (immediately after the CONVST pin is pulled high). The maximum conversion time is 5us. This pin can be freed up so that the shield no longer uses it. See details below.

Setting the SPI communication level

The shield can communicate over SPI at both 5V and 3.3V levels. The input pins (CONVST, RD, SCK, SDI) have a high level input threshold of 2.4V and are therefore compatible with all Arduino variants. The SPI COM LEVEL solder jumper on the back of the shield sets the voltage swing of the shield's output pins SDO and BUSY. If you are using a 3.3V tolerant Arduino variant like the Due, set this solder jumper as shown on the back of the board using a small bead of solder that connects the center pad to the right pad. Please note that although SPI communication at 3.3V is possible, the shield must be powered from a 5V supply.

Setting User Defined pin usage

If you wish to use different pins than those listed above for SPI communication, for example a different digital pin for RD (slave select), you can do so using solder jumpers SJ1 through SJ6 and P5 on the shield. If you remove the solder bead from these solder jumpers, you can run wires from any Arduino pin or external control directly to the ADC IC's inputs using the connections at P5. This is useful if you have multiple devices communicating on the same SPI bus and you need to have a different slave select line controlling this shield. A special initialization function exists in the ExtendedADCShield library for changing which Arduino pin is used for CONVST, RD, and BUSY. The jumpers are also useful if you want to communicate with the shield using software SPI (bit banging). A helpful tutorial on SPI bit banging can be found here. Bit banging mode is not supported by the ExtendedADCShield library, so read the LTC175x datasheet for more information on manually sending commands to the shield.

Freeing Up Pin 9 (BUSY)

The BUSY output line from the shield is pulled low while the onboard ADC is performing a conversion, immediately after the CONVST pin is pulled high by the Arduino. The maximum conversion time for the ADC is 5us, which is shorter than the time it takes an Arduino Uno to toggle the CONVST line high and then back low. Therefore, when a

conversion is triggered by the Arduino, the conversion is complete before the BUSY line can be checked to see whether the ADC conversion is finished. This means that the BUSY line is not needed, since the conversion result will always be ready by the time the Arduino asks for it. This is the case for 16 MHz Arduinos, but not true for those running at a faster clock speed, like the Due. Therefore you cannot free up this pin on Arduinos running faster than 16 MHz. The ExtendedADCShield library was written to cover all Arduinos and therefore has a wait loop to check when the BUSY line goes high.

If you wish to free up Pin 9 so that is not used for the BUSY signal, you must do two things: 1) remove the solder jumper from SJ6 on the back of the shield and 2) comment out the following lines from the file ExtendedADCShield.cpp in the library:

`pinMode(_BUSY, INPUT);` (in two places)

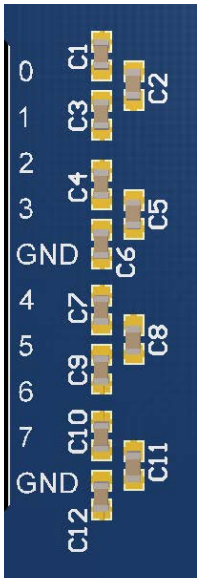
`while(digitalRead(_BUSY)==LOW);`

Installing Input Filtering Capacitors

If you need to smooth out noise on the incoming analog signals, you may populate capacitor positions C1 through C12 on the shield with 0603-sized capacitors. The LTC185x datasheet recommends a 3000pF NP0-type capacitor in order to minimize wideband noise. The capacitors that should be placed depend on whether you are using single-ended inputs or differential inputs. See the table below for which capacitor positions to use and the image for capacitor orientation:

Single Ended Input Channel	Capacitor Position
0	C1
1	C3
2	C4
3	C6
4	C7
5	C9
6	C10
7	C12

Differential Input Channel	Capacitor Position
0-1	C2
2-3	C5
4-5	C8
6-7	C11



Extended ADC Shield Schematic

The schematic for the shield can be found here.

ExtendedADCShield Library for Arduino
The ExtendedADCShield library was created to make initializing and controlling the shield easier. It hides the SPI message setup and communication along with toggling of the extra control lines.

How to install the library:
1. Quit the Arduino application if it's running
2. Download ExtendedADCShield.zip and unzip it.
3. Move the ExtendedADCShield folder to your Arduino libraries folder:
    a. Windows: My Documents\Arduino\libraries
    b. Mac: Documents/Arduino/libraries
    c. Linux: the "libraries" folder in your sketchbook
4. Restart the Arduino application

How to Use the library:
1. Start the Arduino application
2. Begin with a demo:
    a. File > Examples > ExtendedADCShield > Examples > (choose one of the following demos)
        i. ExtendedADCShieldDemo: an example of how to use the shield for a variety of analog inputs, communicating with the shield over the Arduino's built in SPI
        ii. ExtendedADCShield100KHz: an example of how to achieve 100kHz sampling rate on a single input
3. Add the ExtendedADCShield library to an existing sketch:
    a. Sketch > Import Library > ExtendedADCShield
    b. See the function reference below

ExtendedADCShield Library Function Reference:

`ExtendedADCShield(byte number_bits)`

Description: Use this to create an instance of the ExtendedADCShield library
Parameters: - **number_bits**: the number of bits that the analog to digital converter IC on the shield is capable of. Use 12 for the LTC1857, 14 for the LTC1858, or 16 for the LTC1859
Return Value: None

`ExtendedADCShield(byte CONVST, byte RD, byte BUSY, byte number_bits)`

Description: Create an instance of the ExtendedADCShield library with user-defined pins
Parameters: - **CONVST**: the Arduino pin number connected to the shield's CONVST input - **RD**: the Arduino pin number connected to the shield's RD input - **BUSY**: the Arduino pin number connected to the shield's BUSY output - **number_bits**: the number of bits that the analog to digital converter IC on the shield is capable of. Use 12 for the LTC1857, 14 for the LTC1858, or 16 for the LTC1859
Return Value: None

`float analogReadConfigNext(byte channel, byte sgl_diff, byte uni_bipolar, byte range)`

Description: Reads the last configured analog input, sets up the configuration of the next analog input. Note that the inputs to this function are used to set up the **next** analog input to read, while the output is the value of the **last** configured analog input at the time this function is called. See the example below for reference.

Parameters:
- **channel**: the channel (1 to 7) that you want to configure next. For differential analog inputs, set this to be the positive input channel of the differential pair. The differential pairs are 0-1, 2-3, 4-5, and 6-7. For example, if you want the value of input 7 minus input 6, set 'channel' to be 7.
- **sgl_diff**: set whether the next configured channel should be single ended (a single input pin) or differential (one input minus another input) using one of the following values:
  - **SINGLE_ENDED**
  - **DIFFERENTIAL**
- **uni_bipolar**: set whether the next channel to configure should be unipolar (0 to 5V and 0 to 10V) or bipolar (-5 to 5V and -10 to 10V) using one the following values:
  - **UNIPOLAR**
  - **BIPOLAR**
- **range**: set whether the next channel to configure should be 5V (and -5 to 5V) or 10V (and -10 to 10V) using one of the following values:
  - **RANGE5V**
  - **RANGE10V**

Return Value: the voltage of the last configured analog input at the time this function is called.

`analogReadConfigNext` example:

```
//Read input 0, set up input 1 as single-ended unipoloar 0 to 5V
//Assumes that input 0 was configured previously
ch0 = extendedADCShield.analogReadConfigNext(1, SINGLE_ENDED, UNIPOLAR, RANGE5V);
```