About the ADS1x1x library

The ads1x1x library can handle six different but compatible ADC chips: ADS1013, ADS1014, ADS1015, ADS1113, ADS1114, ADS1115. The difference between them is the number of inputs and the conversion resolution. The ADS101x are 12-bits converters, the ADS111x have a 16-bit resolution. The library can handle all of them, but you must specify the chip during initialisation. Since it is possible to configure the I²C address of the ADC chip, you must also tell the library which address to use.

An effort has been made to make the library as user-friendly as possible and functions have been provided to control every possible parameter of the chip. Because the ADC1x1x has only one configuration register holding all the parameters, the bit-fields have been declared as enumerated types (`enum`), not as macros (`#define`). This allows for compile-time error checking of these fields, making it difficult to supply completely wrong values for a parameter that may accidentally overwrite another parameter. But, for those that like to have full control, three low-level functions have been made public as well.

To allow for systems with more than one ADC chip (up to four since the ADS1x1x only knows four I²C addresses) the configuration of the chip is stored in a user-provided data structure. For every ADS1x1x in the system such a structure is needed. This structure has to be passed to every call of the library so that the library knows which configuration to use. This is a sort of C way of doing object oriented (C++) things. If you want you can push this paradigm even further and also apply it to the I²C driver so that the library can use multiple I²C busses.

The ADC read function uses the chip configuration that is stored in the user-provided data structure and consecutive reads will use the same chip configuration.

The library is written so that it can be used in C and C++ projects.

All the data types used are either defined by the library itself or borrowed from stdint.h to remove any ambiguity regarding their sizes in bits and their signedness. By doing things this way the library can be used easily without modifications on most platforms. If your compiler does not have stdint.h you must define `uint8_t` (8-bit unsigned integer), `int16_t` (16-bit signed integer) and `uint16_t` (16-bit unsigned integer) yourself.

The ADS1x1x communicates over an I²C bus, but the library does not implement the low-level I²C driver, only so-called "stubs" are available. These are like pins on a chip that you must connect to something to make it all work. The user is responsible for providing the driver and to connect the stubs. The library has five stubs that the user must connect to his/her own I²C driver:

- `uint8_t ADS1x1x_i2c_start_write(uint8_t i2c_address)`

- `uint8_t ADS1x1x_i2c_write(uint8_t x)`
- `uint8_t ADS1x1x_i2c_start_read(uint8_t i2c_address, uint16_t bytes_to_read)`
- `uint8_t ADS1x1x_i2c_read(void)`
- `uint8_t ADS1x1x_i2c_stop(void)`

These functions are needed to initiate a write or read transaction on the I²C bus, to read and write bytes and to stop the I²C bus.

For your convenience a simple software I²C driver is included in the download that implements a bit-banged I²C bus on almost any pair of general purpose input/output (GPIO) pins. The software I²C library needs also a number of user-provided stubs to talk to the GPIO pins.

A simple Arduino example is included in the download that shows how to use both the ADC and the software I²C drivers. The example will read every possible ADC input (combination) and output the results converted to volts on the serial port.

CPV, 6/6/2014

Related Elektor projects

- 130485 Arduino 16-bit Datalogger, https://www.elektormagazine.com/130485
- 140169-1, ADS1115 eBoB, https://www.elektormagazine.com/140169
- 140169-2, Filter for ADS1115 eBoB, https://www.elektormagazine.com/140409
- 140409, AC/DC Power Meter, https://www.elektormagazine.com/140409