Interfacing with the AD7124-4 using Simblee Microcontroller

Nicholas P. Newman

August 5, 2017

Abstract

Signal processing in biological sensing relies on robust systems with large dynamic ranges. The asthmaserver project needs to convert signals with a wide range of amplitudes from different respiratory phenomena into high-resolution data. The use of a PGA (Programmable Gain Amplifier) to adjust the gain of the front-end system appropriately allows for rapid adjustments to measurements, ensuring the highest possible data resolution. To implement this, a preliminary study with the ADS1015 was conducted to verify and explore Simblee compatibility with combined ADC-PGA systems. Then, different higher-level ADCs were evaluated based on several characteristics to determine which would used in the next iteration of sensor, resulting in the selection of Analog Devices' AD7124-4 IC. The next and most substantive part of the project involved writing the interface between the Simblee microcontroller and the new AD7124 chip, designing a high-level C++ library for the same, and packaging the results in a user-friendly, robust git repository. The library includes the relevant configuration, AGC (automatic gain control), power-saving, and other useful functions for foreseeable sensor applications.

Chapter 1

Introduction

1.1 Problem Description

The desired result of this project is a low-power, low-footprint ADC with on-board PGA capable of sampling respiratory action from an analog front-end at between 10 and 16ksps. The IC should consume less than 1mA of current (compared to 150uA consumed by the ADS1015) and the footprint should be comparable to that of the ADS1015, or 5mm x 3mm.

1.2 Assigned Task and Deliverables

1. *Programmable Gain Amplifier & ADC*: We are procuring a combined ADC-PGA IC for you to get started. This is the ADS1015, a low-power 12-bit ADC with an integrated PGA. Adafruit has a whole page for this particular evaluation board which is provided below. With this, you can easily get started learning how to use the ADC and the PGA for sampling the sensor signal. You will be using the Simblee micro-controller module for talking to this IC. The suggested steps for approaching this are:
   (a) Familiarize yourself with I2C since this is the interface for the IC.
   (b) Get started with the Simblee module. Consult with Liam Feeney.
   (c) Next, attempt to continuously sample 2-channels simultaneously using the ADS1015.
   (d) Afterwards, attempt to control PGA for altering gain of system.
   (e) Implement a full AGC system using ADC & PGA for given sensor.
2. *New ADC-PGA IC*: The ADC-PGA IC you used above can only sample at a max rate of 3.3 Ksps. For cough and wheeze sounds, we want to attain sampling rates of about 10 kHz to 16 kHz. In this subtask, you will search the web (Digikey, Adafruit, etc.) for a new ADC with a PGA and use that for implementing the AGC. The steps should be as follows:

(a) Identify a new ADC with an PGA to replace the ADS1015. Specs should be:

i. Sampling Rate: 10 ksps to 16 ksps (it might be easier to get 1 or 2 Msps ICs)

ii. Low power: We want an IC that is sub mA's in current consumption. ADS1015 is 150uA.

iii. Small footprint: The IC need to be small - at least not much larger than the ADS1015

(b) Purchase a breakout board for the new ADC-PGA

(c) Implement a fast (10-16 kHz) dual-channel sampling using new ADC-PGA

(d) Port AGC implementation from one above to new PGA

3. *Interleaving ADCs*: Should you be unable to find a suitable ADC in (2) above or in any other invent where (2) is no longer feasible, consider using dual channel interleaving to achieve a 2x sampling rate. The ADS1015 has 4 channels and since we are only using 2 analog lines, we could take advantage of the extra ADCs to improve our sampling period. Even if (2) works, you should consider taking on this subtask if there is time. In this subtask:

(a) Research and identify a suitable technique for increasing sampling rates of ADCs (e.g. interleave)

(b) Implement technique discovered above for this application

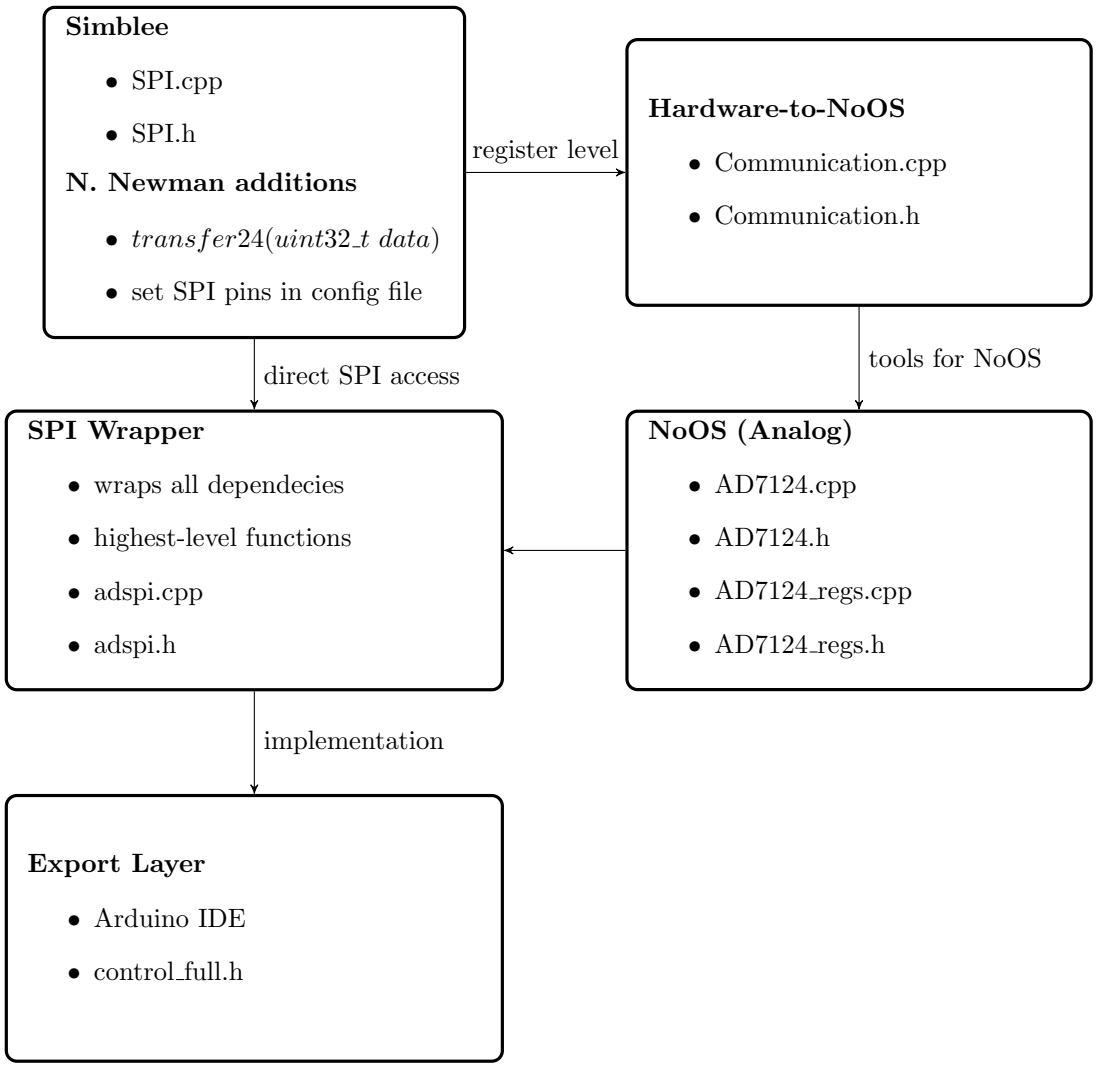
Chapter 2

Implemented System and Solution

2.1 Overview

The implemented system consists of the AD7124-4 analog-to-digital converter, the Simblee microcontroller, and the libraries written in C++ that are used to operate both. The dependency structure of the various libraries can be found (label here). Analog Devices provides what is called "No-OS" software that is designed to work with any microcontroller. To implement this software, all one has to do is write the appropriate functions in the sparse "Communication" library. Various functions' arguments and outputs are specified, and the user has to design them using functions from the microcontroller's SPI library. In this case, the SPI library is contained in SPI.cpp and SPI.h. The SPI library provided by Simblee lacked some functionality, so that was provided by N. Newman. The functions designed in the Communication library are used by the NoOS software (AD7124.cpp etc.), which contains the basic functions required to operate the ADC, such as setups, register reads/writes, and error checking. All of this is contained by the SPI wrapper written by N. Newman, which combines register-level SPI, NoOS, and Arduino functions to provide the user high-level, easily understandable functions for operating the ADC. Finally, the Arduino 'sketch' file control_full.ino is the top-level file that is compiled via the Arduino IDE and uploaded to the Simblee. Ideally, the user should be able to configure all the various states of the ADC using only the adspi library and an Arduino sketch. More intricate operations may require exploration and modification of lower-level libraries.
Other resources provided in this project include an Excel spreadsheet with macros that simplify the calculation of register values in concert with the register value datasheet, a setup and user guide available on the [github name here], and several 'sketch' files appropriate for diagnostics and testing.

2.2 Diagrams and Datapath



2.3 Relevant Preexisting Systems

Chapter 3

Libraries and other Deliverables

3.1 Simblee SPI Library

The Simblee SPI library is provided by the manufacturers of Simblee and contains much of the same functions you get in the standard Arduino communications libraries. It gives some insight into how the register level operations of the Simblee microcontroller function, and provides the most direct interface with the AD7124. Although there is no reference documentation regarding this library, it is possible to intuit the meaning and operation of most of the functions and how they might be modified or adapted to suit the specific requirements of the AD7124 SPI communications. The SPI library does not come with a 24-bit transfer function, so a new one was written by N. Newman and can be used in read/writes with the largest (3-byte) registers on the IC. In some applications, several 8-bit transfers works as well, depending on the required action of the CS pin. The default SPI1 and SPI2 MOSI, MISO, SCLK, and CS pins can be changed in the variant.h file or modified manually using the set_foo() functions.

3.1.1 SPI.begin() Changes

```
void SPIClass::begin()
{
    pinMode(pinSCK, OUTPUT);
    pinMode(pinMOSI, OUTPUT);
    pinMode(pinMISO, INPUT);

    spi->PSELSCK = pinSCK;
    spi->PSELMOSI = pinMOSI;
    spi->PSELMISO = pinMISO;
```

```
    // Default speed set to 4Mhz
    setFrequency(4000);
    setDataMode(SPI_MODE3);
    setBitOrder(MSBFIRST);

    spi->EVENTS_READY = 0;

    spi->ENABLE = (SPI_ENABLE_ENABLE_Enabled << SPI_ENABLE_ENABLE_Pos);
}
```

The AD7124 uses SPI mode 3, MSB first convention, and runs most effectively with a 4MHz SCLK.

3.1.2 SPI.transfer24() Addition

```
uint32_t SPIClass::transfer24(uint32_t _data)
{
    uint8_t highByte, midByte, lowByte;

    if(spi->CONFIG && 0x01)  //LSB first
    {
        spi->EVENTS_READY = 0;          //clear ready event
        // SPI is triple buffered
        spi->TXD = _data & 0xFF;       //lower 8 bits
        spi->TXD = (_data >> 8) & 0xFF;      // middle 8 bits
        spi->TXD = (_data >> 16) & 0xFF;        // upper 8 bits
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;          //clear ready event
        lowByte = spi->RXD;
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;          //clear ready event
        midByte = spi->RXD;
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;          //clear ready event
        highByte = spi->RXD;
    }
    else                                //MSB first
    {
        spi->EVENTS_READY = 0;          //clear ready event
        // SPI is trpible buffered
        spi->TXD = (_data >> 16);         // upper 8 bits
        spi->TXD = (_data >> 8);
        spi->TXD = _data & 0xFF;       //lower 8 bits
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;          //clear ready event
        highByte = spi->RXD;
```

```
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;           //clear ready event
        midByte = spi->RXD;
        while (spi->EVENTS_READY == 0) ; // wait for ready event
        spi->EVENTS_READY = 0;           //clear ready event
        lowByte = spi->RXD;
    }

  uint32_t data = (highByte << 16) | (midByte << 8) | lowByte;

  return data;
}
```

3.2 Communication Library

The Communication library is a shell provided by Analog Devices which the end user must fill with the appropriate functions to be used by the AD7124 library. By default, the SPI read and write functions have been written for non-continuous-read operation, meaning the CS pin dictates byte transfer operations. Continuous-read operation requires a specific chain of events and timings that are implemented in the adspi library. Because roughly half of the on-chip registers on the AD7124 are 24-bit registers, and no standard uint24_t type exists, 24-bit values are treated as uint32_t.

3.3 AD7124 Library

The AD7124 Library is written by Analog Devices and is part of their NoOS software designed to be usable by any microcontroller (granted you can suitably design the Communications Library). This set of programs includes somewhat bulky functions that implement basic error-checking (although only one type) as well as a setup function that configures all the registers serially. The register definitions (base address, start-up value, size) are all contained in the AD7124.h file, which also contains a structure for maintaining a description of the state of the AD7124. The AD7124.cpp file contains all the functions, which compile normally out of the package except for one compiler issue: the Arduino GNU compiler is unhappy about attempts to increment enums ipso facto, so a small change was implemented by N. Newman in order to circumvent it.

```
for(regNr = AD7124_Status; (regNr < AD7124_Offset_0) && !(ret < 0);
        regNr++)
```

must change to

```
for(regInt = AD7124_Status; (regInt < AD7124_Offset_0)
                            && !(ret < 0); regInt ++){
                ad7124_registers regNr = static_cast<ad7124_registers>(regInt);
```

3.4 adspi Library

The adspi library is designed by N. Newman as a wrapper for all the previous libraries. It contains the highest-level functions and resources intended to be easily accessible to engineers without requiring in-depth knowledge of the system itself or further modification to dependencies. Full details of the dependency chain can be found at [figure]. The adspi library calls on two libraries in particular, the AD7124 as a resource for most general register read/writes as well as start-up and operation in all but continuous-read mode. The SPI library is also used for specific register operations as well as all data reads during continuous-read operation. It also has robust testing tools and is compatible with more error checking than AD7124 (although this may change in V_next). Furthermore, the adspi library includes the RunAGC() and SetPGA() functions that manage the gain of the Programmable Gain Amplifier.

3.5 Usage and Results

For a detailed guide on installation, setup, and implementing the hardware and software associated with the AD7124, see the *AD7124-Simblee User Guide*, available on the GitHub repository.

Chapter 4

Conclusions

4.1 Lessons and Challenges

The experience that prepared me best for this project was likely the ENGS62 class taught by Prof. Stephen Taylor. In Microprocessors in Engineered Systems we ran a microcontroller using register-level functions and assembly. It was somewhat intimidating the number of configuration and accessory registers that were present on the chip, and the documentation was comprehensive and therefore intimidating and hard to navigate. It has become clear to me over the course of this project that comprehensive, detailed documentation is in all ways preferable to a lack of the same, regardless of how intimidating it might be. Much of the time spent troubleshooting this problem resulted from a lack of documentation regarding the Simblee microcontroller. For most purposes, the documentation is certainly sufficient. However, running a precise timer, modifying the SPI library, and creating a fast, low-jitter external clock all required repeated parsing of obscure forums and a substantial amount of educated guesswork. As is always the case, maintaining good documentation of the project itself has proven to be an invaluable practice. Taking the time to accurately record successes and ideas will inevitably benefit future work.
The biggest challenge besides the obscure register-level interactions was certainly developing the C++ libraries. In all my engineering classes thus far we have developed code and libraries exclusively in C. Working in C++ has numerous advantages that were rapidly made clear. However, creating interactions between different source files, ensuring the rigor of dependency chains, and eliminating unresolved bugs proved a substantial task. Style guides provided a valuable reference for how best to organize and structure everything.

4.2 Outstanding Work and Moving Forward

1. Verify operation of 10ksps min continuous read

2. Figure out why power level won't increase
3. Finalize state of libraries, document
4. Write installation and user guide
5. Verify operation of AGC with respiratory action
6. Begin interfacing AD7124 library with Liam's data management