SHARC DSP Programming – Report 1

(a work in progress)

Ageu Nunes
Justin Sconza
Mark Wudtke

Spring 2017

ECE395: Advanced Digital Projects Lab
University of Illinois at Urbana-Champaign

Deciphering the SHARC Hardware Reference Manual

This document is long, and for anyone just beginning, this makes for an almost insurmountable obstacle. The most useful advice we came up with while working with this processor is to use this truly as a reference manual – that is to say, it is useless to try and read it like a book. When learning new techniques and new peripherals on the chip, it was most beneficial to look up the relevant section, skim the details, find example code, and keep open the register references.

Deciphering the On-board DAC User Guide

The user guide for the DAC is similar in complexity to the SHARC, but overall, much more readable. Another point worth noting is that some of the quoted abilities of this DAC are too conservative. We found that trying to design around the quoted specs was not as important as the document might lead one to believe.

C Code

In the following discussion, we will briefly go over the role of each function called in the main.

initSPDIF

The function called initSPDIF simply initializes the SHARC SPDIF receiver.

initSRU

The next function, called initSRU initializes the signal routing unit inside the SHARC. This is most easily understood as a patch bay. The signal routing unit is the only way signals between different peripherals can be connected. In our case, there are basically two types of SRU connections being made. The first is for allowing connection of external signals to internal peripherals. The syntax is as follows.

```
SRU(LOW, DAI_PB12_I);
SRU(LOW, PBEN12_I);
SRU(DAI_PB12_O, DIR_I);
```

The second type of SRU connection we used is for connecting one peripheral to another. The syntax for this connection is as follows.

```
SRU(DIR_DAT_O, SPORT0_DA_I);
```

initSPI

This function is written in accordance with the hardware reference manual. In our project, SPI is used to initialize the DAC.

configureAK4396Register

The purpose of this function is to initialize the DAC. The arguments are the address of the specific control register to configure, and the data containing the desired configuration settings. The function builds the initialization word according to the specifications in the DAC user manual, and the word is transmitted from the SHARC to the DAC over SPI.

initDMA

This function initializes the DMAs for the two SPORT channels necessary to receive and transmit audio data. Here we set up the TCBs for our chained DMA operations according to the way DMA is implemented in the SHARC. Most of the necessary information was gleaned from the hardware reference manual and from Analog Devices example code.

delay

This is a function which has a for loop whose only purpose is to insert blank instruction cycles wherever desire.

clearDAIpins

This is simply good practice according to the manual and example code that we found.

processSamples

The basic structure of this function revolves around a while loop that processes all “available” samples. This is accomplished through the argument to the while loop, which will be satisfied whenever the dsp pointer is equal to the receive buffer pointer used in the DMA operation. This is to ensure that the dsp pointer never exceeds the receive buffer pointer, so that we are never trying to process samples that have not been received. The “bones” of this function can be used for any dsp – in this project, it is used for a simple slapback delay.

Overall, the code we borrowed or were inspired by from Mitch, we ultimately changed so much so that it kind of became our own. At the end of this document, we provide both our full code which incorporates our work as well as Mitch’s and then we also provide the code we wrote solely.

Pictures

In the following figure, we present a sawtooth input filtered down to a single sinusoid.

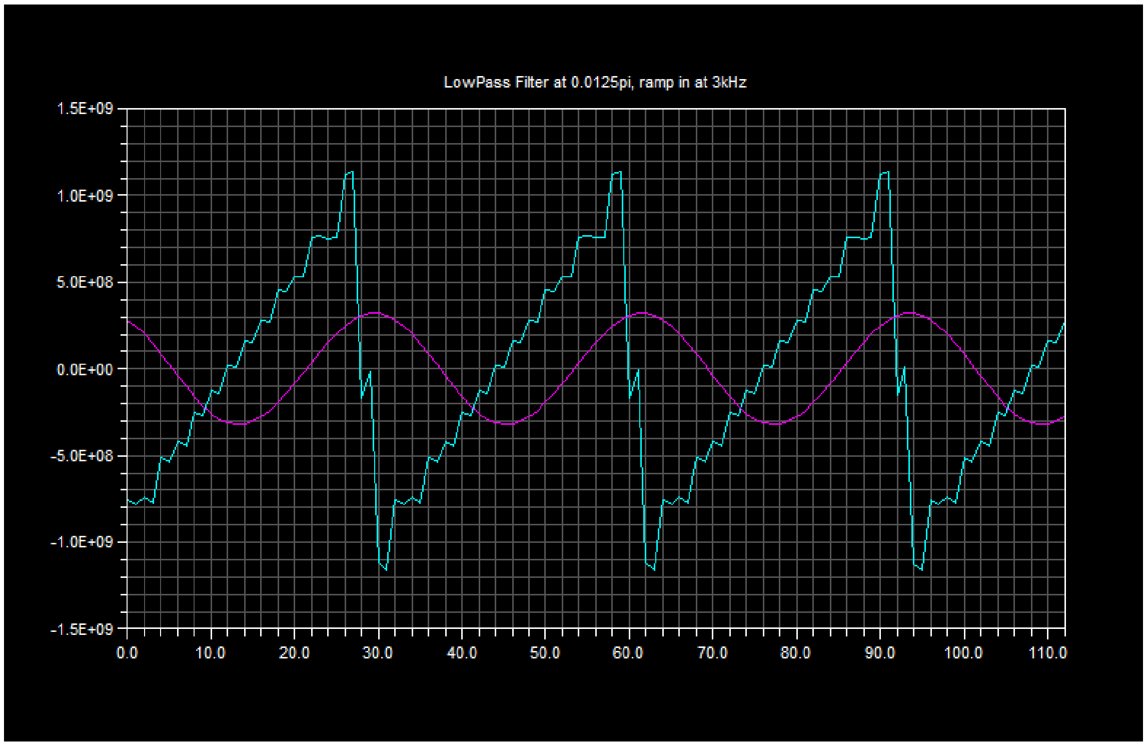


Figure 1: Filtered Sawtooth

The above figure is of a plot generated in VisualDSP++. The input buffer is in cyan and the output is in magenta.

In the next figure, we present the MATLAB frequency response for our FIR filtered implemented above.

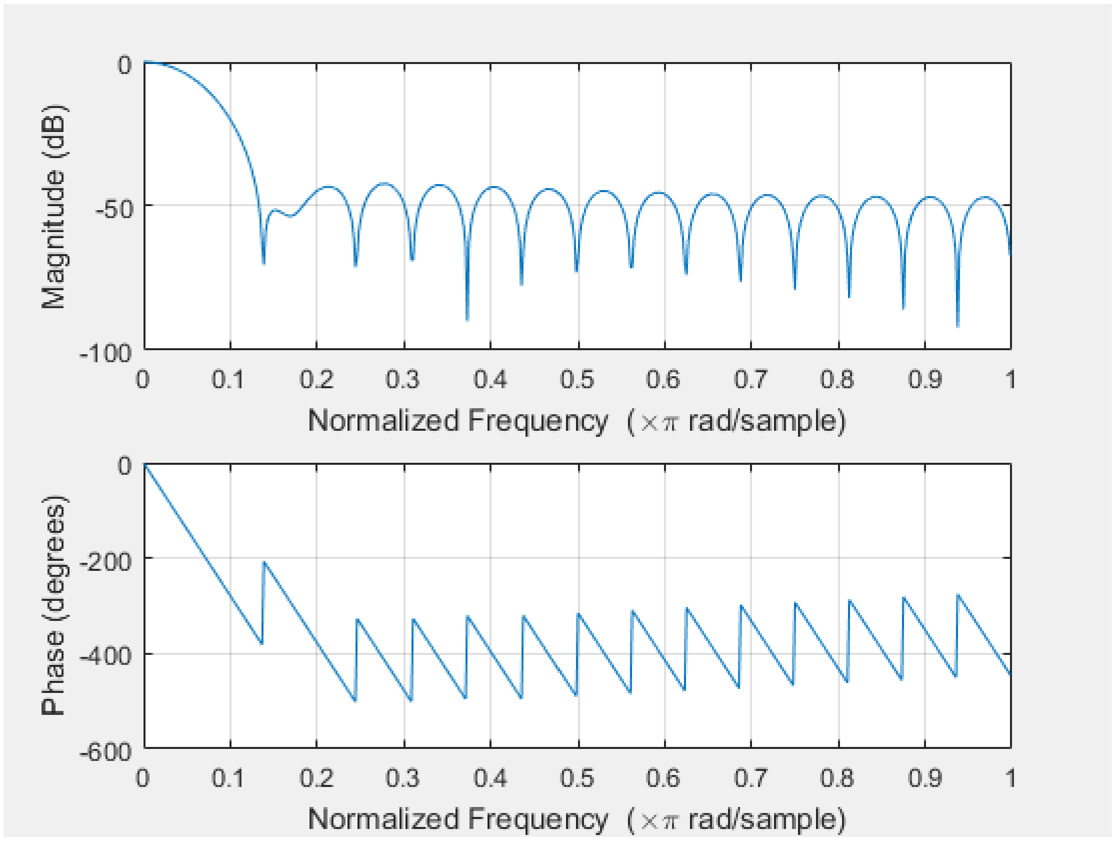


Figure 2: FIR Frequency Response

Biphase Schmitt Trigger Technique

The SPDIF signal coming from the Samson ADC is a biphase stream. We found that the signal was not clean enough for use by the SHARC. This is actually quite a common problem, with a conventional solution. We implemented the following circuit.

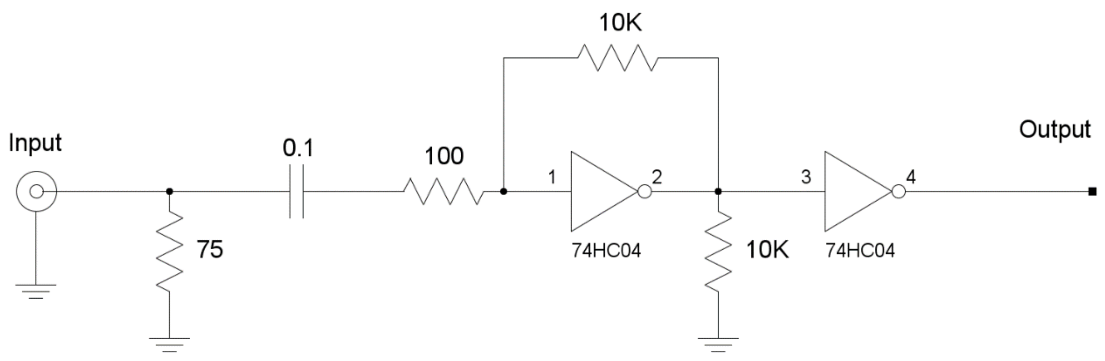
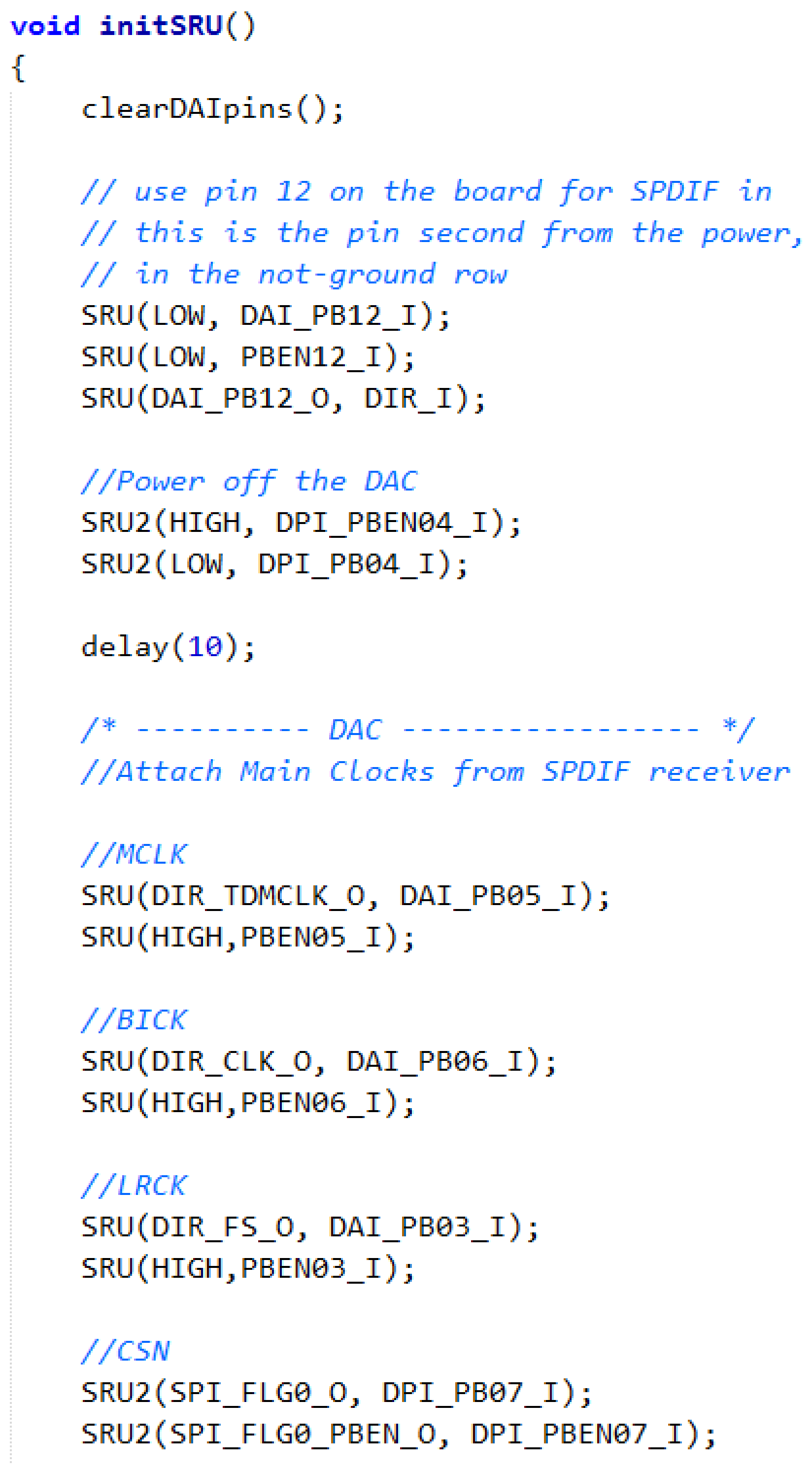


Figure 3: FIR Frequency Response

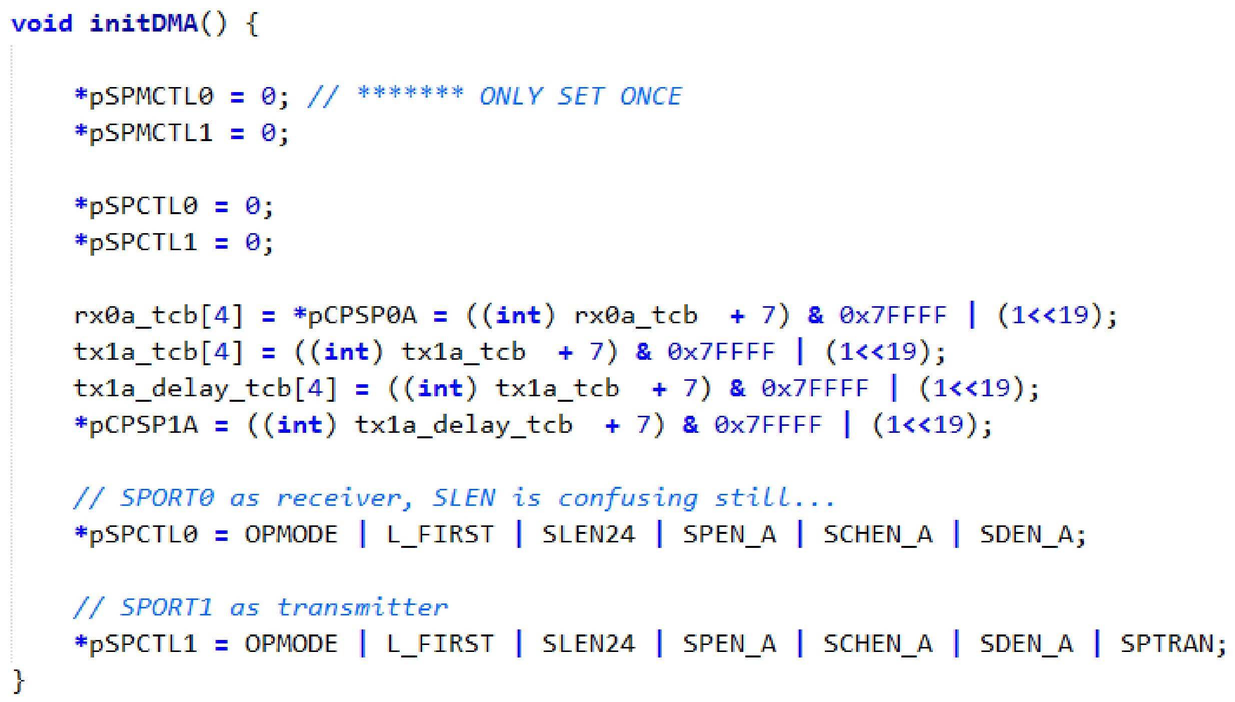
The above circuit basically flips the signal and then slams the signal up to the positive rail and down to the negative rail. The second inverter gets the signal back right-side up. The end result is something the SHARC can properly interpret.

Appendix: Our Original Code



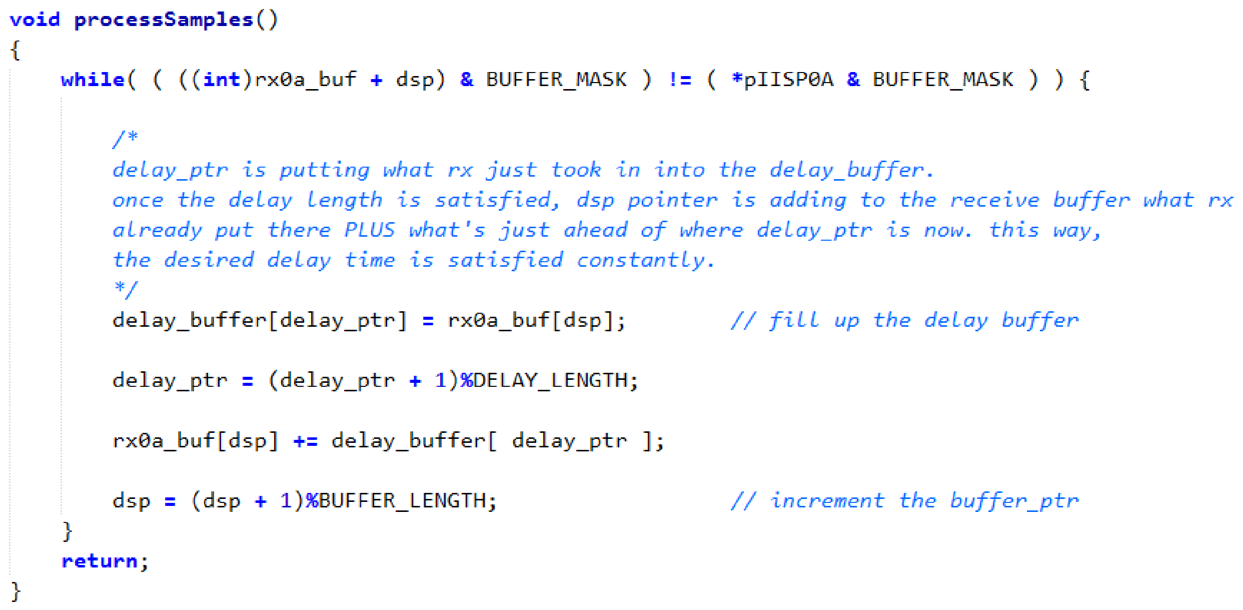
```c
void initSRU()
{
    clearDAIpins();

    // use pin 12 on the board for SPDIF in
    // this is the pin second from the power,
    // in the not-ground row
    SRU(LOW, DAI_PB12_I);
    SRU(LOW, PBEN12_I);
    SRU(DAI_PB12_O, DIR_I);

    //Power off the DAC
    SRU2(HIGH, DPI_PBEN04_I);
    SRU2(LOW, DPI_PB04_I);

    delay(10);

    /* ---------- DAC ----------------- */
    //Attach Main Clocks from SPDIF receiver

    //MCLK
    SRU(DIR_TDMCLK_O, DAI_PB05_I);
    SRU(HIGH,PBEN05_I);

    //BICK
    SRU(DIR_CLK_O, DAI_PB06_I);
    SRU(HIGH,PBEN06_I);

    //LRCK
    SRU(DIR_FS_O, DAI_PB03_I);
    SRU(HIGH,PBEN03_I);

    //CSN
    SRU2(SPI_FLG0_O, DPI_PB07_I);
    SRU2(SPI_FLG0_PBEN_O, DPI_PBEN07_I);
```

Note, the rest of initSRU won’t fit, however, it’s all our own code and can be found in the attached c file.



```
void initDMA() {

    *pSPMCTL0 = 0; // ******* ONLY SET ONCE
    *pSPMCTL1 = 0;

    *pSPCTL0 = 0;
    *pSPCTL1 = 0;

    rx0a_tcb[4] = *pCPSP0A = ((int) rx0a_tcb  + 7) & 0x7FFFF | (1<<19);
    tx1a_tcb[4] = ((int) tx1a_tcb  + 7) & 0x7FFFF | (1<<19);
    tx1a_delay_tcb[4] = ((int) tx1a_tcb  + 7) & 0x7FFFF | (1<<19);
    *pCPSP1A = ((int) tx1a_delay_tcb  + 7) & 0x7FFFF | (1<<19);

    // SPORT0 as receiver, SLEN is confusing still...
    *pSPCTL0 = OPMODE | L_FIRST | SLEN24 | SPEN_A | SCHEN_A | SDEN_A;

    // SPORT1 as transmitter
    *pSPCTL1 = OPMODE | L_FIRST | SLEN24 | SPEN_A | SCHEN_A | SDEN_A | SPTRAN;
}
```

Our above initDMA is inspired by Mitch's code but required significant alterations in order to work with our intended purposes.



```
void processSamples()
{
    while( ( ((int)rx0a_buf + dsp) & BUFFER_MASK ) != ( *pIISP0A & BUFFER_MASK ) ) {

        /*
        delay_ptr is putting what rx just took in into the delay_buffer.
        once the delay length is satisfied, dsp pointer is adding to the receive buffer what rx
        already put there PLUS what's just ahead of where delay_ptr is now. this way,
        the desired delay time is satisfied constantly.
        */
        delay_buffer[delay_ptr] = rx0a_buf[dsp];        // fill up the delay buffer

        delay_ptr = (delay_ptr + 1)%DELAY_LENGTH;

        rx0a_buf[dsp] += delay_buffer[ delay_ptr ];

        dsp = (dsp + 1)%BUFFER_LENGTH;                  // increment the buffer_ptr
    }
    return;
}
```