Lab 6e Sound generation using a Digital to Analog Converter

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

Goals
- DAC conversion,
- SPI interface,
- Design a data structure to represent music,
- Develop a system to play sounds.

Review
- Data sheets on 74HC595,
- Valvano Chapter 7 on SPI interfacing,
- Valvano Chapter 8 on NPN transistors,
- Valvano Chapter 11 on DAC converters.

Starter files • **`OC3`** project

Background

Most digital music devices rely on high-speed DAC converters to create the analog waveforms required to produce high-quality sound. In this lab you will create a very simple sound generation system that illustrates this application of the DAC. Your goal is to play your favorite song. For the first step, you will interface a 74HC595 serial in/parallel out shift register to the SPI port. Please refer to the 74HC595 data sheets for the synchronous serial protocol. The second step you need to perform is to create a DAC from the 8-bit digital output of the 74HC595. You are free to design your DAC with a precision anywhere from 5 to 8 bits. You will convert the binary bits (digital) to an analog output current using a simple resistor network. The third step is to convert the DAC analog output to speaker current using a current-amplifying NPN transistor. It doesn't matter what range the DAC is, as long as there is an approximately linear relationship between the digital data and the speaker current. To do this you will have to run the NPN transistor in it's linear range. The performance score is not based on loudness, but sound quality. On the other hand, sound quality will be a function of the number of DAC bits, the linearity of the analog circuit, and the periodic output rate. If an analog signal is noisy, you can add filter capacitors. It is important to add a 0.1µF bypass capacitor on the power connection of the 74HC595 to prevent output glitches during serial input transmissions. The fourth step is to design a low-level device driver for the DAC. A single 8-bit SPI frame is all that is required to set the DAC output.

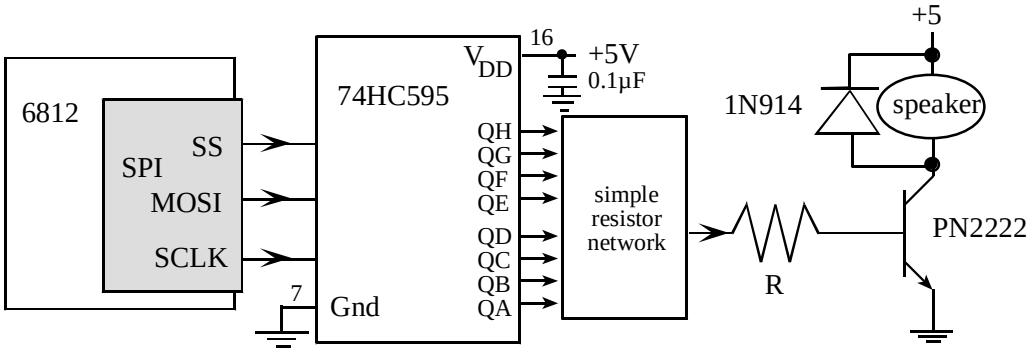

Figure 6.1. DAC allows the software to create music.

The fifth step is to design a data structure to store the sound waveform. You are free to design your own format, as long as it uses a formal data structure (i.e., **`struct`**). Compressed data occupies less storage, but requires runtime calculation. On the other hand, a complete list of points will be simpler to process, but requires more storage than is available on the 6812. The sixth step is to organize the music software into a device driver. Although you will be playing only one song, the song data itself will be stored in the main program, and the device driver will perform all the I/O and interrupts to make it happen. You will need public functions **`Rewind`**, **`Play`** and **`Stop`**, which perform operations like a cassette tape player. The **`Play`** function has an input parameter that defines the song to play. A background thread implemented with output compare will fetch data out of your music structure

and send them to the DAC. The last step is to write a main program that inputs from three binary switches and performs the three public functions.

Preparation (do this before your lab period)

1. Draw the circuit required to interface the 74HC595 to the 6812 SPI port. Include signal names and pin numbers. The bypass capacitor on the +5 V supply of the 74HC595 can be any value from 0.1 µF to 0.22 µF.

2. Design the DAC converter using a simple resistor-divider technique. Use resistors in a 1/2/4/8/16/32 resistance ratio. Select values in the 5 kΩ to 200 kΩ range. For example, you could use 5 kΩ, 10 kΩ, 20 kΩ, and 40 kΩ. Notice that you could create double/half resistance values by placing identical resistors in series/parallel. Using Ohm's law and the properties of the NPN transistor, make a table of the transistor base current for each of the possible digital values. Using the current gain of the NPN transistor, make a table of the speaker voltage and current as a function of digital value. If you add an analog filter connect a capacitor from the analog signal to ground. Use ceramic or Mylar capacitors with a value ranging from 100 to 1000 pF. Do not use tantalum or electrolytic capacitors.

3. Write a low-level device driver for the SPI interface. Include two functions that implement the SPI/DAC interface. The function **`DAC_Init()`** initializes the SPI protocol, and the function **`DAC_Out()`** sends a new data value to the DAC. Create separate **`DAC.h`** and **`DAC.c`** files.

4. Write a couple of simple main programs that test the SPI/DAC interface. You will use this software to test the SPI interface, and the DAC hardware. This main program can be used for static testing.

```
void main(void){unsigned char number;
  SCI_Init(38400);   // initialize SCI interface
  DAC_Init();        // initialize SPI/DAC interface
  while(1){
    number = SCI_InUHex(); // read from PC keyboard
    number = number&0x1F;  // 5-bit only
    DAC_Out(number);       // output to SPI
  }
}
```

This main program can be used for dynamic testing. It creates a triangle waveform.

```
void main(void){unsigned char n;
  DAC_Init();    // initialize SPI/DAC interface
  while(1){
    for(n=0; n<32 ; n++){ // up 0 to 31
      DAC_Out(n);         // output to SPI
    }
    for(n=30; n>0 ; n--){ // down 30 to 1
      DAC_Out(n);         // output to SPI
    }
  }
}
```

5. Design and write the music device driver software. Create separate **`Music.h`** and **`Music.c`** files. Place the data structure format definition in the header file. Add minimally intrusive debugging instruments to allow you to visualize when interrupts are being processed.

6. Write a main program to run the entire system.

A “syntax-error-free” hardcopy listing for the software is required as preparation. The TA will check off your listing at the beginning of the lab period. You are required to do your editing before lab. The debugging will be done during lab. Document clearly the operation of the routines.

Procedure (do this during your lab period)

1. Use the simple main programs to debug the SPI/DAC interface. Experimentally measure the speaker voltage/current versus digital value. Compare the measured data from the predicted data calculated as part of the preparation. Adjust resistance and capacitor values to get an approximately linear relationship between the digital output and the speaker current.

2. Using debugging instruments, measure the maximum time required to execute the periodic interrupt service routine. Adjust the interrupt rate to guarantee no data are lost.

3. Debug the music system.

Deliverables (exact components of the lab report)

A) Objectives (1/2 page maximum)
B) Hardware Design
    Detailed circuit diagram of all hardware attached to the 6812 (preparation 1 and 2)
C) Software Design (no software printout in the report)
    Draw pictures of the data structures used to store the sound data
    Draw a data flow graph illustrating out information in the data structure is converted to music
D) Measurement Data
    Show the theoretical response of speaker current versus digital value (preparation 2)
    Show the experimental response of speaker current versus digital value (procedure 1)
E) Analysis and Discussion (1 page maximum)

Checkout (show this to the TA)

You should be able to demonstrate the three functions **`Rewind`**, **`Play`** and **`Stop`**. You should be prepared to discuss alternative approaches and be able to justify your solution.
Your software files will be copied onto the TA's zip drive during checkout.

Hints

1) There are two versions of the Adapt812 board[1].

[1] Rev1 and Rev2 have different H1 connector pin assignments for PS5 PS6, PS7