Lab 5 – DAC and Timer2 module

Objectives

The objectives for this lab are to:

1. use a DAC to output a desired voltage
2. configuring a timer to overflow at a specific time
3. place and access data in specific memory spaces
4. define function prototypes for internal and external procedures
5. read datasheets to determine how the C8051 DACs operate

Background

The lab this week and next week both deal with some basic subroutines for the digital-to-analog converters (DAC0 and DAC1) that we will use in the project. The DACs will be directly driving the output of our MP3 player. You will need the C8051F120 datasheet for the following information:

- Sections 8 and 9 on the DAC and voltage reference
- Section 23 on Timer 2

You will also need to read the datasheet for the headphone jack:

- SJ-3524NG 3.5mm Stereo Audio Jack

This week's lab is focused on getting the DACs data ready for use and coordinating with the interrupt handler. Next week's lab will focus on sending the data to the DACs with the interrupt handler and controlling the sound volume and balance. The code for that code will be written in C.

The music player creates sound by driving the headphone speakers using sound samples read from specially formatted sound files stored on a microSD memory card. The sampled sounds will be converted by the DAC into an analog voltage in the range +0V to +2.4 V, which will then have the DC offset removed via a capacitor. In order to get decent (but not great) quality the sound data will be stored as 8-bit values, sampled at a frequency of 8.0KHz, 11.025KHz or 22.05KHz.

In order for the sound reproduction to be as accurate as possible, the DACs needs to be updated promptly. We can handle this issue by using Timer 2 to keep the DACs happy. Both DACs can be configured so that the data can be buffered and output only when a timer overflows. You will get an interrupt when the timer overflows, and the handler will then prepare the next byte(s) of data for each channel.

A primary goal for this week's lab is for your group to determine how the C8051 DACs operate by reading the datasheets. The pre-lab questions below focus almost exclusively on preparations for the DACs (each DAC is identical except for the output pin and SFR page setting).

Pre-lab

Prior to the lab, you and your lab partner must turn in a "pre-lab" which answers the following questions. Your group will need the answers to these questions in order to get this lab working correctly.

1. Which pins on the C8051F120 are the DAC0 and DAC1 outputs? Do you need to configure the crossbar for this? If so, which bit(s) in the **XBRn** SFRs need to be set/cleared?
2. Scrutinize Figure 9.1 and SFR Definition 9.1 in the C8051F120 datasheet.
   a) What pin on the C8051 development board's 3 by 32 connector is **VREFD**? Where should it be connected?
   b) What does **REF0CN** do? What value should it have for this application?
3. What values should be loaded into Timer 2's RCAP2 SFRs to cause a timer overflow approximately 8,000 times a second (i.e., at 8KHz)? How much error does this setting cause, if any? Assume that SYSCLK is 12.25MHz (a reasonable assumption, because it will be).
4. How are the SFRs for the DACs configured to automatically output a value on a timer overflow?

Lab work

At the completion of this lab, you need to turn in an assembly module named **daca.asm** with all the procedures listed below. There are three global variables used by your module*:*

- *bytesleft* : this is a 16-bit unsigned integer, big endian format (i.e., MSB first). It contains the number of bytes which remaining to be sent to the DACs. It is placed in direct internal RAM.
- *bufptr* : this is a 16-bit pointer to external RAM (XRAM), big endian format. It contains the address of the next byte of data to be sent to the DACs. It is also placed in direct internal RAM (note that while the *variable is stored* in direct internal RAM, the *address contained in the variable* refers to external RAM).
- *dacactive* : this is a single-bit variable. It signals whether the code is sending data to the DACs

Since the code for next week's lab will need access to these variables, your module must make them **public**.

You will write three procedures for the **daca.asm** module:

1. **dacinit**: This procedure sets up DAC subsystem and timer. Initialization of the module includes initializing the global variables to reasonable values, setting up the DACs, and setting up the Timer 2 interrupt and Timer 2 toggle output pin on the crossbar. After this procedure is called, the timer should be running and the system ready to output an 0x80 to the DAC0/DAC1 data SFRs at a sample rate of 8000 Hz.
2. **dacbusy** : This procedure returns a zero in register R7 if *dacactive* is not set. It is used by the main program to check whether all data has been sent to the DAC.
3. **_dacplay**: This subroutine is passed two parameters: an unsigned 16-bit number (*count*) and a

16-bit pointer to external RAM (*buffer*). *count* is passed in registers R6 and R7 (MSB in R6) and *buffer* is passed in registers R5 and R4 (MSB in R4). It prepares the variables used by the interrupt handler by copying *count* to *bytesleft* and *buffer* to *bufptr*, then setting *dacactive*.

Note: two other things are very important about **_dacplay**:

i. The underscore in the function name is important; you need to include it.

ii. You should not call **_dacplay** without first calling **dacbusy** to be sure *dacactive* is clear.

On the hardware side, you will need four wires: one for each DAC output, one for the *analog* ground, and one for the **VREFD**. Use 100uF capacitors to connect the DAC outputs to the headphone jack.

You will test your module using **lab5test.c** and **lab5dac.obj**, available on Sakai. You will add **lab5test.c** to your project and link **lab5dac.obj** with the project as you did with **scancodes2.obj** in Lab 4. You will also link your **kbd.obj** (the version using interrupts) from Lab 4. The test program sends data for a sine wave to both DACs; you can select which sine waves is sent using the number keys **1** or **2** on the PS/2 keyboard. The **t** key will toggle between the two sets of sine waves.

You should observe:

- the analog waveforms for the default configuration (key **1**) and measure the frequency, amplitude and DC offset on each channel; record these in your lab notebook.
- the waveform for the Timer 2 toggle output and measure the frequency; record these in your lab notebook.
- how the main module uses **dacbusy** and **_dacplay**; you will need to replicate this in future labs.