CMPE 121L: Microprocessor System Design Lab

Spring 2019

Lab Exercise 2

Check-off Due in lab section, week of April 22, 2019

Report due: Monday, April 29, 11:55 PM

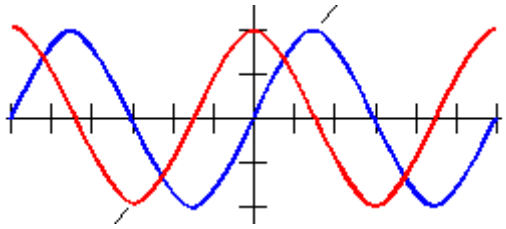
The purpose of this exercise is to learn how to use the DMA Hub in the PSoC 5 system. In the first part you will design a dual-channel waveform generator with controllable phase. In the second part you will use the DMA hub to perform a memory-to-memory block transfer with byte transposition, and contrast this with using a software loop to perform the same block transfer.

Interfacing the LCD Display

You will need to use the 16x2 LCD display for this lab. Follow the instructions in the Lab 1 handout to connect the LCD display to the PSoC board.

Part 1a: Dual-Channel Waveform Generator

This is an extension of Example 3 in the Cypress Application Note *An Introduction to DMA.* You will use a lookup table stored in flash memory to create two sine waves on two output pins with a variable phase difference between them, as shown below.



The blue wave can be generated by following Example 3. The red wave has the same amplitude and frequency, but you should be able to vary its phase with respect to the blue wave from 0 to 360 degrees by turning the potentiometer supplied in the parts kit. The phase shift value in degrees should also be displayed on the 16x2 LCD display. The frequency of both waves must be configurable in the range 100 Hz – 10 kHz through a parameter in your C code.

Here are some hints on how to proceed.

1. First, read the application note carefully and try out some of the examples on the board. You should also read the appendices, which describe how to set up the DMA channels and program them.
2. You may use the DMA Wizard in PSoC Creator (accessible from the tools menu) to create the API function calls, or write the functions calls yourself.
3. You can use the same 128-entry lookup table and 8-bit DACs to generate the sine waves. A single lookup table can drive both waveform outputs, but you will need a separate DMA channel for each. The key is to use the Transaction Descriptors cleverly.

4. Changing the phase of the wave can be achieved by changing the parameters of the Transaction Descriptors associate with the DMA channel (source address and length).
5. There can be some noise in the potentiometer readings and its output voltage can change slightly even when its position is not being changed. If your program re-configures the DMA parameters at a fast rate in response to the noise, the resulting waveform will not be stable. Your design should incorporate some mechanism to deal with this problem (for example, ignore any changes in potentiometer readings below a minimum threshold).
6. Avoid using periodic interrupts, as they consume CPU cycles. A good solution should not generate any interrupts when there is no change in the potentiometer setting.
7. You can place your lookup table array in flash memory using the CYCODE keyword. For example,

   ```
   CYCODE uint8 myTable[4] = {1, 2, 3, 4};
   ```

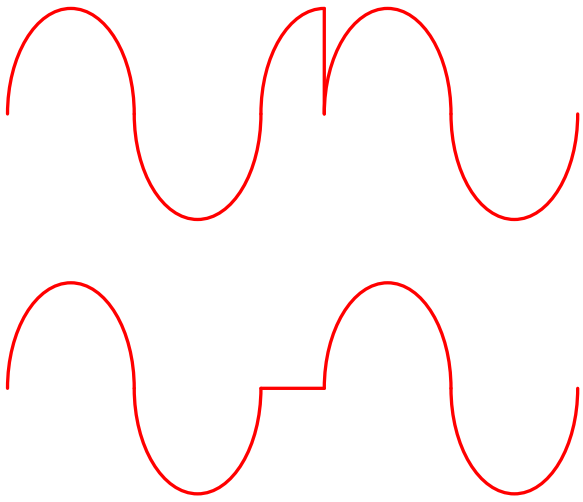
   defines a 4-entry table in the on-chip flash memory. The flash memory is non-volatile, but is much slower than the SRAM, so if you use DMA to transfer data directly from the flash memory, the maximum frequency of the wave will be limited by the slow access time of the flash memory. To improve the speed, you can copy the table from flash to an array in main memory on startup and set the source of the DMA transfer to the array in main memory.

Checkoffs:

1. Your design is able to generate the two waves with frequencies configurable in the range 100 Hz- 10 kHz.
2. The phase of the second wave relative to the first can be changed using the potentiometer in the full ±180-degree range. The phase shift value in degrees is displayed on the 16x2 LCD display.
3. The waveforms remain stable when the potentiometer is not being turned and no interrupts are being generated.

Part 1b: Dual-Channel Waveform Generator with Controlled Phase Change

This is a refinement of Part 1a. The objective is to change the phase of the red sine wave only at its zero-crossing points. That is, if the potentiometer setting is changed, the system should wait until a zero-crossing point (you may choose any zero-crossing point) to shift the waveform. This avoids glitches in the output. This is illustrated in the figure below:



In the example above, the phase of the sine wave is being shifted by 90 degrees. In the top wave, the phase shift did not occur at a zero-crossing point, resulting in a glitch. In the bottom wave, there is no such glitch, which is the desired behavior.

You can switch the phase at any zero-crossing point. Note that a phase change of +x degrees is the same as a phase change of (360-x) degrees, and the exact cycle in which the phase change occurs is not important.

If you set the frequency of the wave very low (around 100 Hz), you can actually see the phase changes on the oscilloscope and confirm that they occur at zero-crossing points.

You can use interrupts for this part, but you should disable them when you don't need them (an interrupt at every zero-crossing point when the phase is not changing will consume too many CPU cycles unnecessarily).

Here are some hints on how to proceed.

1. You can detect a zero-crossing point by configuring a Transaction Descriptor to generate an interrupt.
2. To avoid servicing interrupts when there are no changes, you should normally keep the interrupts disabled and enable them only when the potentiometer is turned. The main loop can check for a change in the potentiometer setting and enable the interrupt on detecting a change. You should disable the interrupts after you have responded to the change in the potentiometer setting by shifting the phase of the wave.
3. One way to switch the phase is to detect a zero-crossing point, then pause the DMA channel for an amount of time corresponding to the phase shift, and restart the DMA.

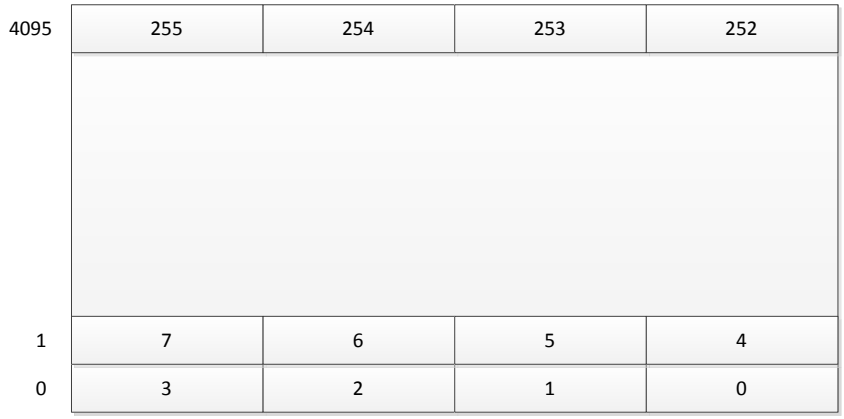
Checkoffs:

1. Your design is able to generate the two waves with frequencies configurable in the range 100 Hz- 10 kHz.

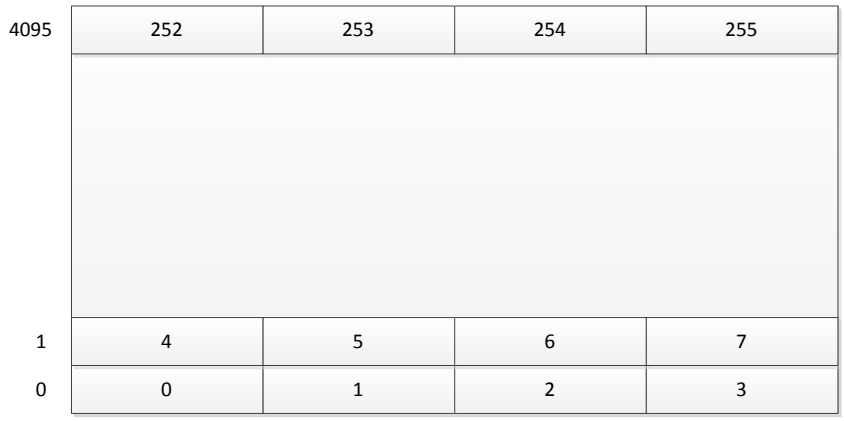
2. The phase of the second wave relative to the first can be changed using the potentiometer in the full ±180-degree range. The phase shift value in degrees is displayed on the 16x2 LCD display.
3. The waveforms remain stable when the potentiometer is not being turned.
4. Phase changes always occur at the zero-crossing points of the second wave.

Part 2: Memory-to-Memory Block Transfer

In this second part, you will use the DMA hub to transfer a block of data from one part of the RAM to another, with a change in the format from little-endian to big-endian.



Source Array



Destination Array

The format of the block of data is shown above. The size of the block is 4096x4 = 16,384 bytes, and the individual byte locations are filled with an increasing pattern. When the program is run, it should copy the values into a destination array in the format shown.

Steps:

1. Write a program to create the source array in RAM and fill it with an increasing pattern of byte values (modulo 256).
2. Clear the destination array by filling with zeroes.
3. Configure a DMA channel to move the data in byte-transposed order to the destination array, and interrupt the main program when the DMA transfer is complete.
4. Note that the Transfer Count parameter of the TD has a maximum limit of 4095. Thus, you can only transfer a maximum of 4095 bytes from the source array to the destination array using a single TD. You will need to use a TD chain with multiple descriptors to transfer the 16K block.
5. In the ISR, set a flag to indicate the end of the transfer. The main loop should wait for this flag to be set, and then use a software loop to compare the source and destination arrays (taking into account that the bytes are transposed), and verify that the transfer was done correctly. It should then display the number of mismatches found on the LCD display.
6. Test your program.
7. Add a hardware timer to your design and measure the time taken for the block transfer using the timer as accurately as possible (start the timer when the DMA is started and stop it when the DMA controller interrupts at the end of the transfer). Display the time taken in microseconds on the LCD.
8. Now replace the main loop with a software loop to perform the same transfer done by the DMA Hub, with the following steps:
    a. Clear the destination array by filling with zeroes.
    b. Transfer the bytes to the destination array from the source array using a program loop, with the bytes transposed.
    c. Verify that the transfer was done correctly by comparing the bytes.
9. Compute the transfer time as in step 6. See if you can find any optimizations to the main loop that will improve the throughput.
10. Compare the throughputs (in bytes transferred per second) for the two approaches (software loop and DMA) to determine the speedup (or slowdown) for the DMA transfer over the program-controlled transfer. Explain why they are different.

Checkoffs:

1. Show a single DMA data move with error checking in the end that displays the number of errors and the time taken in microseconds on the LCD.
2. Show the program-controlled transfer with the time taken on the LCD.

What to submit?

- Schematics and code for all parts
- Report with details of your designs, results, and discussion.