Oscilloscope Project

Scott Ficher

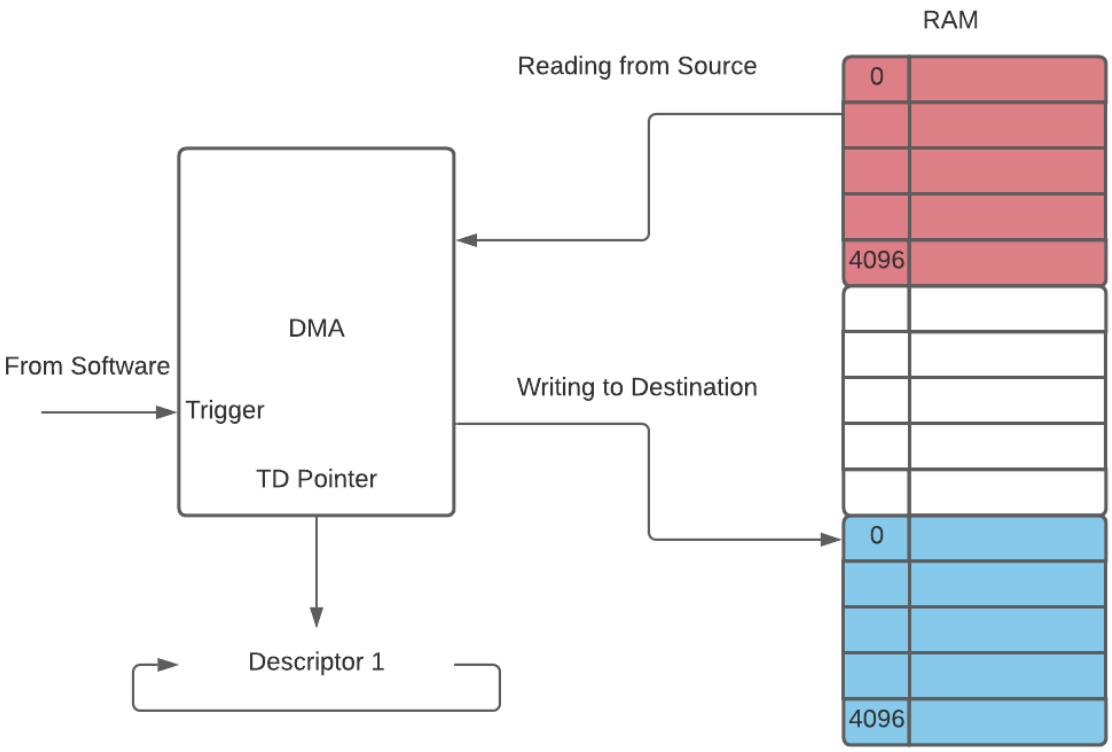
Introduction:

In this project I focus on implementing the DMA component in a number of ways to manipulate data in RAM. I built a system using the DMA and ADC to measure multiple types of waveforms ranging from 1kHz to 100kHz.

DMA:

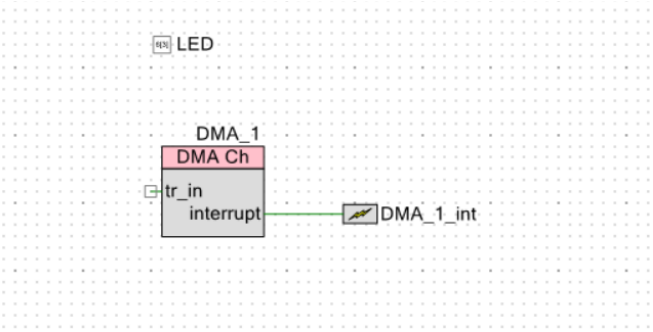
The DMA is essentially a funnel that pushes data from one point to another. In this lab we only implement one channel at a time and utilize multiple descriptors to shift and manipulate the data as we see fit. For example, when we change the endian type we have to shift the indices. To do this I could use 4 descriptors and set the destination index different from the source index.

Design:

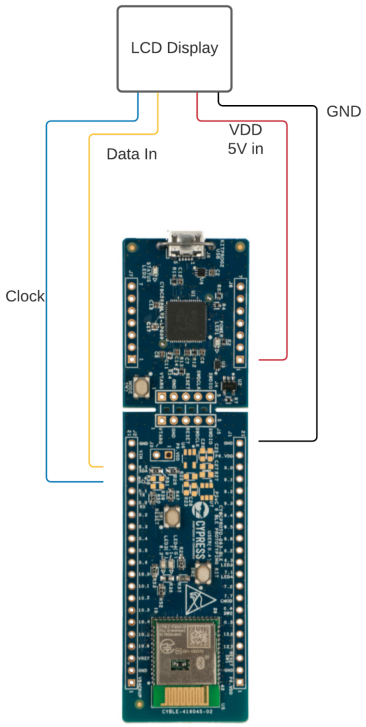


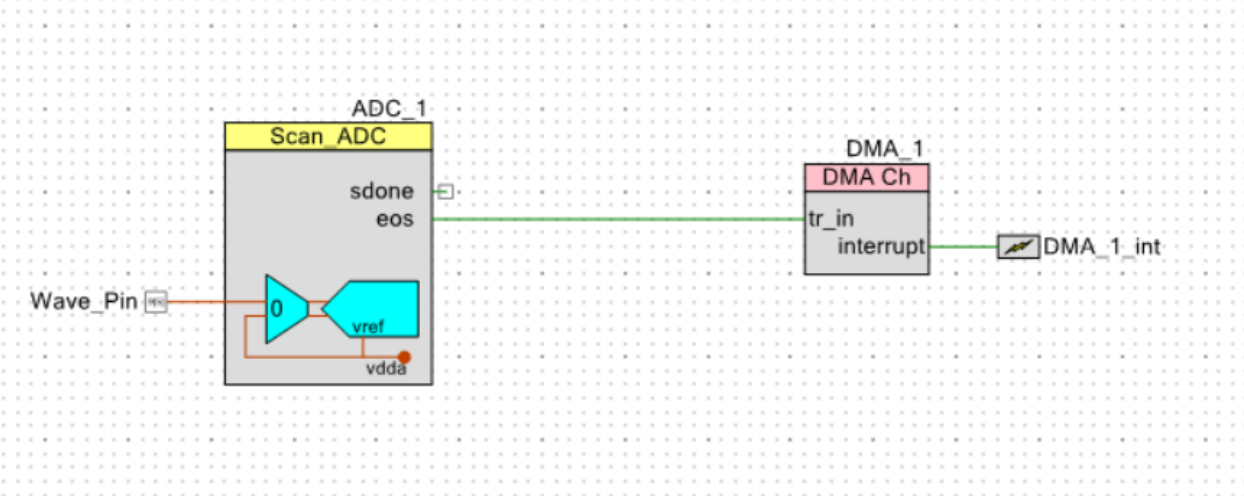
Basically we read from some address in RAM and write the same exact data to a different address in RAM and then increment both addresses by 32 (4 bytes or 1 element size) and repeat 4096 times until both the source array and destination array are respectively either completely read or completely filled.

Oscilloscope :



To design the oscilloscope it got a little more complicated as I needed to bring in the ADC component and connect it to the trigger input of the DMA. This meant that whenever the ADC converted a signal the DMA would transfer that chunk of data (in this case an unsigned 16 bit int) to a section of an array. After 256 chunks of data were transferred we switched to the other array and filled it up while our CPU processed the filled array. To do this I first created the two components and set them up and checked the output of the ADC to ensure it was triggering about every 1us.





Once I was able to ensure the trigger I checked to make sure the DMA interrupt signal was working by switching a flag on and off and printing to the UART terminal. In the ISR I have it set a flag that tells my main loop which array is ready for processing, 1 or 2. Once you enter that if statement, it begins to iterate over the 256 indices of the array. It stores the previous reading and the current reading to create a small window where we look to see if the threshold is crossed during that point in time. If it is crossed we calculate the time since the last crossing and divide 1000 by that time difference to find the frequency in kHz. I take this frequency and add it to a running count and increment the frequency_count variable that keeps track of how many frequencies I have added. I then use this when the arrays are not being processed to find the average and print to the LCD screen. The minimum is about 300ms in between prints to the LCD screen. I chose to do an average as it gave me a way of letting the arrays process very quickly without printing to the screen each time a frequency was found, when I originally had it print each time a frequency was found it was both not accurate and would flicker too rapidly. The average fixed both of these problems. At the end of the array processing I clear the flag in order to show the main loop there is time to print to screen.

Testing:

The oscilloscope was a bit tricky in terms of testing due to the hardware and software components. At first I started with the ADC component to make sure it was producing a pulse correctly. I put the output to a GPIO pin and used the AD2 oscilloscope function to ensure it was correct. This gave me no problems so I moved on to the DMA and used the debug mode in PSoC creator to look at the arrays as each time the interrupt was triggered. I found, just like in the video, that there was some noise with the numbers coming into the array and so at first I figured I could solve this by using a padding system. I had the threshold be exactly the average 0x400 and then if the number was above or below this number by 100 it would still count as the threshold. I found this to not work as well as I thought it would, and there would be times when two numbers would be within the threshold, so the time difference would be low and thus the frequency would be off. I then redesigned it to track the current index and the previous index. If one was above and one was below the average of 0x400 then it would count as crossing the threshold. This fixed the problem of two indices being near the threshold. The next problem I found was that there was still a lot of noise due to the distances between being so low. For example, if we have a high frequency it may cross the frequency every 5 indices, but this would come in as a 5 or a 6. Even a difference of 1 in the measured index meant the frequency was drastically different and would produce a lot of noise. In order to fix this I took the average of the frequency over a certain time. This made the noise drastically reduced. The last problem I encountered was in the speed of the loops. Incrementing over all 256 indices and printing to the screen proved to be costly when our arrays are changing so quickly. Instead I set up the system I talked about in the design section. I wait until the CPU is done processing arrays then find the average frequency and print to the screen. There is also a loop minimum for the print. There must be 3500 loops done before it can print; this ensures the average frequency will have enough data inputs to be accurate. This mostly fixed the speed issue that I was having however occasionally I would still get some errors so I did not speed it up perfectly due to how slow printing to the LCD display proved to be.

Conclusion and Results:

For the oscilloscope I am happy, although I did not totally eliminate all the noise I was able to greatly reduce it and was able to get the entire spectrum from 1kHz to 100kHz. I found that the iterative approach I took to be helpful, although at first I was reluctant to give up my original design. In the future I will try to be less attached and be more willing to redesign my software to match the problems I am currently seeing so that I can continue to innovate my software iteratively and efficiently.