Zak Rowland
CST331
Lab 2 – DMA
Conclusions

a. My primary overarching issue was not fully understanding how DMA works and how it interacts with interrupts. Simply configuring the DMA channels and receiving a character as a test proved to be difficult. One issue I had was still trying to use the UART routine from last lab because I didn't realize the DMA handles everything this routine did. As I progressed through the lab, I began to understand DMA more thoroughly and progress became steady. Once I was able to send and receive data I ran into an issue where the last transaction would send a full block even if the block was only partially full. I figured out that adjusting the cell size will also change when a block complete interrupt takes place, so I was able to fix this. Step 9, which was simply toggling an LED via DMA also gave me some trouble. I was under the impression that the external interrupt had to at least be enabled in order to trigger a DMA transaction, however no interrupt configuration was required besides setting the start IRQ in the DMA configuration.

b. The system clock used is 84MHz, PBCLK2 is 84MHz, and PBCLK3 is 84MHz.

c. It is important to have the LED be lower priority because we don't want the interrupt to trigger and be serviced during a period of receiving data via the DMA because this data could be lost.

d. I determined interrupt prologue latency by setting a breakpoint on the first assembly instruction of the prologue and the first C instruction of the ISR then sending a file from RealTerm. The prologue took 13 cycles or 154.76ns. I determined the latency to DMA receive ready by setting a breakpoint at the first C instruction of the routine and an ending breakpoint at the store word of the DMA 0 channel enable set. The latency until DMA 0 is ready to receive another character is 48 cycles or 571.39ns. The total latency is 61 cycles or 726.15ns.

e. The measured latency was 192ns +/- 1ns which is equivalent to 16.128 cycles at 84MHz. The latency of UART sending data at 115.2K baud was 1119.05ns (uncached.) DMA is much faster because it works independently of the CPU.

f. The time it takes to handle a receive block complete is DMA hardware latency + interrupt latency + block complete execution time. This time equals 192ns + 154.76ns + 571.39ns which is 918.15ns.

g. One way to do this is to leave the UART receive active but the DMA channel is disabled so the UARTs receive FIFO buffer will catch some additional bytes of incoming data before becoming full. If the interrupt service finishes in

time, no data will be lost on the receive. Another way to accomplish this would be setting the DMA destination to a temporary buffer to catch data while the interrupt is being serviced.

h. i. DMA requires less system overhead than interrupts because it runs independently of the CPU. When using interrupts, the CPU must drop what it's doing to service the pending interrupt.
ii. Implementing interrupts seems easier to me because the routine is built like a function and the operation is more understandable. DMA functions "under the hood" so to speak, so debugging it when something isn't working can be difficult. The configuration is also more involved when compared to configuring interrupts.

i. i. If the time between character receives is too long the LED can turn off.
ii. This time is so small that our persistence of vision makes the LED appear steadily on.
iii. I hooked scope probes to the LED signal on RH8 and the UART receive on RD6 and sent over a file from RealTerm as well as typing at various speeds. In some places the LED signal is set low for a fraction of time while receiving characters.
iv. If the characters arrive continuously the first approach has less overhead because the system isn't waiting for varying times to pass. The second approach is better for varying times because the timer is running continuously.

j. The block complete routine tracks the number of 1KB blocks received. If this number is greater than 7 (8KB has been received,) the destination address is reset and the channel is left disabled until the buffer is cleared by transmission.