Lab 8 - Taylor Cowley and Andrew Okazaki

- DMA Controller
    - Driver API
    - Performance Analysis
    - Bug Report

Driver API

Code to interact with the DMA is auto-generated by the EDK, but we edited it to make it a little more friendly.
For an end-user, the only required function is the last one: `DMA_transfer_GO`. The parameters are a `sourceAddress`, a `destinationAddress`, and a `length`. The other two functions are included to demonstrate what DMA_transfer_GO calls.

```
void DMA_Init(Xuint32 BaseAddress, Xuint32 srcAddress, Xu
int32 destAddress, Xuint32 length){
        /*Set control register to burst write operation
*/
      Xil_Out8(BaseAddress+DMA_MST_CNTL_REG_OFFSET, MST_B
RRD);
        /*Set slv_reg0 to source address */
      Xil_Out32(BaseAddress+DMA_SLV_REG0_OFFSET, srcAddre
ss);
```

```
        /*Set slv_reg1 to destination address  */
      Xil_Out32(BaseAddress+DMA_SLV_REG1_OFFSET, destAddr
ess);
        /*Set data transfer length */
      Xil_Out32(BaseAddress+DMA_SLV_REG2_OFFSET, length);
        /*Set byte lane value */
      Xil_Out16(BaseAddress+DMA_MST_BE_REG_OFFSET, 0xFFFF
);
}
void DMA_Go(Xuint32 BaseAddress){
         /*Start user logic master write transfer */
      Xil_Out8(BaseAddress+DMA_MST_GO_PORT_OFFSET, MST_ST
ART);
}

void DMA_Stop(Xuint32 BaseAddress){
         /*STOP user logic master write transfer */
      Xil_Out8(BaseAddress+DMA_MST_GO_PORT_OFFSET, MST_ST
ART+1);
}

// This is the code that actual users will call. Because
this file is including "xparameters.h",
// it already knows what the DMA baseAddress is, so the u
ser does not need to input it.
void DMA_transfer_GO(Xuint32 srcAddress, Xuint32 destAddr
ess, Xuint32 length){
    Xuint32 BaseAddress = XPAR_DMA_0_BASEADDR;
   // We already know the base address!
    DMA_Init(BaseAddress, srcAddress, destAddress, length
); // So hand it to init
    DMA_Go(BaseAddress);
   // and go!
}
```

Performance Analysis

Running the timer for the software copying, it takes `14494300` clock ticks. At 100MHz, that is 0.000014494300 seconds, or 0.01449 ms. Starting the timer when the hardware copy starts and ending it when the DMA interrupt happens, it takes `12972638` ticks. This is 0.000012972638 seconds, or 0.01297 ms This is faster, but only by about 12%.
One would think that the hardware copying would be a lot faster, but this is not the case. I think this is because even the hardware copy requires clock ticks to cycle through the memory, and it can only copy so much memory in parallel; it is limited by the bus.

Bug Report

The bugs we encountered were understanding the signals with in the DMA controller. It took a while to complete the state machine with in our DMA controller. The problem existed when trying to read and write correctly. To solve this problem it took a lot of simulations to understand what everything was doing. While writing the software during our screen capture in with our DMA controller we were only capturing the top quarter of the screen. Wondering what was going on we realized that we down sized the screen meaning that each pixel we used was really four bytes in hardware. After understanding that our program seemed to work smoothly. I think that this was the most bug prone lab we worked on this year.