DMA Capture

Space Invaders

Ty and Hampton Madsen

BYU

12/10/2015

Contents

Chapter 5: DMA Controller

Section 5.1: Driver API

```
/*
 * Initialize the DMA controller
 * Sets the operation type to a burst read
 * Sets source, destination and length to slave registers in the DMA
controller
 * Sets the byte lane to all
 */
void DMA_CONTROLLER_TranseferInitialize(Xuint32 BaseAddress, Xuint32
srcAddress, Xuint32 destAddress, Xuint32 length){

        // Set control register to burst write operation
        Xil_Out8(BaseAddress+DMA_CONTROLLER_MST_CNTL_REG_OFFSET, MST_BRRD);
        xil_printf("control reg: %x (should be: %x)\r\n",
Xil_In8(BaseAddress+DMA_CONTROLLER_MST_CNTL_REG_OFFSET), MST_BRRD);

        // Set slv_reg0 to src address
        Xil_Out32(BaseAddress+DMA_CONTROLLER_SLV_REG0_OFFSET, srcAddress);
        xil_printf("slave reg 0 (source): %x (should be: %x)\r\n",
Xil_In32(BaseAddress+DMA_CONTROLLER_SLV_REG0_OFFSET), srcAddress);
        // Set slv_reg1 to dest address
        Xil_Out32(BaseAddress+DMA_CONTROLLER_SLV_REG1_OFFSET, destAddress);
        xil_printf("slave reg 1 (dest): %x (should be: %x)\r\n",
Xil_In32(BaseAddress+DMA_CONTROLLER_SLV_REG1_OFFSET), destAddress);
        // Set data transfer length
        Xil_Out32(BaseAddress+DMA_CONTROLLER_SLV_REG2_OFFSET, length);
        xil_printf("length reg: %x (should be: %x)\r\n",
Xil_In32(BaseAddress+DMA_CONTROLLER_SLV_REG2_OFFSET), length);
        // Set byte lane value
        Xil_Out16(BaseAddress+DMA_CONTROLLER_MST_BE_REG_OFFSET, 0xFFFF);

}
/*
 * Kicks off the read/write transaction by setting the go value (0x0a) to the
master go register
 */
void DMA_CONTROLLER_TransferGoGoGOOOOO(Xuint32 BaseAddress){
       // Start user logic master write transfer by writting special pattern
to its go port.
        Xil_Out8(BaseAddress+DMA_CONTROLLER_MST_GO_PORT_OFFSET, MST_START);
}
```

Section 5.2: Performance

The performance of the software drastically improved with compiler optimization.

The average time taken to perform each type of screen capture is show in the tables:

Capture Type	Time (in clocks)	Time (in milliseconds)
DMA	13328516	133.285
Software	16593403	165.934

Table 1: No compiler optimization

Capture Type	Time (in clocks)	Time (in milliseconds)
DMA	16127734	161.277
Software	8451879	84.518

Table 2: Best (-03) compiler optimization

As shown, the DMA capture is a little faster than the Software with no optimization, but with the best optimization it takes approximately twice as long as the Software capture.

Section 5.3: Bugs

The primary bugs we encountered were first that our state machine to read and write was not working properly. We needed to manually assert the read request and write request during the respective read and write states, so it would continue to copy data. We also had issues when using the master length register. When we would write the GO command, it would write the data, and then tie up the Microblaze, requiring a power-cycle to reset the Atlys board. We decided to use a slave register for the length and this issue went away, although we are not sure why this solved the problem.