|  |
| --- |
|  |
| DMA Capture |
| Space Invaders |
| Ty and Hampton Madsen |
| **BYU** |
| **12/10/2015** |

|  |
| --- |
|  |

Contents

[Chapter 5: DMA Controller 2](#_Toc437517358)

[Section 5.1: Driver API 2](#_Toc437517359)

[Section 5.2: Performance 3](#_Toc437517360)

[Section 5.3: Bugs 3](#_Toc437517360)

# 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 : No compiler optimization

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

Table : 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.