MIPSfpga by Imagination

Lab 11

Direct Memory Access (DMA)

These materials produced in association with Imagination.
Join our University community for more resources.
community.imgtec.com/university

MIPSfpga Lab 11: Direct Memory Access (DMA)

1. Introduction

In this lab you will modify your MIPSfpga system to enable direct memory access (DMA). You will design, build, and test a DMA Engine. As always, first read the entire document, especially the **What to Turn In** section, before beginning your work.

2. Direct Memory Access (DMA)

DMA enables the transfer of data from one peripheral or memory to another without the use of the CPU. This allows the CPU to continue its work while the data transfer occurs (for example during a memory transfer to/from a graphics card). The CPU initiates the DMA transfer and then the transfer completes while the CPU continues its other work.

To enable DMA transfer, you will build a DMA engine that will:

1. Receive values from the cpu/program that will store to memory-mapped registers. These values set up the configuration parameters for the DMA transfer.
2. Drive the system bus, i.e., the AHB-Lite bus, to transfer data from one peripheral or memory to another.

The CPU and the DMA Engine take turns being the master of the AHB-Lite bus. An Arbiter module, discussed in Section 2.2, must arbitrate between which module becomes the master. While the CPU is the master of the bus, the DMA Engine must wait to access the AHB-Lite bus, and vice versa.

2.1. DMA Registers

The user program configures the DMA transfer by specifying the memory-mapped values shown in Table 1. After the source address, destination address, and size of the transfer are specified, the user program initiates a DMA transfer by asserting DMAstart (i.e., writing 1 to address 0xbf30000c).

Table 1. Required DMA Registers

Name	Memory-mapped I/O Address	Description
DMAsrcAddr	0xbf300000	Source address of data (address can be a memory or a memory-mapped peripheral)
DMAdstAddr	0xbf300004	Destination address of data (address can be a memory or a memory-mapped peripheral)
DMAsize	0xbf300008	Size to be transferred (in bytes)
DMAstart	0xbf30000c	When a 1 is written to this address, the DMA transfer is initated

2.2 AHB-Lite Signals

Table 2 lists a subset of the AHB-Lite system, which you will use to both arbitrate and drive the AHB-Lite bus.

Table 2. AHB-Lite Signals

Name	Description
HADDR[31:0]	Address bus
HRDATA[31:0]	Read data bus
HWDATA[31:0]	Write data bus
HWRITE	Write enable
HCLK	Clock
HREADY	Data ready: 1 = Ready, 0 = Wait
HTRANS[1:0]	Transaction: 10 = transferring data, 00 = idle

We have already discussed all signals except the ones shown in bold: HREADY and HTRANS[1:0]. HREADY is an input to the master modules (CPU or DMA Engine). When HREADY is 1, the master module continues making requests. When HREADY is 0, the master module is stalled, and must wait until HREADY becomes 1 to continue. Figure 1 and Figure 2 below show the AHB-Lite bus with two and one wait cycle, respectively, (due to HREADY=0) upon a read and a write.

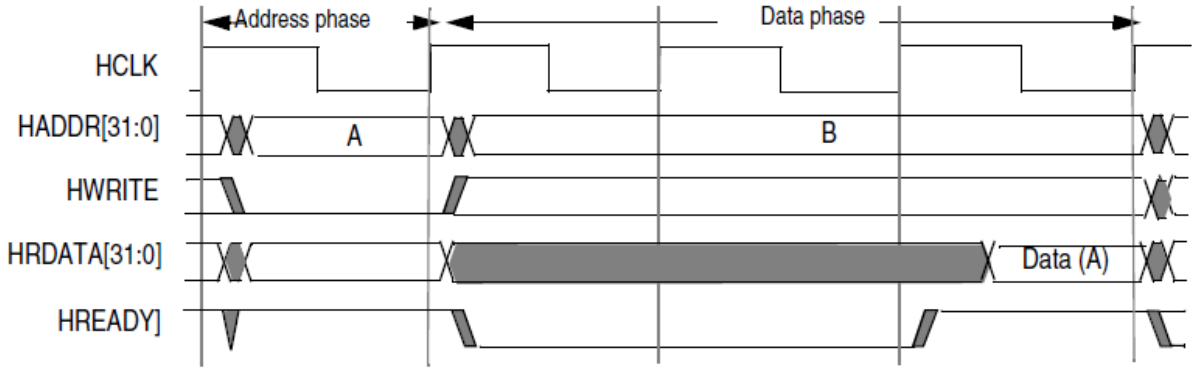

Figure 1. AHB-Lite Read transaction with two wait cycles (due to HREADY = 0)

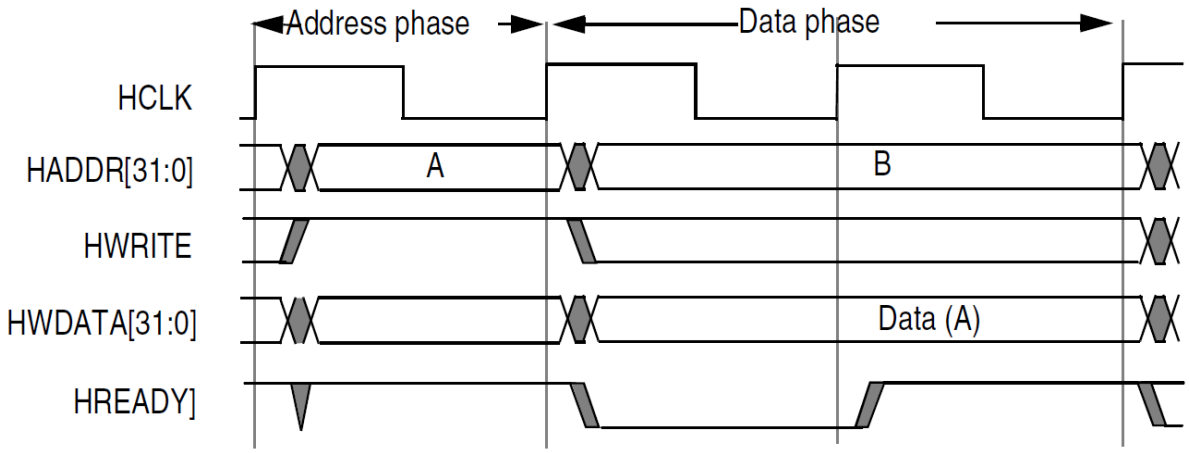

Figure 2. AHB-Lite Write transaction with one wait cycle (due to HREADY = 0)

The HTRANS signal is sent from the master. When HTRANS = **10**, the master is transferring (or attempting to transfer data, depending on the value of HREADY). When HTRANS = 00, the master is idle, i.e. not transferring data across the AHB-Lite bus.

2.3 AHB Arbiter

An AHB Arbiter module arbitrates between the MIPSfpga core and the DMA engine, driving the respective HREADY signals, as shown in Figure 3. The DMA engine will act as a slave when receiving the register values from the MIPSfpga core and a master when conducting a DMA transfer between peripherals/memory. A very high-level view of your system is as shown in Figure 3. The "AHB-Lite" signals indicate the signals listed in Table 2.

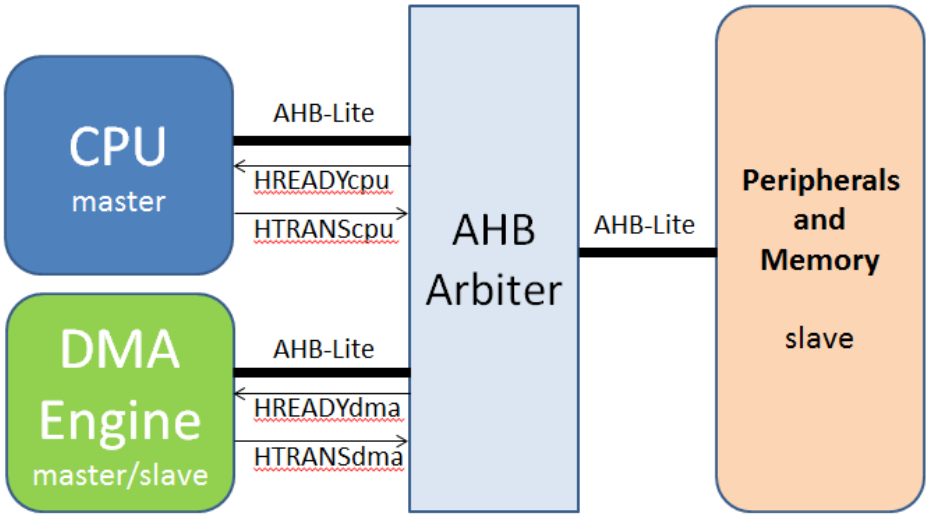

Figure 3. MIPSfpga with DMA Engine

You can choose which arbitration scheme to use, but it is recommended to use a very simple one at first. For example, the DMA engine waits for the CPU to be idle (as indicated by

HTRANScpu) and then takes control – i.e., becomes the master – of the AHB-Lite bus. Only when the DMA Engine has completed its transaction does it relinquish control back to the CPU.

2.4 FIFO

To enable the transfer of data, you will first read data from the source address (via the AHB-Lite bus) into a first-in-first-out (FIFO) memory located in the DMA Engine. The **f**irst data written **i**nto the FIFO will be the **f**irst data read **o**ut (FIFO). So, after reading data from the source (into the DMA engine's FIFO), you will write the data contained in the FIFO (via the AHB-Lite bus) to the destination address.

Your FIFO should hold up to 256 words, although your DMA engine should be able to accommodate transfers larger than this size.

As part of what you turn in, submit any schematics, diagrams, or other information related to your design that clearly, completely, and concisely explains your design. Hand-drawn and scanned/photographed diagrams are fine, as long as they are readable. Be sure to label all blocks and signals between blocks. You need only consider DMA transfers of bytes that are a multiple of four.

3. Program

After you have finished your DMA Engine hardware and have integrated it with the MIPSfpga system, write a program that does the following:

1. Declares two 500-element integer arrays: `src` and `dst`.
2. Initializes the `src` array to some known values (for example the numbers 1-500).
3. Uses DMA to transfer the 500 elements (2000 bytes) in the `src` array to the `dst` array.
4. Displays the `src` and `dst` array values before the DMA transfer (using fdc_printf) and displays the values of the `dst` array after the DMA transfer. Pushing any of the pushbuttons should pause the program so that the user can read the values printed on the OpenOCD screen.

This process should repeat for at least 10 times for different initial values stored in the `src` array.

Note that after initializing the `src` array, the contents of the data cache (D$) must be flushed to memory before DMA occurs. If this is not done, the data cache holds the current values of the array, but memory does not because MIPS uses a writeback cache instead of a writethrough cache. **IMPORTANT:** To flush the contents of the cache, use the following function, which uses inline assembly code.

```
void flushCache() {
    // Flush the data cache with the CACHE instruction, block by block.
    asm volatile
    (
        // $t1 should be set to (TOTAL_CACHE_SIZE_IN_BYTES)/16.
        "   li $t1, 256;"
        "   li $t2, 0;"
        "   lui $t3, 0x0;"
        "loop1:"
        "   cache 0x01, 0($t3);"
        "   addiu $t3, $t3, 16;"
        "   addiu $t2, $t2, 1;"
        "   blt $t2, $t1, loop1;"
    );
}
```


4. What to Turn In

Submit each of the following items, clearly labeled and in the following order as a **single** PDF file. Also submit your **rtl_up** folder – that contains your modified MIPSfpga system and your bitfile – in a single zipped folder.

1. **DMA Engine design documents:** Submit any schematics, diagrams, etc. to explain your design. These should clearly, completely, and concisely describe your design. Add short sentences or paragraphs as needed to aid in the explanation. You will be graded on correctness, organization, and effectiveness at describing the design. Be sure to also briefly describe your arbitration scheme.
2. **DMA Engine modules:** Submit each of your Verilog modules related to the DMA Engine. These should be well-organized and clearly commented.
3. **Program:** Your clearly-commented and well organized C program for testing and using your system, including a 1-paragraph description of what your program does.
4. **Screenshots of DMA code running on your modified system:** a picture of your Nexys4 DDR board running your program. This should show screenshots of the OpenOCD window displaying the contents of both arrays before DMA access and the contents of the destination array after DMA access.

