CS-476:
Embedded System Design

Practical work 4

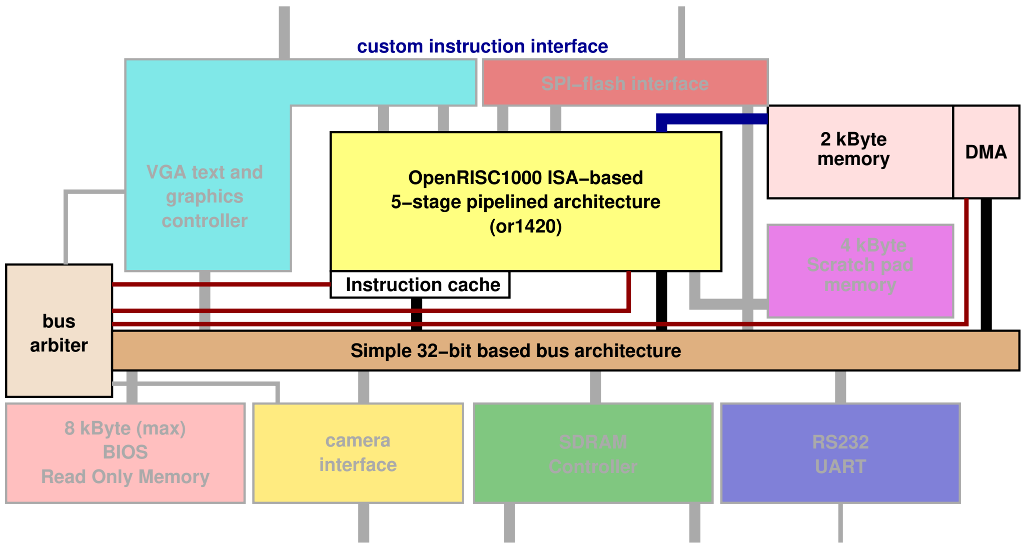
Build-in peripheral DMA-controller

Version:
1.0

Contents

1 Introduction

In the last PW we designed a slave-device on the bus-infrastructure. This PW will concentrate on the implementation of a 2kByte memory that is connected as a custom instruction. This 2 kByte memory will be connected to a build-in DMA-controller that allows to transfer data (32-bit words) to and from external memory to this local memory.


custom instruction interface
SPI-flash interface
VGA text and graphics controller
OpenRISC1000 ISA-based 5-stage pipelined architecture (or1420)
2 kByte memory
DMA
4 kByte Scratch pad memory
Instruction cache
bus arbiter
Simple 32-bit based bus architecture
8 kByte (max) BIOS Read Only Memory
camera interface
SDRAM Controller
RS232 UART

The DMA-controller is a master-device on the bus. For efficiency, the DMA-controller will support burst-mode.
The structure that we are going to design is a basic block that is often used in accelerators or streaming peripherals. Examples of a partial implementation of this block you can find in the camera interface:
`virtual_prototype/modules/camera/verilog/camera.v`
or in the hdmi-controller:
`virtual_prototype/modules/hdmi_720p/verilog/graphicsController.v`

2 Exercises

2.1 Prerequisites

Before starting this practical work, make sure that:

- You revisit the practical work on the grayscale custom instruction to make sure that you understand this interface.
- Go through the slides to familiarize yourself with the different bus-operations.
- Visit the slides of week 2 explaining how to describe memories in Verilog.

2.2 CI-attached memory (3.5 points)

In a first step we are going to design a 2kByte memory that is attached to the custom instruction interface. As the registers are 32-bit wide, this memory is arranged as 512x32-bit words. As we are going to allow for a DMA-controller in a later step, and we want that the CPU and the DMA-controller have direct access to the memory, we are going to use a dual-ported SSRAM. For this exercise you can connect all inputs of the second (B-)port to a constant 0, and leave all outputs open. The CPU will be connected completely to the first (A-)port. The module definition for this first exercise is:

```verilog
module ramDmaCi #( parameter[7:0] customId = 8'h00 )
                 ( input wire          start,
                                       clock,
                                       reset,
                   input wire  [31:0]  valueA,
                                       valueB,
                   input wire  [7:0]   ciN,
                   output wire         done,
                   output wire  [31:0] result );
```

For this first exercise we are going to use register A (`valueA`) as address, and register B (`valueB`) as data interface. As the memory has 512 entries, 9-bits are required to address it. We are going to use the bits `8..0` of register A as address. Bit 9 of register A is the write-enable bit. All other bits of register A need to be 0 to address the memory, hence bits `31..10` must be 0 to perform a memory operation.
Important: As the dual-ported memory is a synchronous memory, a read action takes 2 cycles and the write operation can be done in a single-cycle. We could circumvent this by clocking the first (A-)port on the negative edge of the μC clock, however, you are not allowed to do this (the explanation will follow later on).
Implement the complete system and test its proper functionality.

2.3 DMA to the CI-memory (8 points)

In this second exercise we are going to implement a DMA controller that allows to transfer data from the bus towards the ci-memory. This DMA-controller will connect to the second (B-)port of the dual-ported memory. This second (B-)port we are going to clock on the negative edge of the μC clock. This to prevent the situation that both the μC and the DMA-controller write simultaneously to the same memory location, hence we always have a defined behavior. To control the DMA we are going again to use register A (valueA) as address, and register B (valueB) as data interface. Bit 9 of register A is again the write-enable bit. Bits 12..10 of register A (valueA) are now the DMA-controller configuration registers, where:

A[12..10]	A[9]	Functionality:
000_b	0_b	Read from memory location A[8..0]
000_b	1_b	Write to memory location A[8..0]
001_b	0_b	Read the bus start address of the DMA-transfer
001_b	1_b	Write the bus start address of the DMA-transfer (B[31..0])
010_b	0_b	Read the memory start address of the DMA-transfer
010_b	1_b	Write the memory start address of the DMA-transfer (B[8..0])
011_b	0_b	Read block size (nr. of words) of the DMA-transfer
011_b	1_b	Write block size (nr. of words) of the DMA-transfer (B[9..0])
100_b	0_b	Read the burst size used for the DMA-transfer
100_b	1_b	Write the burst size used for the DMA-transfer (B[7..0])
101_b	0_b	Read the status register
101_b	1_b	Write control register
others	–	No function

To simplify your implementation you may, as for the previous exercise, have the read actions as 2-cycle operation, and the write-actions as single-cycle operation.
For the different registers:

- `The bus start address`: This is the address on the bus where the DMA-controller starts transferring the data. This address is auto-incremented during the DMA-transfer.
- `The memory start address`: This is the address on the ci-memory where the DMA-controller starts writing the data. This address is auto-incremented during the DMA-transfer.
- `The block size`: This is the number of 32-bit words that are transferred by the DMA-controller. Note that if this register holds a 0, the DMA-controller will transfer no data.
- `The burst size`: This is the number of 32-bit words plus one that are transferred during a burst transaction on the bus (hence a 0 indicates 1 word, and 255 indicate 256 words). So the total number of bus transfers required by the DMA-controller is given by: $\left\lceil \frac{blocksize}{burstsize+1} \right\rceil$.
- `The status register`: Bit 0 of this register indicates if a DMA-transfer is in progress (1) or if the DMA-controller is idle (0). Bit 1 of this register indicates if a bus-error occurred during the transfer (1) or if the transfer was successful (0).
- `The control register`: Writing a 1 to bit 0 of this register will start a DMA-transfer in case the DMA-controller is idle. This transfer will be from the bus towards the CI-memory.

Some points to think about:

- The DMA-controller has to accept the data coming from the slave the moment the `dataValid` signal is active, hence it will never activate the `busy` signal.
- In case the DMA-controller is active and a bus-error is detected (the `busError` signal is active), the DMA-controller must end the current transaction, set the bus-error status bit and return to idle.
- The DMA-controller needs to request for the bus through the arbiter. This can be done by using the signals `s_busRequests[27]` and `s_busGrants[27]` found in the top-level Verilog file: `virtual_prototype/systems/singleCore/verilog/or1420SingleCore.v`
- To reduce the critical path on the bus, all signals from the bus (hence all signals on the bus-in port) need to be registered by using flipflops.

Realize this module by extending the previous module `ramDmaCi` with the given functionality. Make a test program that shows the correct functionality. Note that the DMA-controller can only be used in polling mode.

2.4 DMA from the CI-memory (6 points)

Finally we are going to extend our module with the functionality that the data in the CI-memory can be transferred to the bus. For this purpose we are going to extend the `control register` with one bit. Writing a 1 to bit 1 of the `control register` will start a DMA-transfer from the CI-memory towards the bus-system. Some points to think about:

- Writing the value 3 to the `control register` should not activate the DMA-controller, as it is unclear in which direction the data should be transferred. The only valid values for the `control register` are 1 and 2.
- During the data transfer a slave can activate the `busy` signal, the DMA-controller should react correctly to this event.
- In case the DMA-controller is active and a bus-error is detected (the `busError` signal is active), the DMA-controller must end the current transaction, set the bus-error status bit and return to idle.
- It is up to the DMA-controller to transfer the correct number of words during a burst transfer.
- To reduce the critical path on the bus, all signals to the bus (hence all signals on the bus-out port) need to be registered by using flipflops.
- As the dual-ported memory is clocked on the negative edge of the μC clock, the memory acts like a memory with an asynchronous read, hence there is no latency during a read transaction.

Realize this module by extending the previous module `ramDmaCi` with the given functionality. Make a test program that shows the correct functionality. Note that the DMA-controller can only be used in polling mode.