Wiki »

EXERCISE: MM BUS

Exercise Goals

This exercise will through implementation of af MM-Bus compatible components introduce you to:

- The Avalon MM-bus protocol
- More test benches

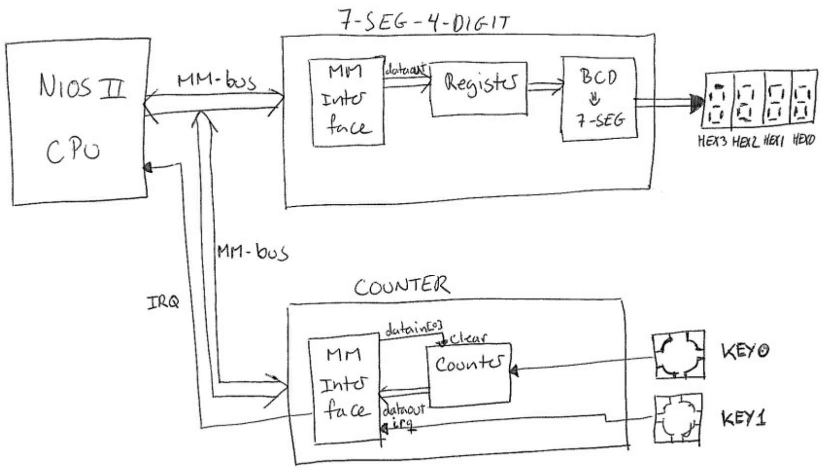
Prerequisites

- Quartus and ModelSim software must be installed and working.
- Avalon Interface Specification chapter 3 + 4
- Template code checked out from repository (exercise_mm_bus + qsys_basic_nios2)
- See Custom MM Bus Component
- See Making Qsys Components
- See Introduction to QSys Tool (Platform Designer = QSys)
- See My First NIOS Software tutorial

Introduction

The memory mapped bus is also know as a traditional parallel address / data bus. This type of interface is well-known from memory and other high-bandwidth peripherals that uses random access. . The interface is easy to use and can be accessed in both synchronous and asynchronous ways.

The purpose of the current exercise, is to create two "components" that will be capable of interfacing with the Avalon MM-bus. A block diagram is shown below:



7-Segment Component

The 7-segment / 4-digit driver can be written to by an avalon MM-bus host. It takes a 16-bit data input and displays the hex value on the four 7-segment displays. The software interface to this component consists of one internal register:

- Address 0x00: 16-bit value to be displayed (write-only)

Counter Module

The counter component has two inputs: One input is directly connected to the MM-bus interrupt interface and allows the module to interrupt the processor directly. The other input is connected to a counter that counts input events. The counter value can be read by a MM-bus host. The host can reset the counter value to zero by writing '1' to address 0x01 on the counter module. The counter component has the following registers:

- Address 0x00: 16-bit counter value(read-only)
- Address 0x01: Counter Reset (0x0001 = reset, otherwise not in reset)

Exercise Steps

Create an MM-bus Compatible 7-Segment Display Driver

In the repo / mm_bus_exercise you will find:

- *mm_bus_seven_seg_four_digit.vhd* contains skeleton code for the component.
- *mm_bus_seven_seg_four_digit_tb.vhd* contains a test bench template
- *bcd2sevenseg.vhd* contains a bcd-to-7-segment converter component
- *sim_avalon.vhd* contains functions to simulate Avalon MM-bus read/writes.

A) Create a new ModelSim project including the above mentioned files.

B) To investigate the write sequence, add an avalon_mm_write to the test bench using the functions found in sim_avalon. Note that sim_avalon is included as a package in the test bench. Compile, simulate and compare with the waveforms found in the Avalon Specification section 3.5.2. You should now have an idea of what to comply with.

C) Create a sequential process in the *mm_bus_seven_seg_four_digit* architecture that interfaces to the Avalon MM-bus and stores the write value in a register.

D) Connect the register to four bcd-to-7-seg components also instantiated in the architecture.

E) Simulate in ModelSim and note that the output changes when writing different values on the Avalon MM-bus.

F) Copy the content of the qsys_basic_nios2 folder to your exercise folder and open the project i Quartus. When the project has opened, start the tool "PlatformDesigner" from the tools menu.

G) When Platform Designer has loaded, open CPU_system.qsys file from the file-open menu. This will load a basic system with CPU, SDRAM, UART, GPIO a.o.

H) Follow the guide Custom MM Bus Component to create a custom component in QSys/Platform Designer, based on your mm_bus_seven_seg_four_digit design.

I) Add the new component to the CPU system in Platform Designer. Connect clock, reset and bus and export external connections, ex. HEX interface, by clicking on the grayed-out text: "double-click to export..". Press Generate HDL, select VHDL and "no .bdf" and start system generation. When completed successfully, copy the instantiation template from the menu Generate-Show Instantiation.. Go to top.vhd in Quartus and update the interfaces accordingly. The HEX outputs of the soc system must be connected to the external interface of the vhd design entity. Build the design and program the FPGA. See Making Qsys Components for further details

J) Open the NIOS II Software Build Tool. You can set the workspace path to the root of all your DSPC exercises. Select File-New-"Nios ii application and BSP from Template". Navigate to the .sopcinfo file located in the qsys_basic_nios project folder. Name the project, select "Hello world" and press finish. The Board Support Package (BSP) contains helper functions and libraries to be used in your application.

K) Build the hello world application, right-click on the project in the navigator and select "run as"-"Nios II Hardware". Refresh connections (Target Connection) and press run. This should fire up your application on the board. NOTE! Quartus' programming tool must be closed when doing this or I may occupy the JTAG connection.

L) Modify the application to write values to your HEX displays. You should include *system.h* and *io.h* (located i bsp and bsp-HAL-include). *system.h* contains defines for base addresses and *io.h* provides io access functions that you can use:

```
// From io.h

// Read Functions
IORD_32DIRECT(BASE, OFFSET)
IORD_16DIRECT(BASE, OFFSET)
IORD_8DIRECT(BASE, OFFSET)

// Write Functions
IOWR_32DIRECT(BASE, OFFSET, DATA)
IOWR_16DIRECT(BASE, OFFSET, DATA)
IOWR_8DIRECT(BASE, OFFSET, DATA)
```

NOTE! 32/16/8 bit access influences byteenable and offset adds 1-4 bytes to the base address according to access size. See My First NIOS Software tutorial for more details

Create an MM-bus Compatible Counter

- mm_bus_counter.vhd contains skeleton code for the component
- mm_bus_counter_tb.vhd contains a test bench template
- sim_avalon.vhd contains functions to simulate Avalon MM-bus read/writes.

A) Create a new ModelSim project including the above mentioned files.

B) Copy the Avalon MM-bus interface process from the 7-segment component into the mm_bus_counter.vhd file.

C) Modify the process to also support read access.

D) Create a counter sequential process. The process shall count events on the *input_counter* input. The register value must be written to the Avalon MM-bus during read accesses to address 0x00 on the counter. Writing '1' on the Avalon MM-bus to address 0x01 should reset the counter.

E) Edit the test bench to read counter values (you may use the functions in avalon_sim package: avalon_mm_read(..)) and reset the counter (avalon_mm_write(...)).

F) Validate your design in ModelSim

G) Invert the input_irq and connect it to ins_irq0_irq

H) Create a new component i Platform Designer, add it to your design, create HDL, update top.vhd, build- and program the Quartus project. Perform the steps H-L from the previous section. To register an interrupt, you must use *alt_ic_isr_register()* from "sys/alt_irq.h".