UCD School of Electrical and Electronic Engineering

EEEN40280
Digital and Embedded Systems

System on Chip

In this assignment you will build a complete embedded system on an FPGA, with an ARM Cortex-M0 processor, connected to various peripheral blocks using an AHB-Lite bus, running software written mostly in C. Most of the peripheral blocks are provided, but you will probably want to design one or two more. You will also have to integrate the blocks that are provided into the system, making the necessary connections to the bus and other signals.

The eventual goal is to collect information from an accelerometer, available on the Digilent Nexys-4 FPGA board, and display it in some useful way.

Project Setup

Download the DES_SoC.zip file on Blackboard, and unzip it into the correct folder on the lab computer : Documents\EmbeddedSystems\???day. Take care to use the folder names when extracting the files – this operation builds a directory structure for the entire assignment.

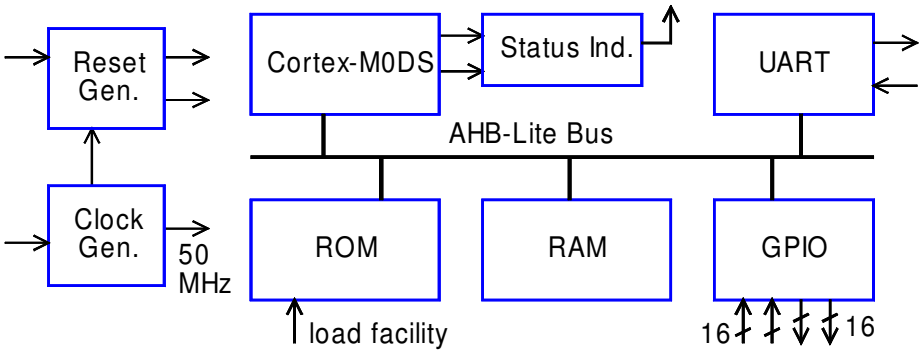
Verilog files are in DES_SoC\Hardware\Source and DES_SoC\Hardware \Testbench, program files are in DES_SoC\Software, documents and data sheets are in DES_SoC\Documents.

Start the Xilinx Vivado software, and open the project DES_M0_SoC – you will find the project file in the DES_SoC\Hardware folder.

When the project opens, you should see the Verilog modules arranged in the appropriate hierarchy in the Project manager window. Modules that are not yet included in the design will be shown separately.

Block Diagram

The top-level block diagram of the test system that you will build is shown below. Many of the blocks are provided – these are listed with brief descriptions below. More detailed information on some blocks in in the DES_SoC\Documents folder.



AHBliteTop.v
This is the top-level module, in which all the other modules should be instantiated. It is provided in incomplete form – the data memory, GPIO and UART peripherals are missing. A simple testbench is provided, which provides the clock and reset signals needed to simulate the system.

clock_gen.v
Clock generator – provides 50 MHz clock for all the hardware and the AHB-Lite bus, using the 100 MHz clock input from the oscillator on the Nexys-4 board.

reset_gen.v
Reset generator. Provides hardware reset signal at power on or if the CPU RESET button is pressed, holds this signal until the clock generator is running. Also provides a separate reset for the processor and AHB-Lite bus, which conforms to the Cortex-M0 reset requirements, and can be requested by software.

status_ind.v
Status indicator – uses two multi-colour LEDS on the Nexys-4 board to display status signals.

Colour	**LD17 – on the left**	**LD16 – on the right**
Blue	Processor and AHB bus reset active	Processor sleeping (wait for interrupt)
Red	Processor lockup	ROM loader active (see below)

CORTEXM0DS.v
This is a wrapper for the Cortex-M0 DesignStart processor, provided by ARM. It makes the CPU registers visible in simulation. This module has been synthesised already, to save time.

AHBDCD.v
Address decoder, defines the address map. It selects the appropriate slave on the AHB-Lite bus in response to the address input. It also controls the multiplexer block, below. You will have to modify the decoder block as you add more slaves to the system.

AHBMUX.v
Multiplexers for signals from the slaves. You should not have to modify this block, but you will have to connect more slave signals to it.

AHBprom.v
Program memory, 32 kByte, read-only from the AHB-Lite bus. The memory used is made up of block RAM components on the FPGA. The memory is configured with the machine code for a simple test program. However, the memory block includes special hardware (ROM loader) to allow new program code to be loaded through the serial port after the FPGA has been configured. See instructions later.

AHBbram.v
Data memory, 16 kByte, read-write, with byte and half-word writes possible. The memory used is made up of block RAM components on the FPGA.

AHBgpio.v
General-purpose input-output block. Provides two 16-bit output ports and two 16-bit input ports, with byte writes possible to the output ports. Full details, including RTL diagram, are in a separate document, in the DES_SoC\Documents folder. A testbench for this block is also provided, which simulates some AHB transactions.

AHBuart2.v
UART – asynchronous serial interface block. Transmits and receives at 19200 bit/s, in groups of 8 data bits with one start bit, no parity and one stop bit. The transmit and receive paths each have 16-byte buffers or queues, and a status register provides bits to indicate if these are full or empty. Each of these bits can be enabled to cause an interrupt, by setting appropriate bits in the UART control register. Full details are in a separate document. A testbench is provided.

Constraints file
Along with all the hardware blocks, a constraint file is provided: Nexys4_Master.xdc. This defines the pin numbers and signal voltages for all the connections to the FPGA. The signals not used in the test system are commented out. You will need to modify this file later, to include more signals.

Block Simulation

In Vivado, select the sim_gpio simulation. Click on Run Simulation and choose a behavioural simulation.

Expand the simulation timing diagram window, and zoom to fit the entire waveforms. You should be able to see some AHB transactions being performed by the testbench – writing to registers and reading from registers in the GPIO block. Examine the Verilog testbench to see what is supposed to happen. Examine the documentation and the Verilog description for the GPIO block to see how it works.

System Integration

The system that you have been given is not complete – some of the slaves are not connected to the AHB-Lite bus. You need to build a complete system, as shown on page 1, and then run some test programs on it.

Add RAM block
Edit the top-level module, and add the AHBbram block to the system. Instantiate the block and connect its ports to the appropriate AHB bus signals. There are wires already defined for its slave select signal and for its output signals – these wires are already connected to the address decoder and multiplexer blocks. You need to identify the wires and connect them to the ports on the AHBbram block.

If you have done this correctly, you should see the RAM block appear in the hierarchy. If you expand it, you will see a block RAM, which was generated from the IP catalog.

At this stage, you should be able to Open Elaborated Design, and see the block diagram of the system so far. You will see some warnings about unconnected ports, and you will also see these on the block diagram – these are for the blocks that you have yet to add.

Add GPIO block
Now add the GPIO block to your system. Instantiate it and connect its bus ports to the bus signals as before. This time, you will have to create wires for the signals specific to this block.

You will also have to connect the other ports on the block. The test program expects the gpio_out0 port to connect to the LEDs on the Nexys-4 board. However, the ROM loader also uses the LEDs when it is active, so there is a multiplexer to allow the two blocks to share the LEDs. You should connect the gpio_out0 port to the multiplexer – the wires are already there.

Connect the switches to port gpio_in0.

Connect the buttons to the least significant bits of gpio_in1, with 0s on the other inputs.

You can check your work by opening the elaborated design again…

Add UART block
Next add the UART block. The software uses this block to communicate with the PC.

Its serial transmit and receive ports connect to ports (with different names) of the top-level module. You also need to connect its interrupt request line to one of the IRQ inputs of the processor – the software expects this interrupt at IRQ bit 1.

Address decoding
Now modify the address decoder block to position the new slaves at the correct places in the address map. The address decoder is designed to check only the 8 most significant bits of the address, so each slave gets a 16 MB range of addresses – far more than it needs.

You should see that the program memory (ROM) is already located at address 0x00000000, and the data memory (RAM) at 0x20000000. You need to add logic to put the GPIO block at 0x50000000 and the UART block at 0x51000000.

Check the elaborated design again. Note that you can expand any block and see what is inside it if you wish.

Implementing Your System on FPGA

In Synthesis Settings, set fsm extraction off. Then Run Synthesis. You can select to run it in the background if you want to get on with other tasks while it is running.

When the synthesis has run, view the synthesis report. You will find many warnings, hidden among even more information messages. You can also see these warnings on the Messages tab at the bottom of the Vivado window. You need to check that all of these are acceptable before you proceed. If you are not sure, ask for help!

Then Run Implementation. This may take some time.

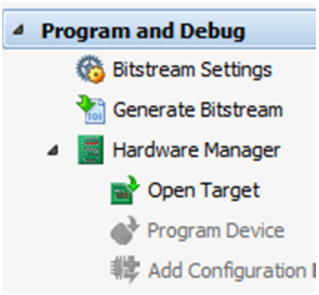
While you are waiting, connect the Nexys-4 board to the PC and switch it on. You should see it demonstrate some of its features as a self-test.

When Implementation completes, with no warnings, Generate Bitstream. This produces the `.bit` file that you need to download to the FPGA to configure it with your design.

HyperTerm
Run HyperTerm – Start menu, All Programs. On the File menu, open the connection file in the project folder: Nexys4.ht. This attempts to connect to a device on port 5 – if it fails, you may need to select a different port. Leave the window open.

If you need to make a new connection, choose a port with a number greater than 3. Configure the connection for 19200 bit/s, 8 data, no parity, 1 stop bit, no handshake. Save the settings.

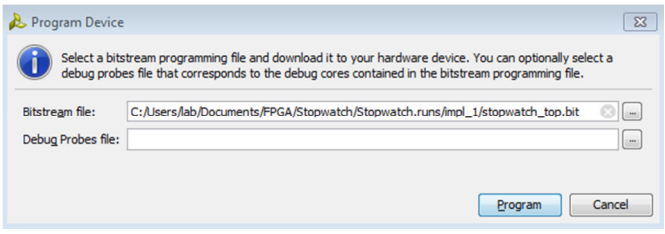
Configure FPGA
To configure the FPGA, use the Hardware Manager.



Select Open Target. The first time you do this, you should choose Open New Target. Accept the defaults, and the Nexys4 board should be found and identified. The FPGA on the board should also be identified.

If you do this again, when you click Open Target you should see the previous FPGA board listed – just select it. If there is more than one board, the top one is usually the most recently used – there is a label on the back of each board with the last six digits of the board number.

When you have opened the target board, click Program Device, and select the only device offered. The correct bitstream file will usually be offered, as shown here. It should have the name of your top-level module, with extension `.bit`. You do not need a Debug Probes File.



If all is correct, click Program.

Your design should now be on the FPGA, and ready for testing. If you switch off the board, the configuration will be lost, and you will have to download the `.bit` file again – Program Device. There is no need to repeat synthesis and implementation if the design is the same.

Testing

When the FPGA has been configured, press the CPU RESET button. You should see a message appear in the HyperTerminal window. If you type characters here, they will be echoed back to you. When you press return, the whole sentence will be repeated, with the letters modified.

The LEDs on the Nexys-4 board should show each character code as it is received. When you press return, they will flash a pattern that depends on the switch settings. You will also notice the right-most status LED (LD16) showing blue most of the time – this indicates that the processor is sleeping, waiting for an interrupt.

Software

Run the uVision4 software. Open the DES_M0_SoC_Basic_Uart project located in DES_SoC/Software.

Check the project options – on the Target tab, you should see the ROM and RAM areas defined. Check the path to the .sfr file – it may need to be changed to point to the file on your computer.

Examine the code. This is the program that was already loaded into the program memory in your system, and is now running on the processor in the FPGA on the Nexys-4 board. You should be able to understand why the system is behaving as it is.

Modify the program

When you understand the software provided, change it. Make it send a different welcome message through the serial port, so you will recognise the new version. Modify it so that it reports the number of characters that you have typed, and also reports the state of the switches as a number.

Build your new program. You should find an output file called ROMcode.hex in the Software folder. Examine this – it contains the 32-bit words that should be in the program memory.

The first word defines the initial value of the stack pointer. The second is the address of the first instruction to be executed after reset. The 18th word should be the address of the interrupt handler for the UART interrupt.

Now download and test your modified program.

Download to program memory

On the Nexys-4 board, press the black pushbutton marked BTNU. While holding this down, press and release the reset button. Then release BTNU. This activates the ROM loader hardware, which also keeps the processor in reset. You should see the status indicators showing blue and red.

In HyperTerm, on the Transfer menu, select Send Text File… Select the ROMcode.hex file – you will have to change the type of file to "All Files" to see this. Opening the ROMcode.hex file will cause HyperTerm to send the file, byte by byte. The ROM loader should receive it – you should see the LEDs on the Nexys-4 board displaying a count of the words received.

When the file has been sent, click in the HyperTerm window and press Q on the keyboard. This tells the ROM loader that the transfer is complete. The status indicators should change as the processor starts running your program.

Interim Report

When you have completed the initial tasks, and before you start on the system design, submit your modified Verilog files: AHBliteTop.v and AHBDCD.v for review. Also submit your modified program file, main.c for review. Use Notepad++ to print just the **relevant sections** of these files, staple them together and write your names on the first page.

These pages will be returned with feedback, but will not be graded. One submission per team is sufficient.

System Design

The accelerometer on the Nexys-4 board is an Analog Devices ADXL362. The data sheets for the accelerometer and for the Nexys-4 board are in the DES_SoC\Documents folder, and also on Blackboard.

Look at the data sheet for the accelerometer. There is no need to read every word at this stage – you just need to get an overview of what information is available from the accelerometer, and how you can get access to this information.

Your assignment is vague: to collect information from the accelerometer and display it in some useful way. You need to decide what information you will collect and how you will display it.

As you have seen in the sample program, you can send information to the PC, to be displayed in HyperTerm. You can also display information on the line of LEDs on the Nexys-4 board. You could display information in a more user-friendly way on the 7-segment displays on the board. You could use the switches on the board to decide which information to display…

Architecture

When you have decided what you want to do, you can start designing a system to do it. First break the problem into a few large blocks, and think about the information flows between the blocks. Then design one block at a time – if necessary, divide a block into smaller blocks…

For example, you will have to connect the FPGA to the accelerometer. The accelerometer uses a SPI interface – it acts as a slave. So your design will need a SPI master interface to communicate with the accelerometer. This will probably be one of the blocks in your design.

You could design a hardware SPI interface to connect to the AHB-Lite bus – this could handle all the SPI signals, with the correct timing, and provide some registers that the software could access to control the interface and to send and receive data. Alternatively, you could connect the SPI signals to some input and output signals on the GPIO block, and handle the interface in software. In this assignment, the hardware solution would be preferred, but if you cannot design hardware, you could choose the software solution.

A similar decision would arise if you wanted to use the multiplexed 7-segment display…

Block design

When you have completed the top-level design, divide the block design tasks among the members of the team. For example, somebody needs to do the detailed design of the SPI master interface, either as hardware or as a set of software functions.

You should also make a plan that will allow the team to complete the rest of the work in the time remaining. Note that all laboratory work must end by Friday 2 December – the lab will be closed after that date.

If you wish, you can discuss your plan with the staff in the lab at this point, to see if it makes sense or to get advice on possible improvements.

Final Report

You will work on this assignment as part of a team, but you must submit your own report. Block diagrams, RTL diagrams and flow charts may be shared within the team, with credit given to whoever created the diagram. The text of the report should be your own.

The deadline for all reports is 16:00 on Wednesday 7 December. Standard penalties apply for late submission. You should submit a paper copy of your report, and also upload it through the submission channel in Blackboard. Both versions must meet the deadline, but a small extension will be allowed for paper reports arriving by post.

Your report should start with the declaration sheet that will be provided in the laboratory and on Blackboard. It should include:

- An overview of your design, with details of the information that you can display, the user interface, etc. Also details of the design process: the main blocks that you identified, your decisions on how each block would be implemented, etc. This section should make clear who worked on each block.
- Details of the blocks that you designed. For software blocks, explain how you broke the problem into separate functions, and what each function does. Use a flow chart or diagram if you wish. For hardware blocks, give an RTL diagram, and explain how the block works and why you designed it that way.
- A brief description of the tests you performed, to verify that your system works as intended. For hardware blocks that you designed, include an example timing diagram from simulation.

There is no need to include all the Verilog and program files in your report. However, you may want to include parts of these files, to show how you solved a particular problem.

One member of each team should collect all the source files used in the assignment: Verilog design files, testbench files and program files. Combine all of these into one zip file. Do **not** include the entire project folder – that will be many megabytes in size, and only a few kilobytes are needed.

This zip file should be uploaded through a separate channel in Blackboard. Keep the filename short, but include something to identify it as yours – maybe one name or a set of initials. There is no need to include a student number – Blackboard will add that automatically.

Grading

Grading will take into account:

- The capability of the system that you have designed: what can it do?
- The design choices that you made: for example, a hardware SPI interface would be preferred to a software solution;
- The quality of the implementation: we expect well-designed hardware and well-structured software;
- The quality of the documentation: the report and the comments on the Verilog and C code;
- The number of students in the team – a larger team will be expected to achieve more.