Informatik 12
Cauerstr. 11
91058 Erlangen

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Reconfigurable Computing
Lab 4

Problem 1 (Softcore Processors and Hardware Acceleration)

Overview: A softcore processor is a hardware description language (HDL) model of a specific processor (CPU) that can be customized for a given application and synthesized for an ASIC or FPGA target. In many applications, soft-core processors provide several advantages over custom designed processors such as reduced cost, flexibility, platform independence and greater immunity to obsolescence. Embedded systems are hardware and software components working together to perform a specific function [2]. Usually, they contain embedded processors that are often in the form of soft-core processors that execute software codes and in heterogeneous multiprocessor system-on-chip (MPSoC) scenario, a dedicated hardware accelerator is often used to speedup applications.

In this exercise, we demonstrate the advantages of controlling a hardware accelerator using a soft-core processor by simulating a typical RISC processor (LEON3) system on a Xilinx Zynq device. The DDR3 memory attached to the Cortex-A9 processor system (PS) is used as LEON3 memory, and accessed through a custom AHB/AXI bridge (ahb2axi.vhd) and using a LEON3 processor, we will create and attach a hardware module to accelerate an edge detection algorithm.

Before starting, an example of a 2-D convolution (edge detection) is specified in Eq. (1). Here, the convolved output pixel at location (m,n) for a given window size of $w_h \times w_v$ is computed as follows:

$$y(m,n) = \sum_{i=\lfloor -w_h/2 \rfloor}^{\lfloor w_h/2 \rfloor} \sum_{j=\lfloor -w_v/2 \rfloor}^{\lfloor w_v/2 \rfloor} h(i,j) \cdot x(m-i, n-j) \tag{1}$$

where x represents the input pixel stream and h represents the convolution window (laplace) which the coefficients are defined as bellow:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Your tasks in this laboratory are described, as follows:

Work Description: For implementing a Sparc LEON3 core, we use the GRLIB IP Library. The GRLIB is an integrated set of reusable IP cores, designed for system-on-chip (SOC) development. The IP cores are centered around a common on-chip bus, and use a coherent method for simulation and synthesis. The library is vendor independent, with support for different CAD tools and target technologies. A plug&play method is used to configure and connect the IP cores without the need to modify any global resources.

The GRLIB is designed to be a bus-based system, i.e. it is assumed that most of the IP cores will be connected through an on-chip bus. The AMBA-2.0 AHB/APB bus is used as the common on-chip bus. Figure 1 shows an example of a LEON3 system designed with GRLIB [1].

All the next folders mentioned in this tutorial are accessed from this location: `/scratch-local/rc/lab04/LeonCore/grlib-gpl-1.3.7-b4144/designs/leon3-digilent-xc7z020`.

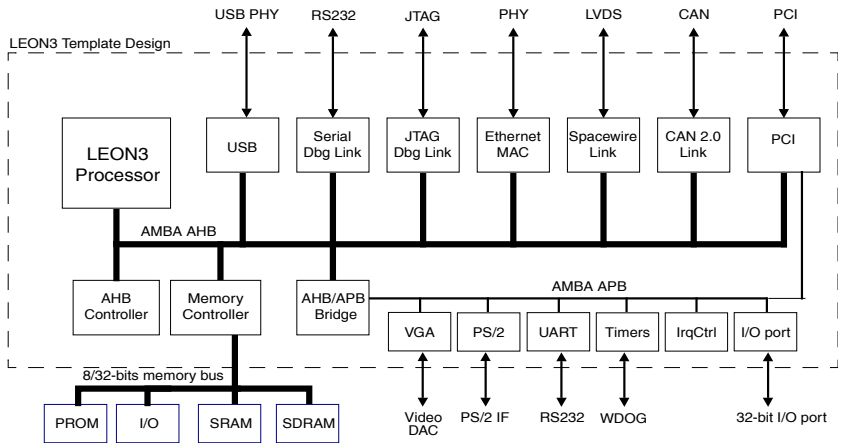

Figure 1: An example of a LEON3 system designed with GRLIB IP library.

1. **Software**

   Let's compile a simple `Hello World` example.

   - Goto folder `software`
   - Compile the program `hello_world.c` using a `sparc` compiler:
     `sparc-leon-elf-gcc -Wall -o hello_world.exe hello_world.c`
   - Copy the binary file into the RAM:
     `sparc-leon-elf-objcopy -O srec --gap-fill 0 hello_world.exe ../ram.srec`

2. **Simulation**

   Here, we will test the software simulating the design with Modelsim.

   - In the folder `/leon3-digilent-xc7z020` load the Modelsim:
     `module load modelsim/10.2c_x86_64-pc-linux`
   - Remove all temporary files: `make distclean`
   - Compile the design: `make vsim`
   - Start Modelsim: `vsim testbench.mpf` or `make vsim-launch`
   - Start the simulation executig the script `do start_sim` at the prompt of the simulator. Note, if you change any HDL file you need to recompile the design again
   - Run the simulation: `run -all`

3. **Counter**

   Now, using the Advance Peripheral Bus (APB), let's create and connect a counter module that will be used to measure the performance of both software and hardware computation. Note that this module will be accessible via software.

   - Access the folder `vhdl`. Here, you will see the file `counter.vhd`. Complete the file and add it in the simulation
   - In the file `leon3mp.vhd` set the constant values `CFG_COUNTER` and `COUNTER_INDEX` to `1` and `4`, respectively
   - Write a software to read the counter values. For that, you can simply complete the program `counter.c` located in the folder `software`. Then, compile the program and load it into the RAM

- Compile the design
- Before starting the simulation. Open the script `start_sim` and uncomment the lines to show the waveforms of the counter
- Repeat the simulation. The expected output is presented in Figure 2

4. **Hardware Accelerator**

   Similar to step 3, we will connect the hardware accelerator for the edge detection using APB.

Figure 2: Integration of Counter and Hardware Accelerator Modules on LEON3 using APB

```
VSIM 171> run
# LEON3 Xilinx Zedboard Demonstration design
# GRLIB Version 1.3.7, build 4144
# Target technology: zynq7000  , memory library: zynq7000
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area at 0xfff00000, 1 Mbyte
# ahbctrl: AHB masters: 2, AHB slaves: 8
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Aeroflex Gaisler        LEON3 SPARC V8 Processor
# ahbctrl: mst1: Aeroflex Gaisler        JTAG Debug Link
# ahbctrl: slv0: Aeroflex Gaisler        Generic AHB ROM
# ahbctrl:       memory at 0x00000000, size 1 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Aeroflex Gaisler        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# ahbctrl: slv2: Aeroflex Gaisler        LEON3 Debug Support Unit
# ahbctrl:       memory at 0x90000000, size 256 Mbyte
# ahbctrl: slv3: Aeroflex Gaisler        Xilinx MIG DDR2 Controller
# ahbctrl:       memory at 0x40000000, size 256 Mbyte, cacheable, prefetch
# ahbctrl: slv6: Aeroflex Gaisler        Test report module
# ahbctrl:       memory at 0x20000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: Aeroflex Gaisler        Xilinx MIG DDR2 Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Aeroflex Gaisler        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Aeroflex Gaisler        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Aeroflex Gaisler        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv4: FAU-Computer Science 12 COUNTER
# apbctrl:       I/O ports at 0x80000400, size 256 byte
# apbctrl: slv5: FAU-Computer Science 12 EDGE DETECTION
# apbctrl:       I/O ports at 0x80000500, size 256 byte
```

Figure 3: Counter and Hardware Accelerator Modules on LEON3

- In the folder `vhdl` are the files `top_for_edge_detection.vhd` and `edge_detection.vhd`. Complete these files and add them in the simulation
- In the file `leon3mp.vhd` set the constant values `CFG_COUNTER` and `COUNTER_EDGE_DETECTION` to `1` and `5`, respectively
- Complete the program `edge_detection.c` located in the folder `software`. In this same file you also have to complete the instructions for starting the hardware accelerator.
- In order to achieve a better performance, compile the software using the optimization flag `-O2` that is used for optimizing the software execution
- Compile the design
- Before starting a simulation. Open the script `start_sim` and uncomment the lines to show the waveforms of the hardware accelerator
- In this step, the counter as well as the hardware accelerator are integrated in the system as depicted in Figure 2
- Repeat the simulation. The expected output is presented in Figure 3
- At the prompt of the simulator, the number of cycles needed to execute both software and hardware is shown. Thus, answer how faster is the hardware accelerator in comparison with the software execution?

References

[1] Aeroflex Gaisler. GRLIB IP Library User's Manual, 2014.

[2] J. Tong, I. Anderson, and M. Khalid. Soft-core processors for embedded systems. In *Microelectronics, 2006. ICM '06. International Conference on*, pages 170–173, Dec 2006.