Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

System Construction Course 2018,

Assignment 11

Felix Friedrich, ETH Zürich, 2018

Multicore Computing on FPGAs – Implementation of the Simon Game.

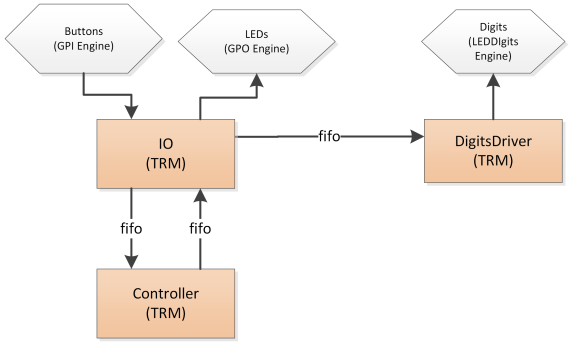
Lessons to Learn

- Understand the features and limits of a message passing architecture.
- Understand how software code can be mapped to and deployed on a multicore architecture on FPGA.
- Understand how configurable hardware can be adapted to satisfy software requirements.

Introduction

The Simon Game was a very simple electronic game popular in the early 1980s. The objective of the player was to reproduce a sequence of light signals by pushing buttons in the order how they were previously presented by the computer.
In the original game, the lighting of the buttons was accompanied by audio signals, which significantly helped to memorize the order of the sequence. As a side effect of this lab, you can implement the game on a Spartan 3 FPGA, solely using buttons and LEDs.

We work with a message-passing architecture on an FPGA. The system consists of three processors – Tiny Register Machines (TRMs) – that are connected to each other over FIFOs. In addition, the processors are wired to hardware signals such as clock, reset and interaction interfaces such as LEDs, a 7-digit LED display and buttons. We provide the following architecture for this lab.



You may wonder why there is a separate component for the 7-digit display. The reason is that controlling the 7-digit display, on a first glance, is not so trivial, so we solved it with a little software loop being implemented on TRM.

It is your task to implement a variant of the game that has been roughly described above. But first and foremost, you are asked to solve some technical problems with the given implementation.

Preparation

1. Update your repository.
2. Open a console in directory assignments/assignment11
3. Extract the contained file `source.zip` in the same directory.
4. Linux users do
   - Extract `linux64.zip`.
   - Check that file `linux64/LoadHWL.txt` provides the correct Xilinx bin path. The default is `AcXilinx.SetIseBinPath "/opt/Xilinx/14.7/ISE_DS/ISE/bin/lin64/"`
   - `chmod +x oberon64`
   - Start the Oberon/ActiveCells toolchain using `./oberon64 run linuxSettings`.
5. Windows users do
   - Extract `win64.zip`.
   - Check that file `win64/LoadHWL.txt` provides the correct Xilinx bin path. The default is `AcXilinx.SetIseBinPath "c:/Xilinx/14.7/ISE_DS/ISE/bin/nt/"`
   - Start the Oberon/ActiveCells toolchain using `oberon64 run windowsSettings`.
6. We have prepared shortcuts for the most important commands, loaded automatically from the shell from file Shell.Alias:
   (a) Compilation. Shell command: `compile`, a shortcut for executing the compilation commands in file CompileGame.txt.
   (b) Loading the Hardware Library. Shell command `load`, a shortcut for executing the compilation commands in file `LoadHWL.txt` contained in directory `linux64` or `win64`.
   (c) Building the hardware. Type `build`, a shortcut for `AcHdlBackend.Build --target="Spartan3StarterBoard" -p="ISE" --outputPath="Ac3Projects" "Game.Toplevel"`.
   (d) If you change the hardware library, you have to force a rebuild of the application, which can be done using `rebuild`.
7. Deploy the application to the FPGA using impact (bitstream is in `Ac3Projects/Toplevel/Toplevel.bit`.
8. Once uploaded to the FPGA, the program should give LED-feedback on button pushes and should show the number of button pushes on the 7-digit display. Only the three buttons to the right are active. The left most button serves as reset signal.
9. If you want to make use of the A2 GUI (deprecated at this stage of the course), you can start it using command `run a2`.

Task

The 7-digit LED display is currently programmed with a software loop on a separate core. When a digit is selected for output, all other digits currently switch off. This is due to the mechanism how the digits are controlled. *Please read Chapter 3 of the user manual of the spartan board in order to understand how it works*. Only the fact that it happens so often (and potentially the luminescence of the digits) can make this invisible to a human.

First modify the software such that the flickering on the display goes away.[1]

[1] We left the flickering in on purpose in order to visualize the mechanism.

Then implement a driver for the 4-digits LED display in order to get rid of the software loop. Modify the existing LED digits hardware module in file HardwareLibrary/IO/LEDDigits.v. Do not change its semantics and interface, only insert a loop in hardware that replaces the software loop. You need to register the digits data coming from the user but other than that the interface can stay as it is. Once you have changed this hardware module, you need to rebuild. If you get error messages, open ISE and continue from there.

In order to try your modifications of the hardware code, we suggest you to reduce the Game module to just one cell. The reason is the great amount of time saving for synthesis and routing. Once you know it works, you can integrate the solution in the game module as it was and remove the digits TRM Cell.

When you apply hardware changes, you may want to work within ISE. The project file can be found in `Ac3Projects/Toplevel/ISE/Toplevel.xise`. Once you have applied all changes in ISE (and rebuilt the hardware), do not forget to run `build` within the Oberon shell (in order to patch the instruction stream into the bit files).

Implement the game (optional)

1. The game starts with one single ternary signal (left, middle or right) displayed on the LEDs. Duration of signal display should be, say, 300ms.
2. When a sequence of signals (alternating corresponding LEDs on and all LEDs off) has been presented, the user is to repeat the sequence by button pushes.
3. If the entered sequence has been entered correctly, the length of the sequence is increased by one.
4. The 7-digit display should show the length of the correctly entered signal sequence.

You are free in the choice of how to implement the game. The choice of interconnect is motivated by the minimal requirements of this game. To add more channels between the cores is not possible because of the very limited resources of the exercise Spartan3 board. However, if required you can implement further logical channels in software, for example from input to display core.

Try to make use of the fact that you have three independent cores. For example, you may want to implement a protocol for processing whole sequences of signals on the display core or to send a sequence of signals to the input core for which the button input is verified autonomously.

Known Problems

1. when the cable drivers do not work: run impact with super user account. Make sure the cable is connected (physically and also logically in a VM) before trying to connect from impact.
2. when the build script reports problems when calling data2mem: make sure you have enough main memory allocated (to your virtual machine). You need more than 1GB.

Documents

- Spartan-3 Starter Kit Board User Guide. File documents/DigilentSpartan3/S3BOARD_RM.pdf
- Active Cells papers in directory documents/ActiveCells.
- Slides from the lecture homepage.