System Construction Course 2015,

Assignment 11

Felix Friedrich, ETH Zürich, 8.12.2015

Multicore Computing on FPGAs – Implementation of the Simon Game.

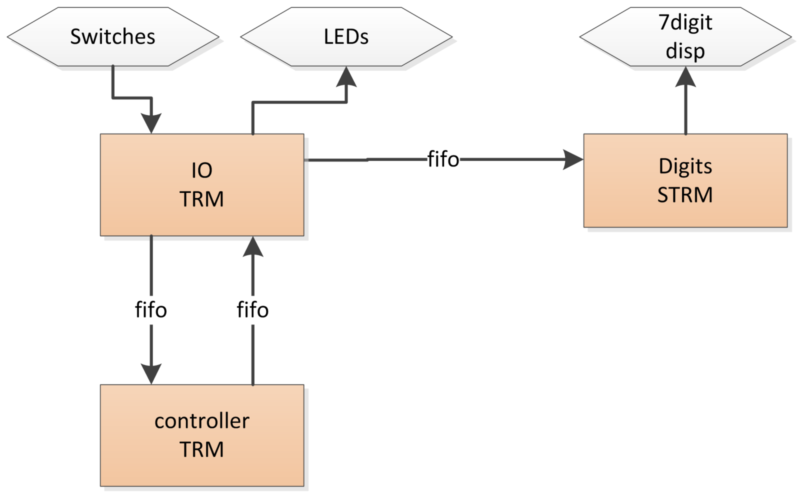
Lessons to Learn

- Understand the features and limits of a message passing architecture.
- Understand how software code is mapped to and inserted in 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 TRMS, an even simpler variant of the Tiny Register Machine.[1]

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. Please update your repository in order to retrieve a version of A2 that contains the compiler and an example module.
2. Windows users are required to
   (a) Unzip the A2 system from ActiveCells/Windows/A2.zip.
   (b) In order to be able to drive the hardware compilation from within A2, please make sure that the Xilinx tools such as data2mem are available via (Windows) search path. Usually this means adding `C:\Xilinx\14.7\ISE_DS\ISE\bin\nt64` to the environment path variable.
3. Linux Aos Users are required to
   (a) Install UnixAos rev 5500 found in `ActiveCells/UnixAos/` and use the work folder found in the same directory as your Aos work path.
   (b) Compile the compiler. Please open file `oc/linux.Fox.Tool` and execute the compilation command and restart A2.
   (c) In order to patch the bit-stream and upload it to the Spartan3 board, a separate command has to be executed after the compilation took place.
   The command `TRMTools.BuildHardware` does not yet work within LinuxAos. Please execute the script Game.sh to patch and upload the stream.
4. A build file for the provided game skeleton can be found in `Assignment11/Assignment.Tool`. It consists of a compilation command to generate hardware and patching scripts. For WinAos users, hardware compilation requirements are automatically detected and resolved by the execution of command `TRMTools.BuildHardware`: the tool-chain automatically patches what is necessary in the hardware stream and uploads the bitstream to the FPGA.
5. 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.

Tasks

1. You may have noticed that a single button push sometimes result in a bunch of signals because of some jitter on the electrical contacts. It is the first task of this exercise to get rid of this jitter. Please modify the code on the input core accordingly. Hint: `Timer.Get(VAR time: LONGINT)` can be used in order to get timer ticks. Constant `Timer.ms` contains the number of ticks per ms.
2. 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

[1] Interrupt handling, which we do not use here anyway, and multiplication engine has been stripped from TRMS. The reason why we need this here is the very limited resources of our Spartan3 board.

(and potentially the luminescence of the digits) makes this invisible to a human. Modify the hardware such that the 7-digit LED display does not have to be programmed with a software loop. Then you can get rid of the separate TRMS component for this. This is what programmable hardware is for, after all!

In order to try your modifications of the hardware code, we suggest you to provide a very simple Cell to test and not use the Game module. 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 and remove the digits Cell.

3. Implement the game (optional)

   (a) The game starts with one single ternary signal (left, middle or right) displayed on the LEDs. Duration of signal display should be, say, 300ms.

   (b) 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.

   (c) If the entered sequence has been entered correctly, the length of the sequence is increased by one.

   (d) 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.

Documents

- Spartan-3 Starter Kit Board User Guide. File `S3BOARD_RM.pdf` in folder `documents/DigilentSpartan3`
- Active Cells - A Computing Model for Rapid Construction of On-Chip Multi-Core Systems. File `ActiveCells.pdf` in folder `documents/ActiveCells`