In4073 QR: Guide to partial VHDL Approach 2008-2009 v1.0

Widita Budhysutanto
Tiemen Schreuder

Embedded Software Lab
Software Technology Department
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
January 2009

Aim

This document gives some hints and pointers regarding the implementation of part of the IN4073 Quad Rotor assignment in VHDL, by giving details about X32 structure/conventions. Some advantages and disadvantages of taking the (partial-)VHDL approach are listed. How to get started on sensor filtering in discussed slightly more in-depth. The aim of this document is not to teach you VHDL, nor to make you familiar with the internals of the X32 softcore. See the Project Resources for more information on those topics.

Introduction

The experimental X32 softcore (itself written in VHDL) allows less time-critical parts of the embedded system to be written in C. This allows groups to write the entire QR application in C. It is however also allowed to write certain parts of the application in VHDL, which has several advantages, most notably the performance increase. C code running on the X32 runs at only 1-2 MIPS, so anything you don't have to do in C, but instead implement as a VHDL peripheral, buys more time for the controller task of the ES, potentially leading to a more stable flight. There are of course also several disadvantages, which will be mentioned as well. Prior knowledge about the VHDL language is not required, but certainly helpful.

VHDL

VHDL is a type of HDL. HDL stands for Hardware Description Language. It is basically that, a language to describe hardware at a high level. VHDL code can be translated (synthesized and routed) to a digital circuit description (gates, signal lines, etc). This digital circuit can be mapped onto programmable hardware (the FPGA in our case). Important to realize is that VHDL code is not 'executed', it simply results in a specific hardware configuration that conforms to its model (the VHDL code). The synthesizing and routing is a complex process, as the circuit has to be optimized in many ways. For this project the free Xilinx tool chain is used, which comes with the NEXYS boards.

Links to good introductions and tutorials about VHDL can be found on the course resource page. As IN4073 is not a course on learning VHDL, students are expected to learn VHDL autonomously (with their group). One peripheral that may be interesting to study if you are new to VHDL is the Digital Signal to Pulse Converter, found in the student package in the file `/in4073_xufo/x32/vhdl/peripherals/pdc_dpc/dpc.vhd`.

X32

The X32 is basically a byte code interpreter in VHDL. The VHDL code of the X32 is included in the student packages, although it may be a bit complex to learn from. When adding extra functionality, for example sensor filtering, students will have to integrate this as an extra peripheral in the existing X32 code. Currently the X32 code consists of the X32-core (the byte code interpreter) and some extra peripherals, such as a peripheral to control the segment display. Also provided, specific for this project, is the `xufo_comm` peripheral, which handles the communication between the FPGA and QR (according to a specific protocol). The students peripheral(s) can be tested/used by converting all the code (X32 core + extra peripherals + student peripherals) into a bit file using the Xilinx tool chain, that can be uploaded to the FPGA (NEXYS).

Partial VHDL approach

Introduction

There are several parts of the application that can be made in VHDL instead of C. For example (in order of recommended implementation):

- Sensor filtering
- PID controller calculations
- Engine control (based on PID)

What for example isn't recommended to do in VHDL is logging (although it is possible of course). In the sections below some general conventions and guidelines are described. The sensor filtering is taken as example. Implementation of the other components is completely up to the students. Note that the Teaching Assistants most likely do not have intimate knowledge of VHDL (it is not a mandatory part of the course), so they can only help with VHDL up to a certain level. Please take this into consideration.

Conventions – Directory structure

There are several conventions in the X32 project structure and building procedure that are important to know. The X32 project can be found in the student package under `/in4073_xufo/x32/`. Important in this directory are the Makefile and the configuration directories. The X32 can be build in several configurations: eg: just the core, the core with some demo peripherals, etc. For this project the `nx32-combo` configuration (directory) is used. This configuration consists of the X32-core and also for example the `xufo_comm` peripheral. Additionally some specific details about the NEXYS board (eg: number of gates) are part of the configuration. Which VHDL peripherals are included can be found in `nx32-combo/config.vh`. This file contains several ADD_* functions, which are defined elsewhere in other *.vh* files. For example ADD_XUFO_COMM(..) is defined in `/vhdl/peripherals/xufo_comm.vh`. This file was included at the top of `config.vh`. The actual 'implementation' of the `xufo_comm` peripheral is described in `/vhdl/peripherals/xufo_comm/xufo_comm.vhd`. What each argument means in ADD_XUFO_COMM(..) can be deducted from these two files.

In general, the following convention are true:

- .vh Description / port mapping
- .vhd Implementation
- ._vhd Will be preprocessed and renamed to .vhd

Conventions – Sharing with C

To make any partial VHDL approach useful, it is required to 'share' variables with the part written in C. To see how this works it's easiest to study the `xufo_comm` component. The C code should include the x32.h header file. This x32.h file is generated automatically after each x32 build, so if you correctly map a variable to memory in VHDL (learn from examples), a '`#define`' entry is added for you to, so you can reference it from C.

General idea

The `xufo_comm` component retrieves the sensor values from the QR so, among others, its outputs are s0, s1, s2 up to s6, one for each sensor. A filter component would wrap around (or link to) the `xufo_comm` component and take these values as input, and give fs0, fs1, etc as outputs, fs0 being the filtered value of s0. Internally the filter component would keep a buffer of previous values. The values fs0 up to fs6 will have to be mapped to memory, like s0 to s6, so that they can be accessed from C.

Adding a peripheral

When you have created a peripheral, let's say '`xufo_filter`', and want to add it to the configuration, please make sure of the following:

- You made a file `xufo_filter.vh` in `/vhdl/peripherals/`
- You made a file `xufo_filter.vhd` in `/vhdl/peripherals/xufo_filter/`
- You have defined some ADD_* function in `xufo_filter.vh`
- The file `xufo_filter.vh` is included from the relevant `config.vh`
- This ADD_* function is called from `config.vh`
- The Makefile includes `/vhdl/peripherals/xufo_filter/xufo_filter.vhd`
- The Makefile 'default' is set to `nx32-combo` (or a custom configuration by the students)

Building and debugging

Before building please '*`source setup_xilinx`*' in `in4073_xufo/`. This will add the Xilinx tools to the PATH. Synthesizing, routing and generating the bit file is done by simply typing '`make`' in `/in4073_xufo/x32/`. This will start the process of synthesizing and routing the VHDL code. The process will abort if the `xst` script can't be found (`source setup_xilinx`), or if the VHDL code contains errors. The error descriptions aren't always very elaborate. If the errors are reported in peripherals.vhd, try to scroll up for warnings/errors reported in earlier (your) files.

Debugging your code is quite tricky. In general, it's useful to look through the log of the build and review any warnings generated, they may provide clues why something is not working as expected.

Some debugging techniques:

- Use an oscilloscope to verify if the correct signal is put out.
- Use the X32 as debugging platform, by sharing the output of basic operations with C through memory mapping.
- Build a minimal test system (requires a new configuration, to compile without X32) to plug your peripheral into, for example triggered by a button.

- Use the ModelSim tool (not provided) to simulate the hardware design. Will not work well simulating the entire X32 softcore, so use with the minimal test system.

Keep the following keywords in mind when reviewing your code for errors, especially if you are used to software programming:

- non-sequential 'execution'
- signals vs variables
- process sensitivity list

Advantages / disadvantages

Advantages:

- VHDL code results in a hardware circuit: ultra fast
- Less issues in getting the timing of the C part right
- More time for controlling the QR: more stable

Disadvantages:

- Learning a new language, VHDL, takes time
  - Especially since VHDL is not sequential!
- Getting familiar with X32 (directory) structure/conventions takes some time
- VHDL is hard to debug
- Safety mechanisms may be more difficult to verify
- TA's may not be of much help if you get stuck