FPGA Implementation Projects

FPGA implementation projects will use a Terasic DE2i-150 development board, as shown below. The DE2i-150 features an Intel Atom N2600 processor and an Altera Cyclone IV GX FPGA, along with a variety of standard I/O peripherals (USB, Ethernet, VGA, etc.). The DE2i-150 ships with Yocto Linux pre-installed. My recommendation is to stick with Yocto, rather than installing a more mainstream distribution, as the objective of this course project is not to perform operating system modifications, but instead to focus on application development and acceleration using the FPGA.

To use the DE2i-150, all you need to do is plug in a monitor, mouse, and keyboard, and you are ready to go. It is strongly recommended that you use a desktop PC or server to run the FPGA CAD tools; they will overwhelm the lower-end Atom processor on the DE2i-150 board.

A typical FPGA implementation project will involve the following steps:
(1) Obtain a software implementation of an algorithm in a low-level language (e.g., C)
(2) Profile the software to identify the computational bottlenecks
(3) Design and implement a hardware accelerator for each bottleneck segment of the application. Each hardware accelerator will be synthesized onto the FPGA; the processor and FPGA communicate via PCIe.
(4) Repeat the process until no further acceleration is possible or there is no longer room on the FPGA to hold any additional accelerators.
As an example of the type of work you might do for this project, please refer to the following paper, which describes a set of hardware accelerators for the Dynamic Time Warping (DTW) application (a favorite of Eamonn Keogh’s):


Although this particular project did not use a DE2i-150 board, it could, in principle, have used one. The instructor did not have the DE2i-150 boards at the time this project began.

Details:

Language and Implementation: C and C++ are the preferred languages to use for application acceleration. The reason is that they are compiled directly and tend to be very low-level. If you use an application written in Java, for example, you will need to interface the Java Virtual Machine (running on the Atom processor) with the FPGA, which may be potentially quite complex. Another challenging language is MATLAB, because it does not use standard integer/floating-point data formats; thus, it will be very difficult to obtain correct results of a MATLAB application on an FPGA. It will be much easier to re-implement a MATLAB application in C (using standard integer/floating-point data formats) before trying to use the FPGA for acceleration.

Profiling: A variety of software tools have been developed that can profile your application code to determine the computational bottlenecks. For background and an overview of profiling, please refer to the following Wikipedia page:

https://en.wikipedia.org/wiki/Profiling_%28computer_programming%29

You may use any profiling tool that you want. The applications that we will consider for this course are not particularly challenging to profile, so do not put too much time and effort into the choice of profiling tool. Old tools like ATOM or gprof should suffice, although many other profiling tools are available as well.

Hardware Design Languages: You may design the accelerators using any hardware design language that you prefer; VHDL are Verilog are the most popular. If you do not know a hardware design language (yet), please consult an online tutorial; further assistance can be provided by contacting the instructor.

A good VHDL tutorial developed by a UCR graduate student (presently on-leave at Intel) can be found here: http://www.cs.ucr.edu/~jtarango/cs122a.html

FPGA CAD Tools: The DE2i-150 boards should come with a license to use Altera’s CAD software. If you run into any problems with the software, please let the instructor know so that a more effective license server can be set up.
Hardware/Software Optimization and Accelerator Design: Many programmers write software without truly understanding the underlying hardware and its implications. For example, many software developers simply use float or double data types to represent real numbers (in the abstract algebraic sense), without truly understanding that an N-bit number (floating-point or otherwise) can only represent at most $2^N$ distinct values. Alternative options, such as fixed-point, are rarely considered. Similarly, programmers may not understand the implications of overflow and underflow when using integer data types (char, short, long, etc.).

For projects that involve real-valued data, it is expected that you will consider both fixed- and floating-point implementations. For example, it is easy to profile the range of values that a floating-point variable in C may hold over the lifetime of the program; based on the range, you may be able to convert the variable to fixed-point without loss of accuracy, and the range itself may tell you precisely how many bits you need. This issue may vary from application to application.

I highly recommend a publicly available software tool called FloPoCo to generate fixed- and floating-point data paths in VHDL. FloPoCo can be obtained at the following URL:

http://flopoco.gforge.inria.fr/

FloPoCo can generate specific operations (ADD, SUB, MUL, DIV, SQRT, etc.), but it can also generate highly optimized datapaths that span multiple operators, where the optimizations cross the boundaries of individual operators.

In the instructor’s experience, the best way to use FloPoCo is to generate single-cycle operators and then pipeline them by hand or by using FPGA CAD tools.

Hardware/Software Co-design Process: To obtain the best overall performance, the software and hardware must be optimized in a manner that ensures maximal synergy, and the best implementation choices are often specific to a given platform (e.g., this development board, with its PCIe interface, etc.). This means that it may be necessary to rewrite and/or otherwise restructure the software implementation of the algorithm, before, during, and/or after accelerators are introduced to the FPGA. In short, whatever implementation decisions lead to the best overall performance should be made (although eventually, you will need to stop tweaking the implementation, and simply complete the project).
Projects and Applications:

1. MIT Sparse FFT (3-4 students)

In 2012, researchers at MIT introduced a new algorithm to compute the Fast Fourier Transform (FFT) for sparse signals. This result was theoretically important and has profound implications for present and future signal processing applications.

http://groups.csail.mit.edu/netmit/sFFT/index.html

The MIT research group has released source code for several different variants of their algorithm; additionally, a research group at ETH Zurich was able to further optimize the MIT implementation.

Here are links to the MIT and ETH Zurich software download pages (which, fortunately, include documentation):

http://groups.csail.mit.edu/netmit/sFFT/code.html
http://spiral.net/software/sfft.html

The software is written in C++. Representative input data samples are provided.
2. SensorSIFT (2 students)

SensorSIFT is an algorithm/framework that transforms raw sensor data into a representation that tries to minimize exposure of user-defined “private” attributes while maximally exposing application-required “public” attributes.

For example:

Suppose that an application running on a camera-enabled entertainment system (like the Kinect) wishes to determine Alice’s gender to personalize her avatar’s virtual appearance. Suppose also that Alice (the user) has specified that race information should not be available to applications. At present, Alice can either avoid using the application (and thus sacrifice utility) or choose to use the application and forfeit her ability to ensure privacy.

A natural solution to this tension would be to allow data access which is based on pre-defined public and private attributes. While workable for well-known attributes like race and gender, this approach limits innovation as developers are restricted to the pre-defined public attributes. Under the SensorSift scheme, applications can opt to use standard public and private attributes or can propose novel public attributes not known by the platform in advance (private attributes are still defined by the system in advance and exposed to users as options).

Returning to our example, on a SensorSift supporting platform Alice can specify race as a private attribute. The system would then transform the raw camera data samples to adhere to this policy by maximally removing race information while exposing application-desired attributes. These public attributes could be anything defined by the developers — including attributes not known to the platform designers; for simplicity of exposition, however, we’ll use gender as the public attribute.

The transformed sensor data would only be made available to the application if the system successfully verifies (using an ensemble of state-of-the-art classifiers) that the sifted data cannot be used to recognize the private attribute significantly beyond random guessing. If the sift is verified, the target application would receive the transformed data which could then be post-processed to infer the gender value.

Details on the SensorSIFT algorithm, including a published paper, can be found at the following website: [http://homes.cs.washington.edu/~miro/sensorsift/](http://homes.cs.washington.edu/~miro/sensorsift/)

Source code written in MATLAB is provided. To accelerate the application, it may be necessary to rewrite the SensorSIFT software in C or C++, using floating-point (and possibly fixed-point) rather than integer, data types, before attempting acceleration using the FPGA.
3. BCIBench (6-8 students)

BCIBench is a suite of embedded and signal processing applications in the area of brain-computer interface (BCI). A typical BCI application includes 3 steps:

1. Pre-processing
2. Feature Extraction
3. Classification

BCIBench includes 3 preprocessing applications, 7 feature extraction applications, and 4 classification applications. It also includes 4 complete BCI applications for Motor Imagery (2 apps), Visually Evoked Potential (VEP), and P300 PCI.

A link to the BCIBench webpage can be found here: [http://bcibench.org/](http://bcibench.org/)

In this project, students will work together to accelerate as many BCIBench kernels as possible, along with the four complete applications, using the FPGA. This particular benchmark suite is likely implemented in fixed-point (I haven’t checked), and may be sensitive to changes in fixed-point precision, etc.

4. ICBI Superresolution (1 student)

Superresolution is a process by which an algorithm can (try to) improve the resolution quality of a low-resolution image. One particularly effective algorithm for superresolution is Iterative Curve Based Interpolation (ICBI), which is a fairly straightforward image processing technique (e.g., it does not rely on machine learning).

A description of the ICBI algorithm, including source code, can be found at the following URL: [http://www.andreagiachetti.it/icbi/](http://www.andreagiachetti.it/icbi/)

Source code is provided in MATLAB, C++, and CUDA (for GPU execution). It should be fairly straightforward to accelerate the C++ implementation.
5. Lanczos Superresolution in the GIMP (2 students)

The Gnu Image Manipulation Project (GIMP) is a free software project that can perform tasks such as photo editing (http://www.gimp.org/). The GIMP includes several algorithms for superresolution, the most effective of which appears to be based on Lanczos sampling:

https://en.wikipedia.org/wiki/Lanczos_resampling

In the GIMP, low-resolution images are often obtained by increasing their size. To improve the quality of the image, interpolation-based superresolution is applied (http://docs.gimp.org/en/gimp-image-scale.html). The Sync (Lanczos 3) method is among the most effective superresolution techniques that the instructor has observed.

In this project, the students will download the GIMP, and isolate the Sync (Lanczos 3) command. (It may or may not be feasible to run the GIMP on the Atom processor on the DE2i-150 board; if necessary, extract the superresolution code and make it a standalone program). Profile the superresolution algorithm and implement accelerators on the FPGA.

6. Scrypt (2 students)

In cryptography, scrypt is a password-based key derivation function created by Colin Percival, originally for the Tarsnap online backup service. The algorithm was specifically designed to make it costly to perform large-scale custom hardware attacks by requiring large amounts of memory. In 2012, the scrypt algorithm was published by IETF as an Internet Draft, intended to become an informational RFC, which has since expired. A simplified version of scrypt is used as a proof-of-work scheme by a number of cryptocurrencies, such as Litecoin and Dogecoin.

For more details, please visit the Wikipedia page. https://en.wikipedia.org/wiki/Scrypt

The Wikipedia page includes links to several other websites where source code for Scrypt can be downloaded. For this project, a C or C++ implementation of Scrypt should be brought up and running on the DE2i-150 Atom processor, and then accelerated to the best extent possible using the FPGA (note that Scrypt itself is designed to be memory-bound, rather than compute-bound, so the benefits may be limited).

7. Histograms of Sparse Codes for Object Detection (2-4 students)

Histograms of Sparse Codes was introduced at CPVR 2013 as a method for object detection; it was shown to be more effective that Histograms of Oriented Gradients (HOG) classifiers, which were the previously best-known method.

A link to the project page, which includes a paper and source code, is here: https://www.ics.uci.edu/~dramanan/software/sparse/
Additionally, code and training data has been made available for simpler contour detection algorithms based on sparse codes. The paper describing the approach, along with application and training code is here:

http://homes.cs.washington.edu/~xren/research/nips2012/contour_detection_training_v0_2.zip

8. Scene Labeling (2-3 students)

Scene labeling attempts to semantically classify and “label” different regions of a 2-dimensional scene (see below). Low-cost depth cameras, such as Microsoft Kinect, have allowed depth information to be combined with an RGB image to improve the quality of scene labeling.

The following paper, published at CVPR 2012, and source code describe and implement a very effective scene labeling algorithm.

9. Smart Grid Prototype Control (2-3 students)

The instructor has been indirectly involved with an undergraduate senior design project in the EE department, implemented by Roben Van Dusen (UCR IEEE President, 2013-2014) and others. The smart grid prototype is not (yet) truly smart, because it has not yet been placed under software control. In this project, students from CS 223 will work with Roben and his team to control the smart grid prototype using a DE2i-150 board.

(A set of slides in PDF format, provided by Roben, is posted on iLearn).

This project is somewhat open-ended compared to the others, as the most important aspect is proper control, as opposed to application acceleration. Details will be provided following a meeting with Robe on Monday, April 14th.