

Using Gedae with HandelC to Target FPGAs

Peter Taylor
BAE SYSTEMS
Advanced Technology Centre
Chelmsford CM2 8HN
+44 (0) 1245 242242
peter.w.taylor@baesystems.com

Robert Zemurs
BAE SYSTEMS
Advanced Technology Centre
Chelmsford CM2 8HN
+44 (0) 1245 242014
robert.zemurs@baesystems.com

ABSTRACT

In this paper, a novel technique for targeting Field Programmable Gate Arrays (FPGAs) using Gedae and HandelC is presented. The additions to Gedae to support this approach are described together with the data flow model used in the FPGA. Hard real-time input-output and signal processing techniques within the FPGA are described together with lower speed techniques for host computer interactions such as parameter setting and status reporting. Finally, a real-time video image-processing example using the techniques is presented.

1. INTRODUCTION

The ability of Gedae to target a wide range of hardware is based largely on its use of the portable 'C' programming language. HandelC [1] is an approximation to 'C' that can be used to directly synthesize code for use in an FPGA. A combination of these two tools is described that allows Gedae flow graphs to be targeted to COTS FPGA hardware.

2. HandelC Features

HandelC is a commercially available C-like language that can be directly synthesized for execution on an FPGA and has additional features to express fully parallel computations. HandelC is also cycle-accurate, allowing applications to be constructed that execute at a predictable precise rate determined by the FPGA clocks.

The combination of HandelC and an FPGA target provides a number of key advantageous features that must be retained when considering the use of Gedae to target FPGAs

- Cycle accurate computations – the precise timing of computations is predictable and repeatable unlike regular CPU approaches.
- Direct access to Input/Output pins – glue-less connection to external components such as RAM devices, serial and parallel interfaces.
- Parallelism – the FPGA can carry out a large number of computations in parallel - typically 10^3 concurrent arithmetic operations.
- Pipelining - the FPGA can carry out more complex operations in a pipeline; beginning the processing of a new data token before completion of the previous operation. This allows an apparently sequential sequence of processing stages to be executed in parallel thereby increasing the throughput.

3. Technique Developed

The technique developed involves coding primitives in HandelC and using the Gedae tool to interconnect the primitives to form a complete FPGA application. As is usual with Gedae, a hierarchical approach is taken, with low-level FPGA primitives implementing basic operations and higher-level operations implemented as collections of interconnected primitives.

The Gedae functional block diagram facility is used to connect together a series of primitives and hierarchical blocks into a complete FPGA application in the normal way.

Two key aspects of Gedae are not desirable for FPGA targets:

- Static scheduling – whilst static analysis is helpful (to detect deadlocks for example), the Gedae approach ends up with a sequential schedule and assumes primitives are required to execute strictly sequentially rather than concurrently.
- Execution kernel – again the execution of the schedule involves the Gedae kernel calling the primitives in a pre-defined sequential order.

In order that the primitives can be executed concurrently on an FPGA, they are each coded in HandelC as a separate parallel thread that repeatedly reads input data, performs a computation and outputs the result on each FPGA clock cycle. This is rather like using separate threads or processes on a regular CPU except:

- The threads execute truly in parallel – there is no context switching.
- Each thread can compute on every clock cycle without slowing down other threads.

A custom set of HandelC primitives has been developed for use in the FPGA. These include integer operations such as add, subtract and multiply, video processing primitives such as edge detection and threshold, and specialized I/O primitives to access hardware including external RAM blocks.

4. Scheduling in the FPGA

When targeting a regular serial processor, Gedae has to do a great deal of work to determine the best order of execution of the primitives and the granularity of each execution, but with the FPGA target this scheduling analysis is not needed – all the primitives can be designed to execute concurrently and with a granularity of one. Each primitive is coded in HandelC as an infinite ‘while’ loop that:

- Reads input data from each input stream
- Carries out a computation
- Outputs the results to each output stream

In fact, HandelC can execute each of the above steps in a pipeline and we end up with a pipelined single clock cycle primitive.

In effect, each primitive can poll its inputs on each FPGA clock cycle, something that would be very wasteful on a regular CPU but is entirely acceptable on a FPGA.

The technique employed is a one-way handshaking scheme – a single handshake bit that is emitted on the output ports of each primitive and checked at the connected input ports on each clock cycle. That is, a primitive can regulate the rate it outputs new data tokens (up to a maximum of once per FPGA clock cycle), but must always be able to accept input data tokens at the maximum rate. This extra bit on each data stream indicates that the rest of the bits are valid data – the combination of this ‘data valid’ bit and the actual data must be output every clock cycle and also read by downstream primitives each clock cycle.

5. Additions to Gedae

In order to achieve the FPGA capability, an extra facility has been added to the Gedae tool accessible through a top-level menu item. (This is called ‘Extras’ - see Figure 1). The code for this extra functionality resides entirely in the ‘UserGedae.c’ file (used to hold user customizations) and does not require changes to the Gedae tool as such. When one of the menu items is selected, the code automatically traverses the Gedae application flowgraph hierarchy and collates the HandelC code from each FPGA primitive encountered into a single source code output file. The HandelC tool is then run to compile this file into a standard EDIF netlist. The FPGA toolset is used to place-and-route the netlist ready for loading into the FPGA. Finally, a Tcl/Tk [2] GUI is launched allowing the FPGA application to be loaded and run.

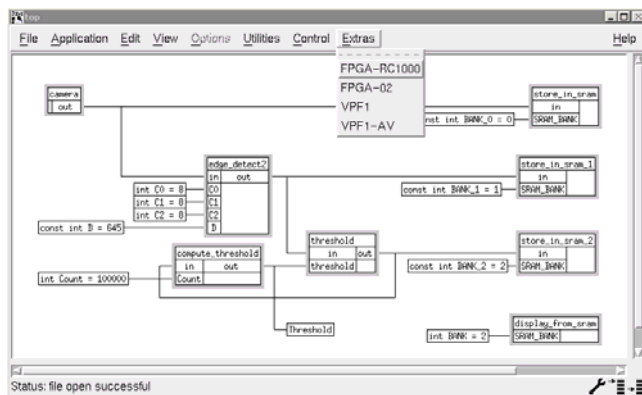


Figure 1 Extra Gedae Menu for FPGA Code Generation

The generated HandelC source code consists of the following sections formed primarily by concatenating the methods in each primitive encountered:

- Each of the include methods
- A variable to hold each parameter value – this needs a unique valid ‘C’ name so the ‘path’ to the parameter is used. That is, each flowgraph name from the top level down to the primitive is concatenated using two underscores “__” as a separator.
- Similarly, a variable to hold the value of each output of each primitive instance.
- A function for each primitive instance. The function body consists of:
 - The function declaration
 - Any local variables
 - The reset method
 - The apply method in an infinite ‘while’ loop.

A separate instance of the function is produced for each instance of a primitive. This is required in HandelC so that each instance is implemented as unshared logic in the FPGA and can therefore execute concurrently and have its own local variables.

- A ‘main’ function that just calls each of the primitive functions in parallel.

5.1 Tcl/Tk FPGA Control Panel

As well as the normal techniques for changing application parameters on the Gedae flowgraph (e.g. using the pop-up dialogs), a simple Tcl/Tk control panel interface is also generated. This allows a single mouse click to download and start the FPGA application and then presents the user with a simple control panel allowing each of the application parameters to be changed by operating a slider – see Figure 2.

Each of the parameters in the flow graph (except those marked ‘const’ i.e. assumed to be constant) is mapped to a slider. The slider range is from 0 to twice the value set on the flowgraph, but the application begins execution with the parameter value specified on the flowgraph.

This approach means that once Gedae has been used to produce the FPGA application, it can be run without having the Gedae tool present. The user can also change parameters in a convenient way with immediate effect. However, changes to ‘const’ parameters are made using the Gedae tool and typically mean that the HandelC code produced is changed and therefore needs to be recompiled.

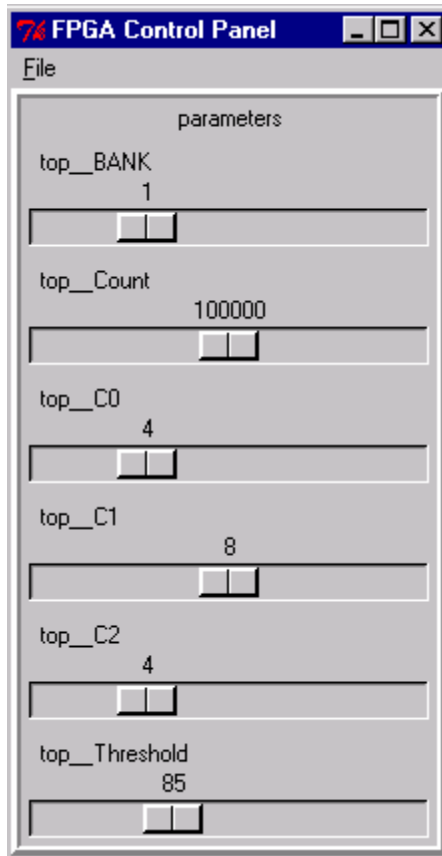


Figure 2 Auto Generated Tcl/Tk FPGA Control Panel

6. Use of FIFOs to Connect Primitives

The use of FIFO (first-in first-out) buffers to connect primitives has been considered. There are basically two application areas where this is useful:

- When the FPGA needs to accept data from or send data to a regular CPU. Normally the CPU would use DMA (direct memory access) to transfer bulk data in and out of the FIFO. This approach allows the CPU part of the Gedae flowgraph to operate at a high granularity to achieve the best throughput performance (but at the expense of increased latency).
- For internal connection within the FPGA small FIFOs can be used (typically 16 deep) in order to connect primitives that operate at different FPGA clock speeds. Work has been carried out in this area using Xilinx/COREGEN EDIF based asynchronous FIFO components. They can be readily called from Handel-C programs and have fully separate read and write clocks that are connected to the clocks signals of the two FPGA clock domains being interfaced. However, the FIFO component only works reliably when the two clock frequencies are derived from a common clock i.e. they are synchronously related.

Some investigations have been carried out in the use of external RAM FIFOs i.e. FIFOs located in external RAM blocks which are directly connected to the FPGA pins (e.g. Transtech DSP FPGA-02 development board). These types of FIFOs allow FPGA

primitive functions to more easily access and process large data structures (such as complete images, which would not fit into internal FPGA RAM) or allow primitive functions to run at high granularities where (granularity x token size) is large. The latter case would be applicable where a CPU is transferring data to an FPGA and good throughput performance is required using block DMA.

A parallel, pipelined image processing application has been implemented using primitive functions connected via external RAM FIFOs. Each primitive operates on a complete image and processed images are transferred along the pipeline via external FIFO data buffers, which internally operate a dual buffer scheme. This allows all the primitives to run in parallel, with primitive N reading from its FIFO buffer A and the previous primitive N-1 writing to FIFO buffer B at the same time. This can only be achieved if the external RAM blocks are in fact dual-port RAMs with separate busses (as is the case with the FPGA-02 board). The dual buffers are swapped over after all the primitives have completed the processing of their input image and have written the processed image into their output FIFO buffer.

However, there are a number of disadvantages of using external RAM FIFO schemes:

- Requires the FPGA target board to have several external, independently accessible dual-port RAM blocks.
- Only a limited number (typically 4-6) of primitive functions can be run in parallel corresponding to the number of independent RAM blocks available.
- Synthesis and Place and Route (PAR) can be difficult, time consuming and require a significant amount of resources of even a large FPGA (Virtex II). This is mainly due the number and size of internal FPGA busses needed to connect the primitives to the external memory.

7. Board Support Package

A Gedae Board Support Package (BSP) for a COTS FPGA board has been developed which allows the generated code to be loaded into the FPGA, parameter values to be transferred into the FPGA and data streams to be passed between the host computer and the FPGA.

The BSP consist of the following components:

- A build script, that compiles the generated HandelC code for the target, places and routes the FPGA layout and produces a 'bit-file' ready for loading into the FPGA.
- A run script that resets the FPGA, downloads the 'bit-file' into the FPGA and sets the FPGA operational.
- A set of registers inside the FPGA, these can be read or written (typically over a PCI bus) by the PC host system and can also be read and written by the FPGA application at any time. The registers are used as parameter registers with the value written by the PC host and then read, as required, by the FPGA. The registers are also used in the other direction, the FPGA produces an output 'parameter' (really a status value), that can be read at any time by the host.

- The registers are also used to provide a set of moderate speed (about 1 Mbyte/sec) streaming ports allowing application data streams to be fed from a host system into the FPGA and the results transferred back to the host for analysis
- A Tcl/Tk control panel (common to all FPGA BSPs) that starts the FPGA application running and allows parameter setting. Each time one of the sliders is operated, the desired register value is transferred into the FPGA. In addition, if the FPGA modifies the parameter register, this is reflected on the control panel; the slider automatically moves to show the new value. This is used to show dynamic status values e.g. the adapted threshold value in Figure 1 and Figure 2.

8. Example Image Processing Application

The technique has been used to develop a real-time video image processing demonstration in which an FPGA takes video data stream directly from a digital camera, performs image processing such as 2-D convolution filtering and directly outputs a video signal suitable for display on a VGA monitor.

Figure 1 shows the application flowgraph that can be compiled to run on a range of COTS FPGA hardware, for example the Transtech DSP FPGA-02 or their VPF1 [3]. The key flowgraph components are:

- A camera interface. The camera is a low cost web-cam type with a digital stream output that can be fed directly into the FPGA with no addition 'glue' logic. The video data rate is 13.5M pixels per second with a VGA resolution of 640 x 480 pixels. The camera primitive interfaces to the hardware and outputs a stream of pixels. Each pixel is an 8-bit value indicating the luminosity.
- A dual-port RAM interface that can concurrently store and retrieve entire 640 x 480 pixels images in memory.
- A video output primitive that generates the required synchronization and Red, Green and Blue waveforms for direct connection to a PC VGA monitor.
- The image processing primitives, in this case, an edge detector and an adaptive thresholder. The edge detector uses a 3x3 convolution kernel with the supplied parameter values.
- The threshold level is determined automatically to achieve the required picture density. This is done by counting the number of pixels exceeding the threshold in the output image, and increasing or decreasing the threshold appropriately.

8.1 Results of Running a Simple Image Processing Application

Figure 3 shows the unprocessed image as obtained from the video camera.

Figure 4 shows the result of applying a 3x3 convolution filter set for edge detection. Stronger edges appear brighter and regions with no edges appear dark

Figure 5 shows the result of a threshold operation selecting only the strongest edges.



Figure 3 Unprocessed Video Image



Figure 4 Video Image After Edge Detection



Figure 5 Video Image After Threshold Operation

9. Conclusions

- The approach taken allows the Gedae tool to be used to quickly change the application flowgraph during development whilst retaining the predictable real-time capability inherent in the FPGA device.
- Further work is required to develop a useful set of FPGA primitives both for the image processing domain and other domains such as communications where predictable hard real-time performance is a key requirement.
- Many COTS FPGA boards provide high performance data streaming between the FPGA and embedded data processors; this should be exploited to take advantage of the benefits of heterogeneous signal processing.

10. Acknowledgements

GEDAE is a trademark of Blue Horizons Development Software

HandelC is a trademark of Celoxica Limited.

This work was directed and funded by BAE SYSTEMS Avionics as part of their Electro-Optic Demonstrator Project.

11. REFERENCES

- [1] HandelC is a COTS tool available from Celoxica, <http://www.celoxica.com/>
- [2] Tcl/Tk is a portable scripting language with built-in Graphical User Interface facilities. <http://www.tcl.tk/>
- [3] Transtech DSP a vendor of COTS FPGA hardware <http://www.transtech-dsp.com/>