A Run-Time Reconfigurable Engine for Image Interpolation

Rhett D. Hudson, David I. Lehn and Peter M. Athanas
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, Virginia 24061-0111
rhudson@vt.edu

Abstract

Custom Computing Machines (CCM's) have demonstrated significant performance advantages over general-purpose processors for certain classes of problems. However, problems can always be found which require computational resources in excess of those available on a particular CCM. Exploiting the reconfigurable nature of FPGAs can alleviate this limitation. The FPGAs' computational resources can be time multiplexed to allow different portions of the computation to execute in stages. Intermediate results are saved to memory and passed on to later stages of the computation. This technique is used in this work to implement an image interpolation engine on the Xilinx XC6264 Reference Board. The engine utilizes 2-5-2 splines to take advantage of their computationally convenient powers-of-two arithmetic.

1 Introduction

Most early work in the FPGA-based computing field has dealt with platforms that provide the ability to rapidly prototype application specific hardware. This form of computation follows the traditional design techniques adopted by the ASIC community. In fact, most commercial FPGA design tools are ASIC design tools that were simply retargeted from VLSI cell libraries to support similar libraries of primitives for FPGAs. These rapid prototyping platforms configure their FPGAs once. That configuration is used throughout the execution of an application. The obvious limitation of this approach is that one can always conceive of a problem that requires more resources than are available on a given computing array.

While silicon-based ASIC solutions must adopt this kind of static design, applications based on the inherently flexible FPGA do not. Computing solutions that utilize the dynamic nature of FPGAs are called run-time reconfigurable (RTR) solutions. RTR applications solve the scalability problem of traditional rapid prototyping techniques by adopting a divide and conquer approach. Large problems are broken down and partitioned temporally into stages, each of which fits onto the array. The first stage receives input data, performs computations and stores the results into a memory. The array is then reconfigured for the next stage, which computes results based on the outputs of previous stages. This process is then repeated until all the required stages have executed and the final results are available. Using these techniques, the size of a problem that an RTR application can solve is limited only by the size of the memory required to store intermediate results and possible latency requirements.

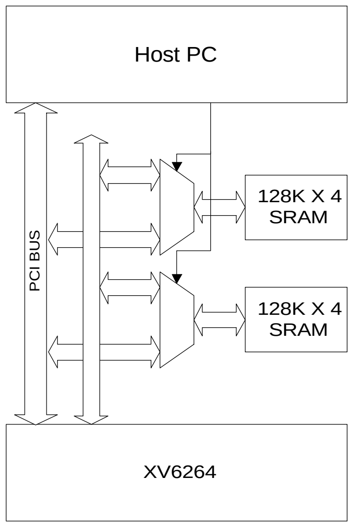

Figure 1: Reference Board.

The efficiency of these systems is dependent on the amount of time the platform spends doing useful computations. Time spent reconfiguring the FPGAs and reordering the intermediate results is lost as overhead. Consequently, each reconfiguration has a cost in terms of efficiency and performance. Ideally, the time spent reconfiguring should be negligible with respect to the time spent computing.

This work examines the implementation of a spline-based image interpolation engine. Image interpolation has a variety of uses, including image compression and high fidelity home theater systems. The interpolation engine was implemented on the Xilinx XC6264 Reference Board. Figure 1 presents a block diagram of the Reference Board depicting the interconnection of the host, on-board memory and the XC6264.

The next section explains the theory of the 2-5-2 splines used by the image interpolator. Following that, Section 3 explains the general approach of temporal partitioning for run-time reconfiguration. The fourth section covers some of the implementation details and the fifth section looks at the performance of the engine. Section 6 compares the interpolator engine with commercial implementations. The final section presents some conclusions.

2 Background

The theoretical basis for the image interpolator process was taken from [FerP97]. The procedure increases the resolution of an image by interpolating new pixels between the existing ones. Typically, this kind of interpolation is done with cubic splines, but the computational requirements of that method are prohibitive for real-time applications. Furthermore, floating point arithmetic is typically needed to maintain acceptable resolution. Bilinear interpolation is a computationally tractable solution, but the resulting image quality is unacceptable for many applications. The 2-5-2 spline approach presented in [FerP97] provides results of acceptable quality with roughly the same computational complexity as the bilinear approach.

The 2-5-2 spline matches the lower computational complexity of the bilinear approach by choosing its spline basis so that the coefficients in the computation of the spline are powers of two. Since multiplication and division by powers of two can be implemented with low-resource extremely fast shift units, computation of 2-5-2 splines is dramatically faster and less resource intensive than other methods. The very nature of the arithmetic makes it well suited for FPGA implementations.

The interpolation procedure is broken down into two parts: the inverse filter (IF) and the Fast Spline Transform (FST). Both computations must be performed in two dimensions, i.e. on both the rows and the columns of the image. Calculations in both directions require pixels from only one

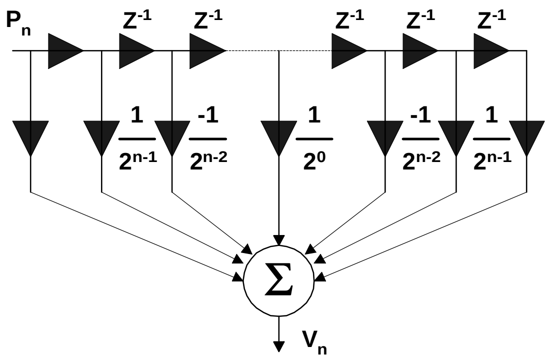

Figure 2: The Inverse Filter [FerP97].

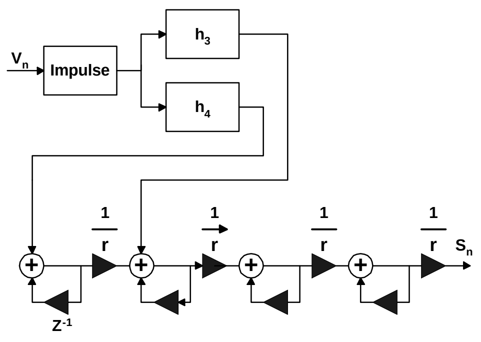

Figure 3: The Fast Spline Transform [FerP97].

row or column at a time. The computation can be performed on the rows first and then the results from that computation can be used in the column calculation. The inverse filter takes the original image and calculates the spline vertices according to the following:

$$V_n = \sum_{k=n-l}^{n+l} \left(-\frac{1}{2}\right)^{|k-n|} P_k$$

where n varies from one to the number of pixels in a row or column and P_k's for $k<1$ *and* $k>m$ are zero. This process is diagrammed in Figure 2.

The output from the vertex calculation is an image that is approximately the same size as the input image. The vertex calculation has some edge effects that make the image slightly larger than the original. This image is expanded to a higher resolution by introducing some number of new pixels between the spline vertices. The FST phase takes the spline vertices and uses 2-5-2 spline arithmetic to interpolate values for the new pixels. This procedure is diagrammed in Figure 3. The result is an image with higher resolution than the original.

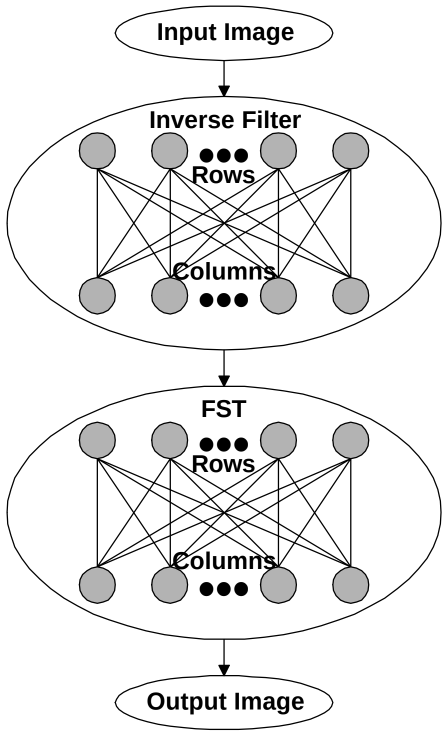

Figure 4: Computational Structure of Image Interpolation.

3 Approach

The implementation of the image interpolation process is an excellent example of using RTR methods to map a large calculation onto the relatively small Xilinx Reference Board. The mapping process began with an analysis of the interpolator's computational structure.

The authors of [FerP97] demonstrated their interpolation engine using MatLab. Based on their MatLab code, a C++ based engine was created that implemented the interpolator using integer arithmetic. The C++ engine was used to verify the procedure using integer math and to determine the bit precision required to achieve acceptable quality in the output image.

3.1 Computational Structure of Image Interpolation

Figure 4 illustrates the computational structure of the image interpolation process. Each gray circle represents either a row or column in the working image. The rows and columns of the inverse filter are mutually dependent on each other. In other words, all the columns must be computed before any of the rows can be computed or vice versa. Whether the rows or the columns are computed first is not significant. The rows and columns of the Fast Spline Transform are also mutually dependent. In addition, the diagram indicates the FST is completely dependent on the results of the IF. All the rows and columns of the IF must be computed before any of the rows or columns of the FST can be computed.

The diagram also exposes inherent parallelism in the computation. For both the IF and the FST, each row is independent of all the other rows and each column is independent of all the other columns. This allows simultaneous calculation of any number of rows or any number of columns.

3.2 Temporal Partitioning

Once the computational structure is understood, temporal partitioning can take place. Since the XC6264 does not have the resources required to implement the entire computation, the required hardware must be partitioned into different stages that can be loaded and executed sequentially on the platform.

The analysis of the computational structure indicates that the entire IF calculation must be performed before any of the FST calculation can be performed. The division between IF and the FST is an obvious candidate for a partition boundary. Within both the IF and the FST, no column can be computed until all of the rows have been computed. This is also a natural choice for a partition boundary. These decisions leave us with four partitions: IF rows, IF columns, FST rows and FST columns. Further partitioning might be necessary if the FPGA resources required by the computations in each of these partitions exceeds those available on the platform. Alternatively, if the resources required represent a fraction of the available resources, then the required hardware may be duplicated within a partition to take advantage of the parallelism discovered in the computational structure. The exact choice of partitions and parallelism cannot be made until some implementation decisions have been reached.

4 Implementation

The image interpolation engine was designed using the VHDL Elaborator [GraD98] and the Xilinx XACTstep Series 6000 Tools [Xili96]. These tools were chosen based on availability and familiarity. Other tool suites for end-to-end development of XC6200 designs are available. The Oberon-based Trianus system is one example [GehL96].

The interpolation engine was coded in structural VHDL with hardware specific attributes to guide the place-and-route tools during the creation of the physical design. The VHDL Elaborator produced an EDIF netlist of Xilinx Unified Library components from the VHDL code. The XACTstep Series 6000 tools were then used to place-and-route the EDIF netlist for implementation on the XC6264.

4.1 Software

The only Xilinx tools available during the design effort were still in the developmental stage. The VHDL Elaborator worked well converting structural VHDL into EDIF netlists. Recent versions have included an extensive selection of parameterized components that would have been very helpful had they been available in the early stages of the design.

The XACTstep Series 6000 place-and-route tools were, generally, incapable of producing layouts with reasonable compactness and low critical paths. Most components had to be placed by hand. This was done using a combination of the XACTstep layout editor and location attributes attached to components in the VHDL. VHDL attributes were used frequently for datapath construction, but were not practical for irregular modules, such as the control unit.

4.2 Engine Architecture

Most of the arithmetic in the interpolation engine is done serially. The driving force in the decision to use serial arithmetic was the XC6200 routing architecture. Serial arithmetic favors local complexity over global control. In the chosen serial implementation, only two nets need to be routed between arithmetic units. One line is the serial data line and the other is a control line that indicates when the most significant bit of a computation is present. This scheme dramatically lowers the number of nets that must be routed a significant distance across the FPGA. Resource requirements for serial arithmetic are also lower than for parallel arithmetic allowing larger designs to fit onto the platform.

The interpolation engine was designed to translate a 128x128 eight-bit image into a 512x512 eight-bit image. The original image is loaded into one of the Reference Board's two banks of SRAM. Each RTR stage of the engine reads the input image from one bank and writes its results into the other bank. The next stage then reads its input image from the bank that was the output bank of the previous stage. Pixels are stored as words in the external SRAM. The interpolator's memory-control unit performs parallel-to-serial conversion while reading operands from memory and serial-to-parallel conversion while writing operands to memory.

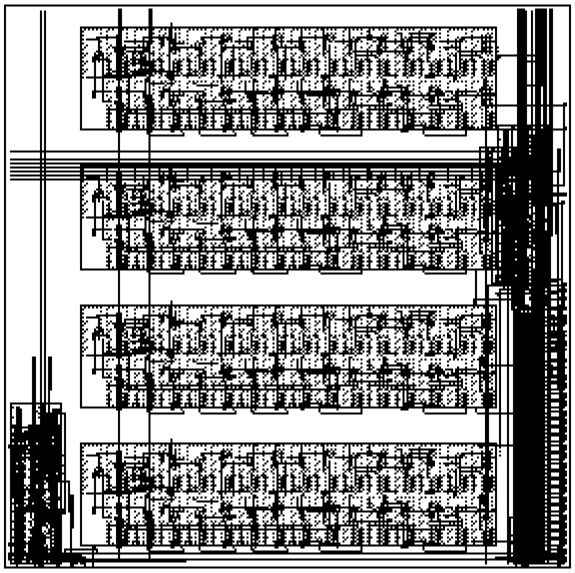
Figure 5: IF Floorplan.

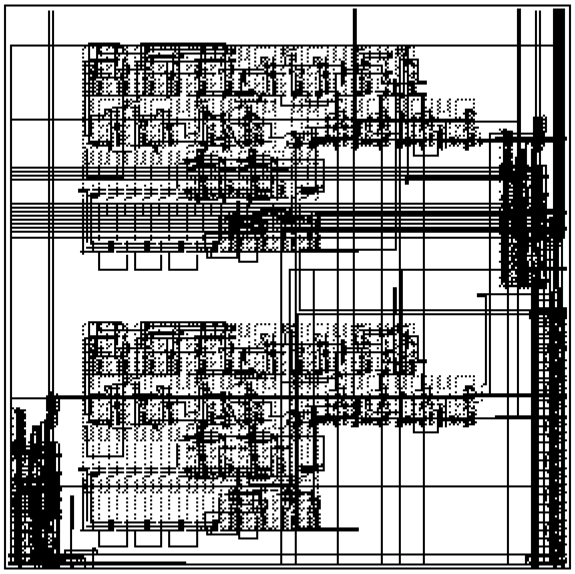
Figure 6: FST Floorplan

Internally, the interpolation process uses a 32-bit fixed-point representation. Since the arithmetic is done using serial techniques, the size of the word does not have an adverse effect on the size of the computational units. Width of the word does, however, increase the latency of the computation.

The IF computation can be implemented as a FIR filter. The implementation of a single tap of the IF requires approximately 72 of the XC6200's functional units (FUs). The IF implementation requires 31 32-bit filter taps, which is approximately 2,232 FU's. There are 16,384 FUs available on a XC6264, which leaves room for exploitation of the row and column parallelism in the computation. Allowing for the overhead of an address generation unit, parallel-to-serial conversion and the tool-imposed need to hand place the taps, four IF units can be implemented on an XC6264. The floorplan for the layout of the IF unit is shown in Figure 5.

The FST computation can be implemented using two of the 20-tap FIR filters and four cumulative sum units. Seven of the taps in the FST filters require multiplies by numbers that are not powers of two. These taps are implemented using a shift and add operation. They require approximately 144 FUs each. There are twelve taps that are multiplies by powers of two; they require 72 FUs each. Twenty-one of the filter taps are multiplies by zero, which only require 48 FUs. There are four cumulative sum units that require 48 FUs and five division units that require 72 FUs. The total requirement for an FST unit is 3,720 FUs. Again, with overhead for address generation, parallel-to-serial conversion and the limitations of the tools for automatic placement, two FST units can be implemented on the XC6264. The floorplan for the layout of the FST unit is shown in Figure 6.

4.3 Partial Reconfiguration

One of the methods used by RTR application designers to reduce configuration overhead is partial reconfiguration. Partial reconfiguration requires hardware support on the FPGA. The XC6200 series devices provide this capability.

When two subsequent stages of an RTR CCM application have similar structures, some of the hardware already loaded onto the CCM may be reused in the next stage. Each FU from a previous stage that does not need to be reconfigured for the next stage represents less configuration information that has to be sent from the host to the FPGA. This reduces the time that is required to reconfigure the device.

The column and row computations for both the IF and the FST are very similar. In fact, they are identical except for minor changes in the address generation circuitry required to read the pixels from memory in a different order. Since these configurations are so similar there is a savings in performing a partial reconfiguration.

Shirazi's *CalDiff* tool [LukS97] was used to create the partial configurations. The *CalDiff* tool can analyze two full configuration files and compute the fastest partial reconfiguration between the two.

5 Results

Limitations of the tools and the Xilinx Reference Board have caused some serious problems in the performance of the actual implementation. The pre-release version of the automated place-and-route tool proved inadequate for circuits of significant complexity. This necessitated hand placement of almost all the components in the final design. The layout of the reference board places the data and address pins to the external memories on opposite sides of the die. Just routing control signals from one side of the chip to the other can easily account for 75% of the critical path. While hand placement of the computational units can result in pipelined serial units with a critical path on the order of 30 ns, routing the control signals for the memory control unit across the chip causes the critical path to grow to around 120 ns. It would be possible to resolve some of these issues by meticulous hand placement of the control and address generation circuitry coupled with a scheme for pipelining the routing delay across the chip. In general, however, it would have been useful to have some other facility for performing address generation. A XC4000 series part coupled close to the memory would have provided the speed required to implement the memory interface. Consequently, the results presented here are based on the timing requirements of the computational units rather than the configuration as a whole.

5.1 Computational Performance

According to the place-and-route tool, the critical path through the computational circuitry has a delay of 33 ns, allowing the engine to be clocked at a maximum rate 30 MHz. The interpolation algorithm requires 32-bit precision to achieve results of acceptable quality. Internally, the engine utilizes bit-serial arithmetic operators.

The IF unit is entirely pipelined with a latency of approximately 500 clocks. When compared with the approximately half-million clocks required to compute an output image, the pipeline latency can be safely neglected from performance calculations. The IF unit can, on average, compute one output pixel every 32 clocks.

The circuitry that resets the IF pipeline between the rows and columns of the image does so by inserting zero-valued pixels into the pipeline. These extra pixels make the effective size of the input image 144x128. With a 33 ns clock and a 32 clock latency between outputs, each IF unit can produce an output pixel once every 1.1 μs. There are four such units operating in parallel. That gives an effective time of 267 ns between each output pixel. Given that the image contains 18,432 pixels, the IF row configuration requires 4.91 ms to compute an entire output image. The

IF column configuration is computationally identical and requires an additional 4.91 ms.

The FST unit contains two 20-tap FIR filter units that run in parallel and are serially connected to four cumulative multiply and sum units. Like the IF, the FST is a fully pipelined serial arithmetic unit. The latency through the FST is shorter than that through the IF, and the size of the input image is larger than that for the IF. Once again, the latency through the FST unit is negligible. The FST can be accurately modeled as producing one output pixel for every 32 clocks.

The critical path through the FST has a 33-ns delay. The effective image size, including extra pixels for resetting the pipeline, is 522x128 for the rows and 522x512 for the columns. There are only two FST units operating in parallel, so the time between output pixels for the FST configuration is 533 ns. The times to process a 66,816-pixel image and a 267,264-pixel image are therefore 35.64 ms and 142.54 ms, respectively.

The sum of each of the times for the various

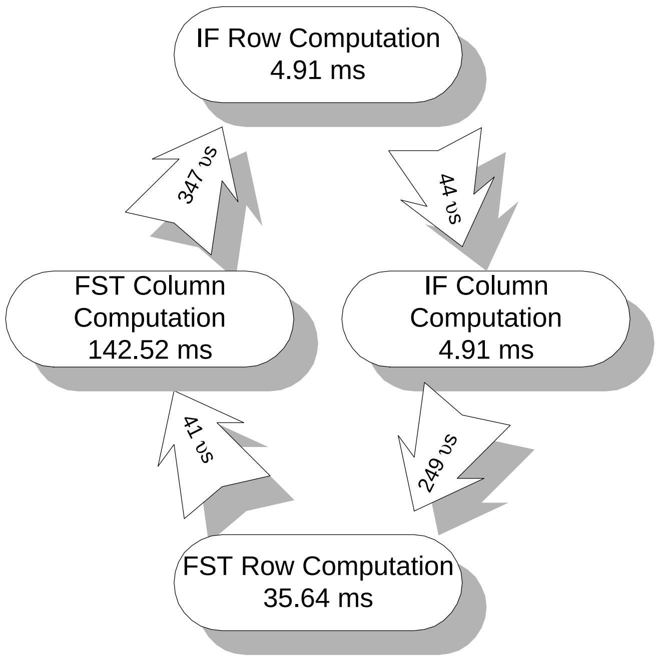

Figure 7: Reconfiguration Cycle.

stages, momentarily setting aside the overhead for reconfiguration, results in a 188-ms latency for each frame. This translates to approximately 5 frames per second. The algorithm scales perfectly up to eight XC6264s operating in parallel. The performance would increase linearly from 5 frames per second using one XC6264 to 40 frames per second for eight chips.

5.2 Reconfiguration Overhead

To be practical, RTR systems must insure that the time spent performing reconfiguration is negligible with respect to the time spent performing calculations. If the system is reconfigured too frequently, more time is spent reconfiguring than calculating useful results. The time wasted performing reconfiguration is called reconfiguration overhead.

The four configurations used in the image interpolator were analyzed using *CalDiff*. The first stage of the image interpolation process requires 11,465 address-data pairs to be written to the XC6264. These writes are performed across the 33 MHz PCI bus, which results in a total configuration time of 347 µs. The configuration from the row stage of the IF computation to the column stage of the IF computation requires less time, because only the address generation and control circuitry needs to be updated. The computational core of the configuration does not need to change. This reconfiguration requires only 1,483 address-data pairs or 44 µs.

The reconfiguration from the IF unit to the FST unit is large since the two overlays have no circuitry in common. It requires 8,228 address-data pairs or 249 µs. The reconfiguration from the first stage FST calculation to the second stage FST calculation is much smaller because the two configurations share the core computation circuitry. This configuration only requires 1,367 address-data pairs, or 41 µs.

The four configuration's total reconfiguration overhead sums to 682 µs. That represents a reconfiguration overhead of 3.6%, which can be neglected when compared to the overall computation time. Therefore, in this particular application, the overall cost of switching between overlays is overshadowed by the core computation. The cycle of reconfiguration is diagramed in Figure 7.

6 Comparison with Other Methods

The RTR techniques presented here may compare favorably to commercial image interpolators. The Runco SC-4200 [Runc98] uses custom ASICs to perform image interpolation for home theater systems. The SC-4200 is a rack-mounted unit that draws 65W during normal operation. The SC-4200's image interpolation algorithm is also significantly more complex than the one presented here. It retails for around $25,000. Its low production volume may make the use of off-the-shelf FPGA technology superior to its use of custom ASICs.

7 Conclusions

The image interpolation procedure is an excellent application for demonstration of RTR principles. The computational structure of the calculation has distinct temporal boundaries and exhibits parallelism within partitions. The large amount of input data required for each stage requires enough compute time that the reconfiguration overhead can be safely neglected. Even though the overhead is negligible, the engine also demonstrates the benefits of partial reconfiguration through dramatic reduction of configuration time for similar computational stages.

8 References

[BurD97] J. Burns, A. Donlin, J. Hogg, S. Singh and M. de Wit, "A Dynamic Reconfiguration Run-Time System," in the *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, CA, pp. 66-76, April 1997.

[FawW96] B. Fawcett and J. Watson, "FPGA Applications in Digital Video Systems," *Proceedings SPIE Workshop on High-Speed Computing, Digital Signal Processing*, and Filtering Using Reconfigurable Logic, Boston MA, pp. 283-294, November 1996.

[FerP97] Ferrari, Leonard A. and Jae H. Park, "An Efficient Spline Basis for Multi-Dimensional Applications: Image-Interpolation", *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 757-760, 1997.

[GehL96] Stephan Gehring and Stefan Ludwig, "The Trianus System and Its Application to Custom Computing", *6th International Workshop on Field-Programmable Logic and Applications*, Darmstadt, Germany, September 1996.

[GraD98] Douglas M. Grant, *Velab Release Notes*, Xilinx Inc., http://www.xilinx.com /apps/velabrel.htm, 1998.

[HadH95] J. Hadley and B. Hutchings, "Design Methodologies for Partially Reconfigured Systems," *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, Napa, CA, pp. 78-84, April 1995.

[LukS97] Wayne Luk and Nabeel Shirazi, "Compilation Tools for Run-Time Reconfigurable Designs", *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa California, April 16-18, 1997, pp. 56-65.

[Runc98] Runco Inc., *Super IDTV II SC-4200 Specifications*, http://www.runco.com/ sc4200specs.html, 1998.

[Xili95] Xilinx Incorporated, *XC6200 Field Programmable Gate Arrays*, 1997.

[Xili96] Xilinx Incorporated, *XACTstep Series 6000 User Guide*, 1996.