A Development System for Creating Real-time Machine Vision Hardware Using Field Programmable Gate Arrays

Thomas H. Drayer, Joseph G. Tront, Richard W. Conners

Bradley Department of Electrical Engineering, Virginia Polytechnic Institute and State University, 360 New Engr. Bldg., Blacksburg, VA 24061 - USA; phone: (540) 231-5067; fax: (540) 231-7498; e-mail: tdrayer@birch.ee.vt.edu, jgtront@vt.edu, conners@birch.ee.vt.edu

Philip A. Araman
United States Forest Service Brooks Forest Products Center, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060-0503 - USA; phone (540) 231-5341

Abstract

In this paper, we introduce a new development system for creating real-time image processing hardware using custom computing machines with multiple Field Programmable Gate Array (FPGA) chips. Three distinct processes are accomplished within the development system: design entry, verification, and translation. A library of modules that implement common low-level machine vision functions is used to create complex designs based on a dataflow graph representation. The library's low-level image processing modules contain both gate-level and chip-level hardware components, of which the gate-level components are compiled into the functionality of available FPGA chips. Standard interfaces are established for input/output of the modules, allowing for the creation of sophisticated software support tools. Experimental results verify the utility of this development system for easily creating real-time machine vision hardware using multiple FPGA-based custom computing machines.

1. Introduction

FPGA-based custom computing machines have been shown to provide the performance required for real-time image processing [1, 2]. However, the gate-level programming of these hardware architectures is difficult and time consuming to accomplish. Currently available FPGA development tools require the expertise of an experienced digital hardware design engineer to create complicated image processing designs. The development system presented in this paper simplifies the process of creating these image processing designs.

FPGA chips provide large arrays of gate-level programmable logic resources that can be reprogrammed an unlimited number of times. Within each chip is an array of Configurable Logic Blocks (CLBs) used to implement sequential or combinational logic. The functionality of each FPGA is established by writing to Static Random Access Memory (SRAM) bits within the chip that control multiplexers for signal routing or look-up tables to define programmable function generators.

A multiple FPGA-based custom computing machine is defined by the availability of several FPGAs, interconnections between the FPGAs, and possibly other chips in the architecture. The "other chips" are referred to as chip-level resources, while the reconfigurable logic elements of the FPGAs are referred to as gate-level resources.

The image processing problems that have been solved using FPGA-based custom computing machines represent application-specific solutions which cannot be used for other real-time image processing tasks or on other FPGA-based architectures. This paper presents a
new development system for creating image processing designs using arbitrary multiple FPGA-based architectures. The development system is created by combining commercially available software and custom interface software. After design entry and verification, image processing designs are then partitioned, placed, and compiled into the resources of an FPGA-based custom computing machine. The custom computing machine used to realize the design is called the destination architecture.

An integrated development system provides several benefits. Although it is difficult to design the logic of the low-level image processing modules, existing modules are easily combined to create complex designs. Second, since the development system can translate the image processing design into arbitrary FPGA architectures, designs that outgrow initial hardware platforms are easily translated into new hardware architectures without major redesign. Finally, the use of standards and conventions in the creation of modules allows the development of sophisticated software tools for verification of image processing designs.

2. Hardware Architectures

Three classes of destination hardware architectures are defined below; completely specified, incompletely specified, and undefined architectures.

The completely specified architectures of the first class represent the traditional approach to hardware design. In this case, the destination architecture is an existing printed circuit board design. All chips are permanently connected to the board and all interconnections between the chips are defined by traces on the printed circuit board. Both the number and type of all chips on the printed circuit board are completely specified. Splash II \[3\] and Champ \[4\] are examples of completely specified architectures.

Incompletely specified architectures allow flexibility in the number and/or type of chips in the architecture, as well as possibly fixing only some of the interconnections. Sockets may be used to allow one of a number of different chips to be embedded in the architecture. When additional functionality is not required, the sockets may remain empty. The MORRPH-ISA \[5\] board is an example of an incompletely specified architecture.

When a new printed circuit board is to be constructed specifically for the current image processing design, the destination architecture is undefined. The designer has complete control over the number and types of chips, as well as the interconnections between chips.

The design entry, verification, and translation processes of this development system support any of the three classes of destination architectures. The verification tools are unaffected by the class, but the steps taken in the design entry and translation processes are influenced by the class of destination architectures.

3. Development System

3.1. Design Entry

Complex image processing tasks can typically be subdivided into several simple low-level image processing functions. Common low-level functions include filters, thresholding, Look-Up Tables (LUT), and morphological operators. A dataflow graph is commonly used to represent the processes and flow of information between low-level image processing tasks in a complicated image processing design. The dataflow graph representation of a simple image processing design is shown in Figure 1. This type of dataflow graph is used as the design entry method for our development system. Several modules are combined from an existing library of parts to create a complex image processing design.

The low-level image processing designs are controlled by some type of host computing device. Each low-level image processing module is a special-purpose processing unit that is designed and optimized at the gate level for a specific processing task. Communication with the host requires additional circuitry that is dependent on the destination architecture. This includes the transfer of image and result data, global variables, and system interrupts. Standard interfaces are defined to allow the development of sophisticated software translation and verification tools. This programming model is illustrated in Figure 2.

A standard interface is established for the transfer of image and result data. This interface allows image
processing modules to be combined in an arbitrary manner. A new bus standard, called the Synchronous Unidirectional Image Transfer or SUIT bus [6] has been specifically developed for this purpose. The standard consists of a 16-bit bus that multiplexes up to sixteen channels of one, two, or three dimensional data. Individual channels may represent image or result data. Time-multiplexing of data on the individual channels allows 8-, 16-, 32-, or 64-bit data to be transmitted.

Similarly, a 16-bit I/O port interface bus standard is defined to provide access to static and dynamic global variables used by the image processing modules. The same gate-level circuitry implements read/write port locations whenever global variables are required.

These port locations can be automatically instantiated and mapped into the I/O space of the destination architecture. All port locations and interrupts communicate with the host computer through an architecture-specific host interface.

Not all low-level modules may be used with every destination architecture. Specifically, the chip-level resources included in some modules may not be available in (or cannot be added to) some incompletely specified or completely specified architectures. However, often these modules are easily redesigned to accommodate the existing chip-level resources of a specific destination architecture. The requirements for the translation of image processing designs into the three classes of destination architectures are defined in the next section.

### 3.2. Translation

After the symbolic representation of an image processing design has been created, it is translated into the physical resources of the destination architecture.

The image processing design consists of chip-level components, gate-level components, and interconnections. Chip-level components are realized using available non-FPGA chips. Gate-level components are compiled into the resources of the available FPGA chips. Interconnections may be implemented by using traces on a printed circuit board, by the interconnection resources of FPGA chips, or a combination of both. The process of translation assigns each component of the image processing design to specific resources of the destination architecture.

The distinction between chip-level and gate-level logic components is made during the design of each individual module, a process called chip partitioning. Chip-level components must be assigned to one of possibly many corresponding IC chips in the destination architecture, a process called chip placement. Gate-level resources are compiled exclusively into the resources of FPGA chips. Three processes are associated with the translation of gate-level components. First, the number and type of FPGA chips must be determined for incompletely specified and undefined architectures, a process called FPGA determination. The second process, called logic partitioning, distributes the gate-level components of the image processing design among the available FPGA chips. Finally, the partitioned logic must be compiled into the resources of each individual FPGA chip. Commercial software tools exist to compile an input design into the logic resources of a single FPGA chip, a process called FPGA compilation. Currently multiple chip designs are not supported by commercial software because there are no satisfactory software tools for efficiently partitioning logic between multiple FPGA chips. Some research has been performed by Kuznar et.al. [7] for the optimal selection of FPGAs from a library of parts. However, our development system must provide the tools for logic partitioning.

Nets of the image processing design that are connected to chip-level components or are inputs and outputs of the design must use board level interconnection resources in the destination architecture. Nets that connect gate-level components in separate FPGAs after partitioning also require board-level interconnection resources. All these nets are called global nets. Nets that only connect gate-level components within the same FPGA are called local nets. Nets that are both connected to multiple gate-level components and exterior logic are subdivided into a single local and a single global net.

Local nets are assigned to the interconnection resources of the individual FPGAs during the FPGA...
compilation process. Global nets are assigned to board level resources by a process called global routing. This process is only required for completely and incompletely specified architectures, which have at least some of their interconnections fixed by the layout of the printed circuit board. When the destination architecture is undefined, the appropriate traces are simply established on the printed circuit board.

The processes defined above are performed in the following order: chip partitioning, FPGA determination, chip placement, logic partitioning, FPGA compilation, and finally global routing. These processes translate the functionality of the image processing design into the destination architecture.

The two processes for which commercial software is not available are logic partitioning and global routing. Two methods are provided by the development system for these processes: a manual and an automated method.

The number of interconnections and gate-level components in a typical image processing design is too large to manually partition the logic of the original circuit. However, the complexity of the logic partitioning and global routing is drastically reduced by adding a single constraint to the partitioning. This constraint requires all the logic of each low-level image processing module in the design to reside in a single FPGA. After adding this constraint to the translation process, manual logic partitioning and global routing are complicated but feasible tasks.

With automated logic partitioning and global routing, the above constraint can be removed. Since the feasibility of global routing is affected by each partition, it is logical to combine partitioning and global routing into a single program. However, both partitioning and global routing represent NP-hard problems. This severely limits the possibility of solving the two problems simultaneously. The approach taken by this development system is to implement both algorithms separately, but to require that the cost function of a partition include a factor related to the difficulty of global routing.

All image processing designs that have been created with the development system to this point were manually partitioned and globally routed. New logic partitioning and global routing algorithms have been created and tested. These programs are currently being integrated into the development system with all current and future image processing designs.

3.3 Verification

Verification utilities are provided for all processes of the design methodology. Both simulation and emulation are used to verify a single low-level module or a complex image processing design. Additionally, a timing analysis can be used to determine the performance of the design. Each of the verification processes provides a different perspective for the verification of a module or image processing design.

The design methodology uses the logic simulator Viewsim, provided by Viewlogic, Inc., for functional simulation. Viewsim uses a unit-time propagation delay for each gate and flip-flop, providing a functional simulation. Custom software has been written to translate image data files into the waveforms required for simulation. Output waveforms are stored to ASCII data files during simulation. Custom software is used to translate and display the results of the simulation. This verifies the functionality of the design, without determining if it will operate at the desired clock speeds.

Timing information is dependent upon how the gate-level components are compiled into the logic and routing resources of the FPGAs. The Xdelay program from Xilinx Inc. is used to analyze the propagation delays in each chip after FPGA compilation. Although this provides only a minimal amount of information, it provides a good estimate of the performance level of the design.

In-circuit emulation is also a useful tool for verifying compiled image processing designs at real-time clock speeds. Some of the logic of the destination architecture or a second FPGA-based custom computing machine may generate known test data. In-circuit emulation verifies the output results of the image processing design from known image data at real-time speeds.

4. Results

Several complex image processing designs have been created and verified using this development system. An example image processing design is illustrated in Figure 3 and details of the performance of specific image processing modules is presented in [6]. The design in Figure 3 creates both a 256-value grayscale histogram and a histogram of the color image after it has been mapped into a palette of 2,000 colors.
This design is created and verified using the tools of the development system, then translated into the incompletely specified architecture of the MORRPH-ISA board. Two Pulnix TL2600 RGB linescan cameras provide the data for two identical MORRPH-ISA boards. Each MORRPH-ISA board contains six Xilinx 4010PG191-5 FPGA chips. The two boards have a data processing rate of 5 Mbytes/sec. The design is used in a commercial system developed for the color sorting of machined parts.

5. Conclusions

This development system has proven to be a valuable tool for the creation of complex image processing designs. Complicated image processing designs are easily created and ported to different classes of FPGA-based custom computing machines. Sophisticated verification tools establish confidence in the correctness of created designs. This development system provides a method for overcoming the limitations of designing real-time image processing hardware using FPGA-based custom computing machines.

6. References
