Media Processing with Field-Programmable Gate Arrays on a Microprocessor's Local Bus

V. Michael Bove, Jr., Mark Lee, Yuan-Min Liu, Christopher McEniry, Thomas Nwodoh, John Watlington

Massachusetts Institute of Technology Media Laboratory
Room E15-324, 20 Ames Street, Cambridge, Massachusetts 02139 USA
vmb@media.mit.edu, http://www.media.mit.edu/~vmb/

ABSTRACT

The Chidi system is a PCI-bus media processor card which performs its processing tasks on a large field-programmable gate array (Altera 10K100) in conjunction with a general purpose CPU (PowerPC 604e). Special address-generation and buffering logic (also implemented on FPGAs) allows the reconfigurable processor to share a local bus with the CPU, turning burst accesses to memory into continuous streams and converting between the memory's 64-bit words and the media data types.

In this paper we present the design requirements for the Chidi system, describe the hardware architecture, and discuss the software model for its use in media processing.

Keywords: video compression, field-programmable gate array, data-flow computing, digital signal processing

1. INTRODUCTION

Now that field-programmable gate arrays (FPGAs) have reached gate densities at which they can perform useful computational tasks, particularly in certain mathematical and signal-processing domains, they are increasingly being applied as processing accelerators in personal computers and workstations. A typical FPGA add-on card for a PC might contain – in addition to the FPGA itself – a bus interface, buffer memory, and provisions for programming the chip.

Certain tasks are particularly suited to reconfigurable hardware implementation, and many multimedia algorithms exhibit the appropriate characteristics:

- *Data parallelism* with minimal dependencies among units or regular blocks of data
- *Continuous high bandwidth* into and/or out of the system
- *Variable-length or unusual datatypes* which are not well supported by typical CPUs
- *Long processing pipelines* without branching
- *Data rearrangement* which can stress a CPU's cache and bus interface

In this project, the Object-Based Media and Spatial Imaging Groups of the MIT Media Laboratory examine a way of placing the FPGA directly on the local bus of a typical high-end processor (a PowerPC 604e) while providing hardware support for stream-based computing. Rather than modifying the motherboard of an existing machine, we have constructed a processing unit based on the PowerPC Common Hardware Reference Platform (CHRP); in order to make the system as useful as possible and to allow parallel operation of these processors we have built the processor (called Chidi) in the form of a PCI-bus card which can be installed in the backplane of another computer.

2. STREAM-BASED COMPUTING

The Object-Based Media Group at the Media Laboratory has for several years proposed stream-based software techniques and hardware archtectures for managing interconnected and possibly heterogeneous parallel processing resources, particularly in the domain of sophisticated video and audio processing.[1] Even in systems with standard hardware architecture the application of stream-based software techniques is advantageous.

The fundamental idea behind stream-based systems is not to consider each function to be performed as fetching and storing data points in multidimensional arrays, but rather to consider functions operating on one or more one-dimensional streams of data that are the result of traversing the arrays by means of an access pattern. An access pattern is a parameterized description of a regular mapping from a multidimensional array to a one-dimensional sequence. The pattern must define the dimensions of the array, and the order in which they are traversed; for each dimension it is necessary to define a starting offset, a step parameter, and a range over which the stream extends. In the case of irregularly-accessed arrays, the stream mechanism can still identify data parallelism and provide memory latency advantages by defining a "bounding volume" which a processing element requires at a given time.

A software application is then described as a set of functions connected together in a graph structure which also declares the access patterns entering and leaving each function. The graph structure makes explicit the dependencies among the functions, enabling a run-time resource management process to exploit process parallelism by executing independent tasks on multiple processors if available; the access patterns aid in memory management and – by making explicit the locality of operations – enable the run-time process to exploit data parallelism by executing the same task on multiple processors if available. In such a system, the programmer does not have to perform explicit mapping of tasks to processors, nor does the mapping become fixed at compilation time. Rather, the same application will run on systems with differing hardware resources, and will adapt dynamically to changes in processor load, or interconnect characteristics.

While stream-based software methods can themselves be of great benefit, they are of even more interest (and efficiency) when accompanied by hardware features such as intelligent DMA controllers that understand the access patterns, and buffering in data paths between processors and storage that hides access latency variations from the processors. The first of these features eliminates the computational load involved in address calculation, which in multidimensional DSP tasks can be significant; the second prevents a processor from having to wait because of nonavailability of required data, or a blocked memory write.

The MagicEight (formerly Floe) system, developed by John Watlington at the MIT Media Laboratory, is a computing environment which supports medium- to coarse-grain parallelism, using a dataflow model of execution.[2] Emphasizing media processing applications, MagicEight works with multidimensional streams of relatively small (8-1024 bits) scalar data elements. It is intended to execute on single general-purpose processors, or networked arrangements of up to roughly 500 heterogeneous processors, using either message passing or shared memory. MagicEight is in many ways a more generalized descendant of the Norman resource management daemon used on our earlier Cheops system, which supported specialized processors combined with a single general-purpose CPU and simple stream-based DMA controllers.[3]

MagicEight uses a hybrid dataflow model of execution, in which sequences of instructions (roughly, functions) with stream inputs and outputs are written in a standard programming language such as C; these functions are then embedded in a dataflow-style structure and scheduled in a dataflow manner. At runtime, an algorithm is mapped onto the most appropriate processing elements available in the executing system, which can reside in the same physical machine or elsewhere across a network. Multiple architecture shared libraries support both standard CPUs as well as configuration data for partially- (*e.g.* specialized DSP devices) and fully-reconfigurable (*e.g.* FPGAs) hardware processing elements. We expect that a programmer/designer who implements (for example) a compiled VHDL library function for an FPGA will also provide a software version of the function for execution on a CPU; thus the same program will execute on systems without the FPGA.

MagicEight is designed for an environment made up of one or more processing nodes, each of which contains at least one general-purpose processor (which executes that node's MagicEight process) and optionally other processing hardware as well. A node's MagicEight process manages its processing resources and memory, and communicates with other MagicEight processes on other nodes. A user invokes an application at one node (which will act as the I/O console for the application); cooperating MagicEight processes then spread tasks among the permitted nodes and coordinate their communication. A debugger called Icepick traces the execution of tasks across a set of processing nodes.

In the case of the Chidi card, the CPU executes a MagicEight process which schedules tasks for execution on both the CPU and the FPGA; this MagicEight process communicates with a similar process running on the host machine's CPU and possibly with MagicEight processes on other Chidi cards or other machines (as Chidi doesn't have a console, a program will be launched from the host machine's MagicEight daemon). Not all stream-based tasks must execute on the FPGA; indeed part of the design requirement is that both the FPGA and the CPU be able to operate simultaneously.

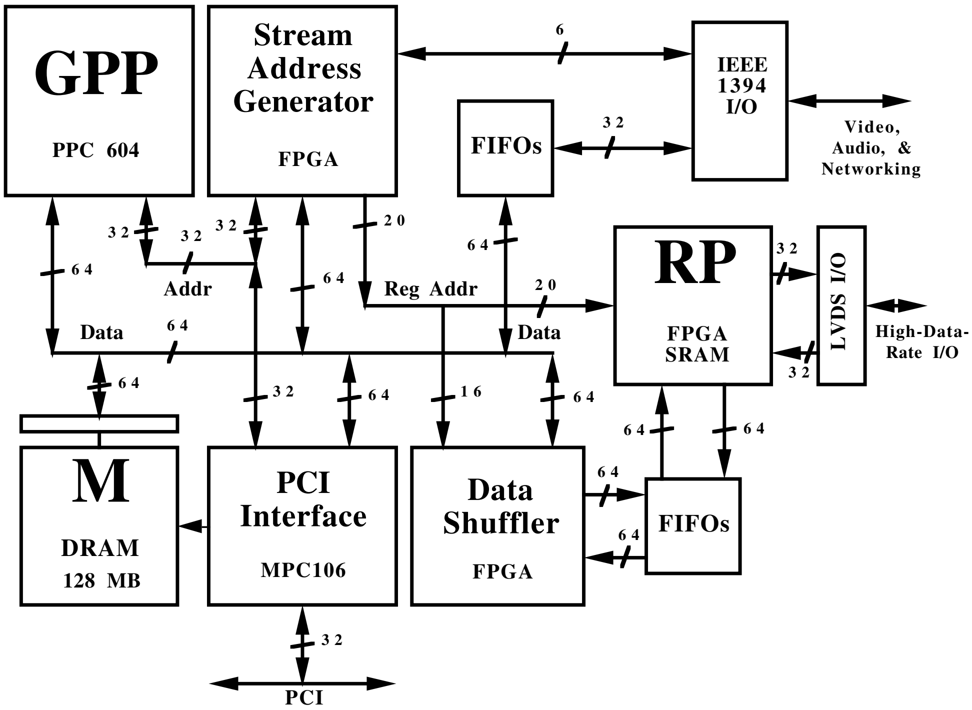

Figure 1: Chidi is a PCI-bus card whose local bus contains both a standard microprocessor (marked GPP) and a 100,000-gate field-programmable gate array (marked RP). Only main datapaths – not control signals – have been indicated.

3. SYSTEM OVERVIEW

Chidi (Figure 1) is a PCI-bus card for personal computers or workstations. It is intended to perform signal-processing computations on streams of data, and contains such stream-friendly features as hardware support for address generation from access patterns, and FIFOs that isolate a high-speed processing element from variable memory-access latency and bus bandwidth. At a system level, Chidi is similar in design to a standard multiprocessor PowerPC motherboard; however in addition to a PowerPC 604e running at 250MHz or faster, Chidi provides a stream-based reconfigurable processor (an Altera 10K100) whose functionality can be changed under software control. The reconfigurable processor (RP), provides the equivalent of 100,000 logic gates by means of 4992 logic elements each comprising a programmable four-input lookup table, a register, and logic for interface and carry/cascade chains. Additionally, the chip supplies twelve "embedded array blocks" each containing 2048 bits of register-interfaced memory. We supplement this with 2 megabytes of external high-speed SRAM, which can be used for large lookup tables, or temporary storage.

The Motorola MPC106 PCI bridge chip both bridges a 66.66MHz local bus to the PCI backplane and serves as a memory controller, permitting up to four PowerPC 604e chips to work together. The bus interface on this CPU supports both single-word and four-beat burst transactions. In the latter case, one address returns four sequential 64-bit data words from memory, reducing the overhead needed for accessing streams of data and effectively increasing the bandwidth of the data bus such that large transactions can occur without one bus master device completely locking out others. On Chidi this means that the reconfigurable processor can read and write data from main memory while the CPU also accomplishes useful tasks. The MPC106 supports up to a gigabyte of DRAM; a Chidi card might typically contain 128 megabytes.

The main design problem in Chidi is making the bus interface of the reconfigurable processor appear to be another 604e, emulating the bursts of 64-bit accesses by which a 604e would read and write main memory while providing continuous streams of data to the FPGA. While this might be accomplished as part of the RP's configuration, as this logic will be common to all RP functions we placed it outside, implementing it as a pair of Altera 10K50 FPGAs and several FIFOs. The functionality was divided into two portions: the Stream Address Generator (SAG) and the Data Shuffler (DS) which will be discussed separately below.

4. THE STREAM ADDRESS GENERATOR

As mentioned above, a fundamental requirement for stream-based media processing is a means of address generation for fetches and stores that can manage multidimensional data arrays. This function is performed by a subsystem known as the Stream Address Generator (SAG).

Implemented as a programmable state machine rather than a set of hardwired loadable counters, the SAG can manage a number of simultaneous streams subject to the limit that the maximum number of streams (both input and output) is eight or less, and that the total number of dimensions being scanned in those streams is 32. I/O pinout limitations on both the RP and the Data Shuffler don't permit all of these "virtual channels" to have their own connections, so they must share a smaller number of physical channels, as will be further explained in the next section.

The SAG also manages several other aspects of the operation of the RP. It implements the register interface used for writing parameters to the RP, the DS, the IEEE 1394 interface, and the on-board alphanumeric status/debug display. It also interfaces to the 604e the interrupts generated by the RP to indicate that it has completed a processing task.

Although the CPU initiates the RP configuration process, it is the SAG that contains all of the logic necessary actually to program the RP. The microprocessor provides the starting address of the configuration file in memory and instructs the SAG to begin RP configuration. Then, the SAG handles the remainder of the configuration details and interrupts the microprocessor when the process is complete. The theoretical minimum configuration time, based upon Altera's published specifications, is 119.3ms. In our system we use a slightly slower configuration clock and have measured a time of 143.2ms. This not insignificant overhead must be taken into account by MagicEight when scheduling tasks.

Because the FPGA used for the RP may not be able to operate some hardware designs at the full 66.6MHMz clock speed, a register in the SAG allows the RP's clock frequency to be set to one of ten values between 66.66MHz and 14.32MHz. The FIFOs between the DS and the RP permit this clock boundary to exist (essentially, an RP running at a clock rate significantly below that of the local bus simply makes its burst accesses to memory farther apart in time). This provision also permits an unoptimized RP configuration to be used until an optimized one is developed.

5. THE DATA SHUFFLER

The Data Shuffler (DS) – implemented by an Altera 10K50 – services the data portion of any transaction going to or coming from the RP. These transactions include main memory accesses for stream processing as well as register interface accesses. For register interface transactions, the data is passed directly to the RP via mailbox registers, while for stream processing, the DS can perform some type of manipulation on the data before loading it into the external FIFOs so that it may be processed by the RP. A block diagram for the Data Shuffler is given in Figure 2.

The DS will need to "shuffle" the stream data going from main memory to the RP for two main reasons. First, the data stream might be offset from a 64-bit word boundary coming out of main memory. The DS will align this

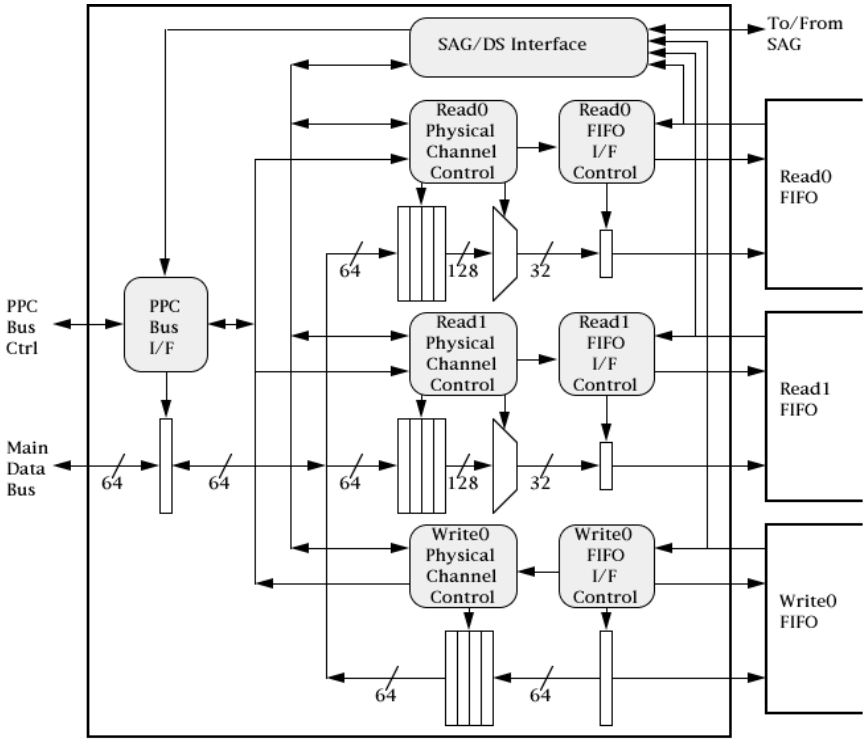

Figure 2: The Data Shuffler converts between streams and 64-bit memory accesses.

data before allowing the RP to begin processing. Second, the data might need to be subsampled in some particular manner based on the application that is running in the RP. An example might be a decimate-by-3 operation to extract one channel from 24-bit packed RGB image data. The Data Shuffler supports byte, short (16-bit), packed RGB (24-bit), word (32-bit), and double word (64-bit) data types.

For stream processing, the Data Shuffler can operate in one of two modes. The first mode implements two independent 32-bit wide input physical channels. Named with respect to the RP, these channels are called Read0 and Read1. Each input physical channel is made up of a section that resides internally to the DS and a section implemented using external FIFOs. The DS can also be configured for 64-bit mode. In this mode, the two physical channels are used as one 64-bit input channel. For either mode, there is a 64-bit output channel, Write0, which holds the output streams generated by the RP.

It is important to note that the programmer doesn't see these physical channels, but rather the *virtual channels* that are multiplexed over them by the SAG. Depending on its I/O needs, the design for a particular RP function may have to provide input demultiplexing and output multiplexing logic, and separate internal FIFOs for each virtual channel (see Figure 3). A "generic" input interface supports eight virtual input channels and four virtual output channels, thus allowing the RP to contain one or more functions that collectively process up to eight streams and generate up to four.

Interfaces between the RP and the SAG, and the DS and the SAG provide a data request mechanism for streams. When only the physical channels are needed, the DS generates the requests, notifying the SAG when it can process another four-beat burst input transaction, or when there is output data available to be written back into main memory. When virtual channels are used, the flow control logic in the RP (marked "SAG/RP interface" in Figure 3) notifies the SAG when some virtual channel can accept more data, but the transaction can proceed only when the DS also indicates that the corresponding physical channel can accept another four-beat burst transaction.

Because of hardware constraints, the current implementation can support at most only eight different "shuffling" patterns at one time. However, this pattern map can be loaded with different patterns by the microprocessor via the register interface. Theoretically, this allows the DS to support an unlimited number of patterns for input data streams, but practically the number of patterns obviously cannot increase indefinitely.

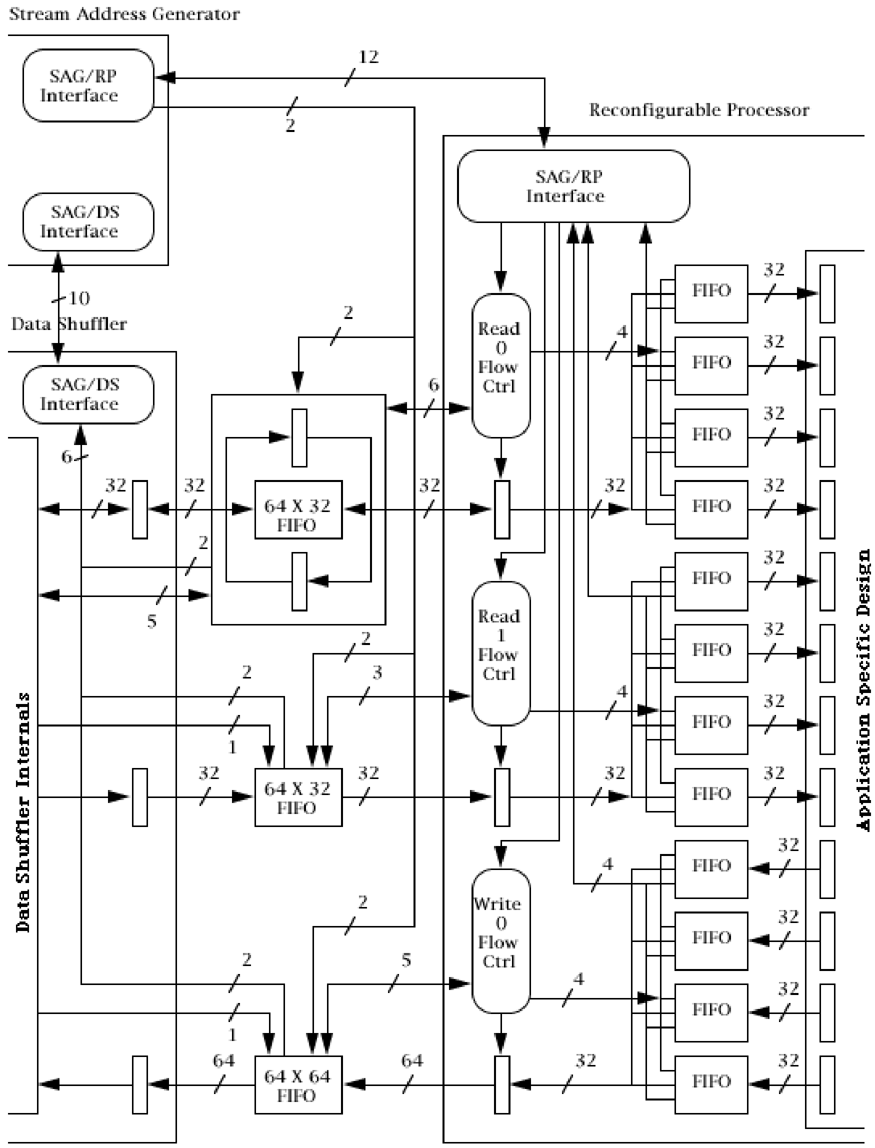

Figure 3: For tasks involving more than two input and one output streams, the Reconfigurable Processor must multiplex/demultiplex the physical channels into virtual channels. The internal FIFOs shown are not needed for one- or two-input, one-output functions.

6. INTERFACES

Besides the PCI bus of the host computer, Chidi can source/acquire data and communicate by means of several other ports. An IEEE 1394 interface can transfer data at 200 Mbits/second to other 1394-compliant devices such as digital cameras, camcorders, and other Chidi cards. FIFO memory provides a clock boundary between this subsection's 32MHz clock and the 66.66MHz clock of the data bus. To software, the 1394 interface appears as one or more streams. A bidirectional low-voltage differential signaling (LVDS) interface allows external data sources to connect directly to the RP at approximately 2 Gbits/second. Among other applications, we intend that multiple Chidi cards will use this port to drive future holographic video displays.[4]

7. CONCLUSIONS AND STATUS

As of the date of this writing, a set of functions used in video segmentation and holographic video is being developed for the RP, and a Chidi version of the MagicEight daemon is under development. The initial production run of Chidi boards in mid-1998 exhibited two annoying but not fatal problems. Because of their fixed address translation, MPC106 PCI interface chips are not really self-compatible, meaning that two Chidi boards cannot coexist in the same backplane, nor can they be used in computers in which the motherboard uses such a chip. A revision of the board to use a different interface chip is underway. Also, the 10K50 devices used for the Stream Address Generator and Data Shuffler were not able to perform at the 66.66MHz local bus speed when implementing all the desired operational modes. A reduced-functionality design of both subsystems is able to run at full speed on the current board; a faster version of the 10K50 chip is now available and has shown in simulation that it will be able to implement the full functionality at full speed so it will be substituted in later board versions.

8. ACKNOWLEDGMENTS

This research has been sponsored by the Digital Life Consortium at the MIT Media Laboratory.

REFERENCES

1. J. A. Watlington and V. M. Bove, Jr., "Stream-Based Computing and Future Television," *SMPTE Journal, 106,* April 1997, pp. 217-224.

2. J. A. Watlington and V. M. Bove, Jr., "A System for Parallel Media Processing," *Parallel Computing, 23:12* December 1997, pp. 1793-1809.

3. V. M. Bove, Jr. and J. A. Watlington, "Cheops: A Reconfigurable Data-Flow System for Video Processing," *IEEE Transactions on Circuits and Systems for Video Technology, 5,* April 1995, pp. 140-149.

4. J. A. Watlington, M. Lucente, C. J. Sparrell, V. M. Bove, Jr., and I. Tamitani, "A Hardware Architecture for Rapid Generation of Electro-Holographic Fringe Patterns," *Proc. SPIE Practical Holography IX, 2406*, 1995, pp. 172-183.