OneChip:
An FPGA Processor W ith Reconfigurable Logic

Ralph D. Wittig* and Paul Chow

Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road
Toronto, ON M5S 3G4

{wittig,pc}@eecg.toronto.edu

Abstract

This paper describes a processor architecture called ***OneChip*** *, which combines a fixed-logic processor core with reconfigurable logic resources. Using the programmable components of this architecture, the performance of speed-critical applications can be improved by customizing OneChip's execution units, or flexibility can be added to the glue logic interfaces of embedded controller applications. OneChip eliminates the shortcomings of other custom compute machines by tightly integrating its reconfigurable resources into a MIPS-like processor. Speedups of close to 50 over strict software implementations on a MIPS R4400 are achievable for computing the DCT.*

1. Introduction

Most computationally complex applications spend 90% of their execution time in only 10% of their code [1]. The core functions executed in this 10% of the code of a given program naturally differ from application to application making it difficult to provide special-purpose instructions to accelerate these core functions and still maintain the notion of a general-purpose CPU. The custom compute machine (CCM), whose execution units can be customized on a per application basis, appears to be the solution to the contradiction of general purpose computing and high performance processing.

Many present day applications utilize a processor and other logic on two or more separate chips. However, with the anticipated ability to build chips with over ten million transistors, it will become possible to implement a processor within a sea of programmable logic, all on one chip. Such a design approach would allow a great degree of programmability freedom, both in hardware and in software: CAD tools could decide which parts of a source code program are actually to be executed in software and which other parts are to be implemented with hardware. The hardware may be needed for application interfacing reasons or may simply represent a coprocessor used to improve execution time.

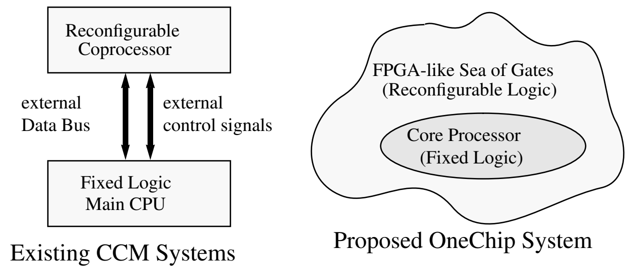

Fig. 1: Reconfigurable Logic Integration Scheme

Independent of the size of the reconfigurable compute element(s), all of the early CCMs are loosely coupled systems with a clearly identifiable processor-coprocessor frontier that always crosses the boundary of one or more chips. However, loose coupling is one of the main limitations for obtaining high speed ups from reconfigurable compute engines. From previous work [2] it is known that achieving reasonable performance from CCMs hinges on two crucial issues that can be identified as:

- the inflexible coprocessor access protocol
- the processor-coprocessor bandwidth limitation

Programmable logic need not only be used for application speed-up, it can also be employed as intelligent glue logic for custom interfacing purposes such as in embedded controller applications. Current single-chip embedded processors attempt to provide very flexible interfaces that can be used in a large number of applications. However, they can often result in interfaces that are less efficient than intended. Furthermore, it might be desirable to perform some bit-level data computations in-between the main processor and the actual I/O interface. *OneChip* 's architecture provides the flexibility required for a general purpose field-configurable interface for embedded processor applications.

In this paper, we present a scheme that couples a 32-bit fixed-logic core RISC processor with the reconfigurable logic more closely than in any of the

*Ralph D. W ittig may now be contacted at: Xilinx Inc.
2100 Logic Drive
San Jose, CA 94125

current CCM systems. As illustrated in Figure 1, it is envisioned to place both types of logic onto a single chip - a system called *OneChip.* We have also built a working prototype system that emulates the *OneChip* configuration. This allows us to examine the specifics of the interfacing requirements of such a system. We are then able to make estimates on the performance potential of a true *OneChip* system.

In Section 2, we discuss previous work and point out some architectural shortcomings. In Section 3, we discuss the details of our architecture as well as implementation issues. Section 4 describes the prototyping environment and the *OneChip* prototype used to evaluate our architecture. Section 5 presents two applications and describes area and performance results. Section 6 concludes our work and Section 7 suggests areas for continuing CCM research.

2. Previous Work

Various research organizations have recently studied the benefits of using reconfigurable logic for building CCMs [2][3][4][5][6][7]. Many interesting systems have been designed, spanning a broad range of interconnection architectures and usable gate counts. They can be broadly categorized by their degree of processor-coprocessor coupling and by the size of the applications to be executed on the reconfigurable part of the CCM:

- systems loosely linking reconfigurable logic to a fixed, front-end host computer
- systems loosely linking reconfigurable logic to a fixed, integrated CPU
- systems closely linking reconfigurable logic to a fixed, integrated CPU

An overview of representative systems is given below.

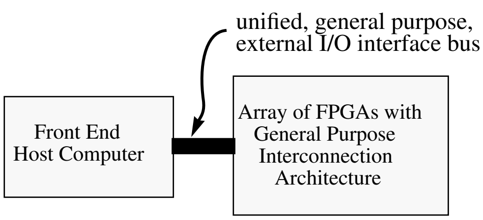

Fig. 2: Loosely Coupled CCM With Front End Host

2.1. Loosely Coupled CCMs With Front End Host

CCM systems of this category follow the configuration depicted in Figure 2. Systems include DEC's PeRLe [6], SRC's Splash[7] and NCSU's AnyBoard [5]. These three systems are referred to here, because they have been very successful at speeding up a broad range of applications.

2.2. Loosely Coupled CCMs With Integrated CPU

Employing a somewhat closer degree of coupling than the systems presented in the previous section, the CCMs of this category utilize separate, dedicated interconnect resources for control and data flow signals. Also, as illustrated in Figure 3, both types of signals no longer pass through a general I/O interface, but rather directly attach to the local bus and/or dedicated pins of the integrated CPU. Typical CCM systems include BU's Prism [3] and UofT's Reconfigurable Coprocessor [2].

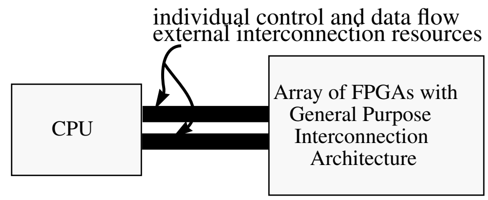

Fig. 3: Loosely Coupled CCM With Integrated CPU

2.3. Closely Coupled CCMs With Integrated CPU

In general, systems of this category attempt to link reconfigurable resources with fixed logic on a single chip as shown in Figure 4. Work of this category includes Harvard's PRISC [8], MIT's DPGA coupled microprocessor[9] as well as BYU's Nano Processor [10] and DISC system [11].

While MIT's work does not present a detailed new fixed / reconfigurable logic interface, it clearly identifies the communication bandwidth and latency of such an interface as the throughput limiting factors.

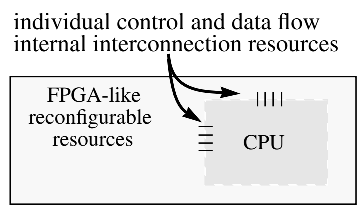

Fig. 4: Closely Coupled CCM With Integrated CPU

Harvard's proposed CCM has only been explored in simulations, though its architecture is clearly defined. It is restricted to small grain hardware functions that can be evaluated in a single clock cycle of the fixed logic CPU.

BYU's work is not limited to single cycle latencies. However, the fixed-logic core processor or static control constructs are only considered to be another module to be downloaded into an FPGA. Issues dealing with explicit fixed silicon are not considered and the eight-bit core processor is not very powerful.

2.4. Comments On CCM Architectures

While the above sampling of CCM systems is by no means complete - an up to date list can be obtained from [12] - most of the existent reconfigurable compute engines can be included into one of the presented fixed/reconfigurable logic interface categories. When analyzing the architectural concepts and the shortcomings of the three classes, several interesting observations can be made.

Amongst the loosely integrated systems, application granularity is particularly important. Significant speedups and super-computer like performance are only achieved with systems where the amount of computation done in the reconfigurable hardware is relatively large compared to the required communication overhead of the functions of the particular application being realized in hardware. To quantify this relative measure one must consider the following equation. It must be satisfied for any speedup to occur:

$$T_H + T_{OV} < T_S$$

T_H = time to execute function in hardware

T_{OV} = time to communicate data and control overhead

T_S = time to execute function in software (or main funct. unit)

This equation may be rewritten as follows:

$$\frac{T_H}{T_S} + \frac{T_{OV}}{T_S} < 1$$

The first fraction represents the actual hardware computational speedup and the second fraction is indicative of the granularity of a given application on a given CCM. A small T_{OV}/T_S ratio (say < 0.1) indicates a larger grain size, while a larger T_{OV}/T_S ratio (say > 0.5) represents a smaller grain size. The actual ratio depends on the characteristics of a given application running on a given CCM system. The sum of both T_H/T_S and T_{OV}/T_S is indicative of the overall speedup - the smaller this sum, the larger the overall speedup of a particular application on a particular CCM system (for sums less than one).

Clearly, CCM systems with a small T_{OV}/T_S ratio require only a smaller hardware speedup (T_H/T_S) to achieve the same overall speedup $(T_H + T_{OV}) / T_S$ than CCM systems whose T_{OV}/T_S is larger, similar to Amdahl's law [1]. In closely coupled CCMs, where the communication overhead is lower, the grain size of the functions implemented in hardware can be reduced. To enhance the performance of as wide a variety of applications as possible, the minimum required grain size for application speedup must be made relatively small by keeping the pertinent communication overhead associated with the CCM architecture at a minimum.

The architecture and structure of the reconfigurable logic required for CCMs has not been studied in detail. Most CCMs described above utilize general purpose FPGA structures to build their flexible logic resources. Only the recently proposed tightly integrated CCM systems stray from this path. Issues like configuration context switching for multitasking and time-sharing applications [8][9][13] and reconfigurable structures for special purpose applications ranging from very fine grain [14] to datapath oriented [15] designs have to be considered.

The explicit use of reconfigurable logic in embedded controller applications has not yet been explored by present CCM systems. Issues involved in the attachment of customizable preprocessing glue logic to a microprocessor have yet to be identified and addressed.

3. Architecture

As outlined in the introduction, the motivation behind this work is to investigate the benefits of a tight integration of reconfigurable resources into fixed logic, inspired by the shortcomings of existent loosely-coupled CCM systems. However, when confronted with the opportunity to add reconfigurable resources into the heart of a processor, instead of just to an external bus, the question of where and how to do so arises.

3.1. Requirements

Based upon the observed limitations of the fixed to reconfigurable logic interfaces discussed in Section 2.4, the following interfacing strategies should be avoided:

- reconfigurable logic attached to the memory bus or any other kind of processor external bus
- a problem specific interfacing solution requiring reconfiguration of the complete processor
- reconfigurable resources not controllable by the standard scheduling and forwarding modules of the core processor
- approaches resulting in an instruction format change/incompatibility with a MIPS-like processor [1]

The data flow and control flow bandwidths of both fixed and reconfigurable execution units must be matched closely. Any kind of interfacing scheme that attaches reconfigurable resources to a slow bus operating either asynchronously or at a lower clocking frequency than the internal datapath of the processor will always be a bottleneck for the computing operations of a CPU. Using a bus that operates slower than the processor's internal datapath as an interfacing point must be avoided.

Only those processor resources required to enhance the computing performance of a particular application should be built using reconfigurable logic.

1. We chose MIPS, because the basic architecture and instruction set are simple to modify and adapt to, and because of the availbale software. Other architectures would also be suitable.

Customizing the whole processor and not just some execution units as suggested in [16] can be costly, as the development time, if done manually, for customized control and datapath structures is considerable. Also, on a direct comparison basis, the implementation of common structures required by most processors like the program counter and the register file using reconfigurable resources is less dense and slower than a custom silicon design, unless such structures are also optimized on a per application basis. This fact has caused the designers of a low resource processor to stress the need for minimizing the use of reconfigurable resources [10].

With the addition of reconfigurable logic to the instruction decode (ID) or the memory access (MEM) stages of the basic five-stage MIPS processor pipeline it would be possible to perform some limited amount of pre and/or post execute (EX) stage data processing. The main computations would still be handled by the execution unit in the EX stage. However, extra control logic would be required to manage ID and MEM stage processing and extra routing resources would be needed to provide the forwarding of data to these new processing elements. It remains questionable whether this extra logic and routing hardware justifies the small performance improvement expected from ID and MEM stage processing.

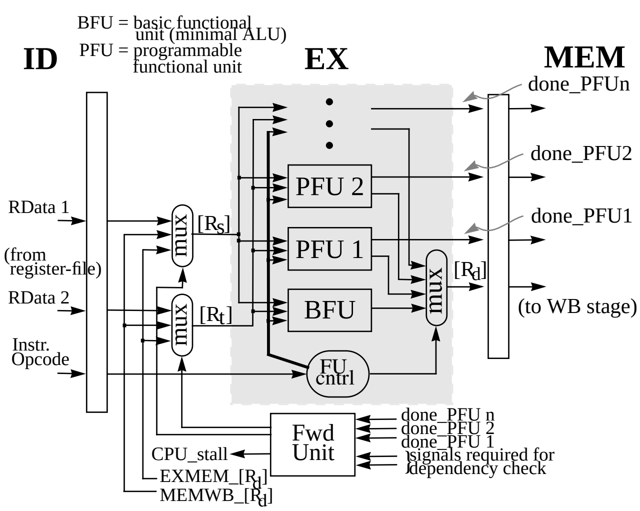

Fig. 5: Integration of ReconfigurableLogic into Programmable Functional Units

Apart from integrating reconfigurable resources into the regular datapath of the existing pipeline stages of the CPU, other more creative interfacing strategies have also been considered. They involved sacrificing the standard MIPS-like six-bit opcode [17] in favour of an application-specific instruction encoding scheme. This change in instruction format and the resulting loss of binary compatibility with basic, unenhanced MIPS executables is not desirable. Recompilation of any code should not be required, unless a performance gain through the use of the new reconfigurable features is desired.

3.2. Fixed/Reconfigurable Logic Interface

All the experience gained from the above considerations have led to the selection of an interfacing scheme as illustrated in Figure 5. The reconfigurable resources are added in parallel to the already existing basic functional unit (BFU) in the form of programmable functional units (PFU). The BFU is responsible for some elementary, arithmetic and logical operations required by many programs. It is built from fixed logic. The PFUs, however, can implement many application specific functions: any combinational or sequential circuit. They can also be used as programmable glue logic with some minimal pre- and/or post I/O processing features. The number and the boundaries of the PFUs depicted in Figure 5 may vary depending on the needs of a particular application. The above interfacing strategy has the following advantages:

- Tight integration: equally high data bandwidths for both BFU and PFU
- Use of standard MIPS datapath and functional modules: logic other than reconfigurable resources can be made small and fast
- Standard control and forwarding schemes: the control logic (FU cntrl) can be easily extended to address reconfigurable resources; the data dependency analysis check and the forwarding are performed as in a standard MIPS CPU; even multi-cycle PFU latency is handled at the minimal cost of an extension of the forwarding unit (fwd unit)
- Binary code compatibility with standard MIPS processors: a BFU plus default PFU configurations can implement all standard functions found in a regular CPU; the PFU configuration can be traded for different computational or interfacing requirements

A similar reconfigurable logic interfacing scheme can be found in Harvard's PRISC project [8]. However, their PFUs can only implement small combinational functions that have an overall logic delay limited to one CPU clock period. This limitation makes their architecture only interesting for applications exhibiting opportunities for bit-level optimizations. Also, the lack of flip flops limits the usefulness of the PFUs as glue logic in micro-controller applications, because the frequently required synchronization of I/O signals cannot be performed.

3.3. Custom Implementation

The prototype of the *OneChip* system, developed as part of this work, is implemented on the

Transmogrifier-1 (TM-1) field-programmable system using commercially available FPGA technology [18]. However, in a custom silicon implementation, TM-1 specific issues need no longer be considered and problem specific physical features can be crafted. The following discussion deals with physical implementation issues involved in:

- the design of the reconfigurable logic structures
- the relative placement of the pins and the various fixed and flexible logic blocks
- the structuring and layout of the routing resources
- the silicon area requirements relative to a XC4010

The custom silicon implementation of *OneChip* provides for a means of switching between PFU contexts as a new PFU image is required. Time-domain multiplexed FPGA architectures have also been proposed by Jones [19] and Bolotski, et al [13]. A tightly packed configuration memory is used to efficiently store the pre-compiled PFU images. This approach is much more area efficient than the selective run-time disabling of pre-compiled, static PFU images using the general-purpose Xilinx logic structures. Jones reports area savings by a factor of four over Xilinx look-up table mapped designs.

In addition to these optimizations of the configuration memory, a restructuring of the circuit-state or computational memory is used to further reduce the required silicon area. Should a given circuit require larger amounts of storage elements, the distributed style of memory found in the XC4010s does not represent a very area efficient design approach. This issue has been realized by the designers of Altera Corporation, who have included embedded circuit-state memory arrays into their new Flex 10k CPL device [20]. The custom silicon implementation of *OneChip* employs an embedded, area efficient, circuit-state memory array to be shared by all PFUs.

Further area reductions are gained through the use of a mixture of conventional XC4010-style reconfigurable resources employed for the implementation of single bit-based control constructs as well as datapath-style resources utilized for repeated logic structures where programming bit sharing can be employed. Cherepacha [15] notes that silicon area savings of up to 50% over an FPGA architecture not employing programming bit sharing may be obtained.

The proportions of control-style, datapath-style and embedded RAM-type logic of typical *OneChip* applications are yet to be determined. However, the proposed relative placement of the logic structures and the pins is depicted in Figure 6. It is known (see below) that the area of the PFUs implementing complex functions in reconfigurable logic resources will be much larger than the area of the fixed logic structures inside the processor core. This observation leads to a central placement of the smaller processor core immersed in the middle of a relatively larger sea of reconfigurable logic, where the distances between points in the fixed and in the reconfigurable structures are symmetrically balanced.

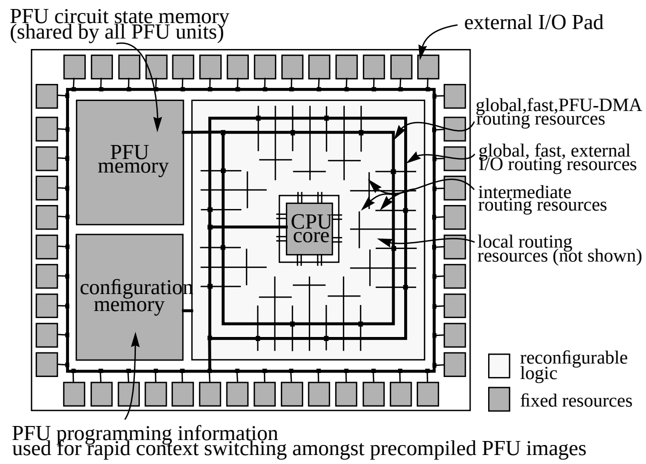

Fig. 6: Physical Layout of OneChip Logic Resources

The two internal memory blocks are tightly packed, regular structures that are placed to the side of the reconfigurable resources block where they can be accessed most easily by the routing resources interconnecting them with the sea of reconfigurable logic. The large number of external I/O pads is placed on the circumference of the block outlined by the previously mentioned structures.

The routing resources of *OneChip* are similar to the ones found in Xilinx devices. Local and intermediate length resources are evenly spread throughout the logic resources. Additional global routing tracks are provided for fast access to the PFU memory and the pad routing ring.

The die area of a 0.8 μm Xilinx 4010 FPGA has been measured from a broken device and found to be approximately 144 mm^2. An early 32-bit MIPS-like processor implementation described in [17] measures 6.24 mm^2 when appropriately scaled to account for the improved process technology. These figures indicate that a complete MIPS-X processor will fit into only 4.33% (6.24 / 144) of the area required by a XC4010, assuming that *OneChip* employs the same amount of reconfigurable resources found on a XC4010.

4. Prototype Setup

Even though the *OneChip* architecture described in the previous section employs fixed hardware resources for the implementation of its core processor, a different implementation approach was taken during the development phase. For the evaluation of architectural issues, a field-programmable system consisting of multiple FPGAs was used.

4.1. Prototyping Environment

The Transmogrifier-1 (TM-1) field-programmable system (FPS) developed at the University of Toronto [18] consists of four Xilinx 4010 FPGAs, two Aptix AX1024 field-programmable interconnect chips (FPIC) and four 32k x 9 SRAMs. It is connected to the SBus of a Sparc5 workstation from where the board can be configured. When programmed, circuit behaviour can be monitored on an additional set of connectors using the SBus connection or other external circuitry including oscilloscopes and logic analyzers.

4.2. Design Flow and Prototype Implementation

The *OneChip* prototype was implemented using an automated VHDL to TM-1 design flow. High-level language synthesis [21] was chosen as the design entry mechanism over a schematic based approach, because the latter can be time consuming for bigger designs and can require extensive work when regular, repetitive structures must be rewired.

Our core processor is completely described in VHDL [22]. Its functionality is based upon the MIPS-like processor outlined in P&H [23]. Due to the limited amount of logic resources of our prototyping system, only six of the processor's 32 registers are implemented. However, our design includes the hooks required for a full processor implementation. The use of VHDL synthesis allowed rapid architectural changes to explore interfacing issues and the quick addition of hardware functions representing configured PFU images.

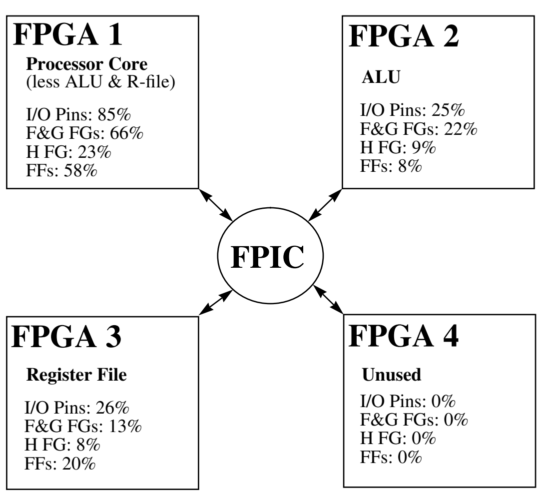

Fig. 7: TM-1 Processor Partitioning And Device Utilization

Figure 7 illustrates the final implementation of the basic *OneChip* system along with the logic resource utilization for each of the TM-1 FPGAs. The implementation of the static processor core uses all the resources of FPGA 1 (routing problems for > 66% logic utilization) as well as a small part of each of FPGAs 2 and 3. The remaining resources of FPGAs 2, 3 and 4 can be used to implement customized functional units. Examples will be discussed in Section 5.

4.3. Core Processor Performance

Interconnect related delays limits the performance of the processor. Pin constraints on the partitioned *OneChip* core required the use of time-division multiplexing to feed up to eight signals across a shared physical wire, similar to MIT's Virtual Wires project [24]. The maximum clocking frequency for the virtual wire, time-division multiplexing logic on the TM-1 is 10 MHz which results in a 1.25 MHz operation of the main system.

5. Results

To demonstrate the benefits of *OneChip*'s tight integration scheme several applications have been coded into PFU images. Typically, *OneChip* is useful for two types of applications:

- embedded controller type problems requiring custom glue logic interfaces
- application specific accelerators utilizing customized computation hardware

For both types of applications, the TM-1 field-programmable system is configured with the basic MIPS-like processor which is augmented with one or more programmable functional units (PFU). Note that the TM-1 based prototype does not allow for the use of any of *OneChip*'s optimized features described in Section 3.3. Also, the PFUs of the processor are configured uniquely at the time of TM-1 power-up, depending on the needs of the given application. For either application type, a multiplexer is used to select amongst the basic functional unit (BFU) and the PFU images.

5.1. Embedded Controller Type Applications

Well established, cheap, easily programmable CPUs like Motorola's 68000 series have become the basic building block for the cost sensitive yet large and lucrative embedded controller applications market. These generic embedded processors attempt to provide as much interfacing functionality as a target application field may require. With the availability of reconfigurable logic in between the external pads and the processor core, *OneChip* is not only able to reduce the logic complexity of the general-purpose controller interfaces, but can also provide those interfacing solutions which may not at all be conceivable with a fixed, generic embedded controller.

To demonstrate the usefulness of *OneChip*'s PFU-based reconfigurable logic integration scheme for embedded controller type applications, several glue logic configurations are considered. The Motorola MC68306 applications [25] have been implemented, because VHDL code describing their functionality could be obtained from another project undertaken in the same research department [26].

The efficiency of *OneChip*'s tight glue logic integration scheme is best illustrated by a simple code sequence. Assuming that one of the PFUs is programmed as a universal asynchronous receiver and transmitter (UART), *OneChip* would only execute two instructions to receive a data word into memory:

```
URTR   $reg            (UART read instr.)
 SW    mem_loc, $reg   (STORE WORD instr.)
```

URTR represents a new instruction which sends the appropriate control signals to the UART PFU and receives a data word from the PFU. In a traditional memory mapped processor, more instructions would have to be executed in a loop to test status bits and to access data. The PFU scheme is much more efficient.

Simple models of the parallel port, the DRAM controller and the universal asynchronous receiver and transmitter (UART) modules of the MC68306 micro-controller are implemented in the reconfigurable structures of a Xilinx 4010 FPGA to obtain area estimates.

Clearly, the major advantage of using reconfigurable controller glue logic lies in the flexibility of being able to customize interfacing logic on an application basis and in the possibility of reusing one CCM system for several different embedded controller-type applications. However, the implementation of logic in reconfigurable structures is not as dense as in full custom layouts. The customization of glue logic interfaces can reduce the overall logic resource requirements by omitting unused structures found in general, fixed interfaces, but an overall penalty of a larger silicon area for the glue logic interface must be paid when reconfigurable resources are utilized for its implementation.

Table 1 presents a comparison of the area of equivalent custom silicon and reconfigurable logic implementations of the presented applications. Entries include unoptimized UART and DRAM controller circuits as well as several implementations of a Parallel Port (PP). The first PP entry represents the unoptimized case. The second PP entry has been optimized by simplifying the addressing scheme as well as the port directionality logic. The third entry represents a generic case to illustrate the achievable flexibility with the reconfigurable *OneChip* architecture.

Application	Custom Silicon Implementation	FPGA Implementation	
	area (mm^2) *I	# of packed CLBs	area (mm^2) *II
UART	2.88	124	44.6
DRAM Controller	2.17	141	50.8
Parallel Port memory addressed, 16x1-bit I/O ports bit-wise programmable	0.80	53	19.1
Parallel Port instruction addressed, 8 input, 8 output ports, fixed directionality	0.23	8	2.88
Parallel Port instruction addressed, 9 input, 19 output ports, fixed directionality	0.394	14	5.040

*I Synopsys' Design Analyzer
*II die area of XC4010 (144 mm^2) multiplied by (# of CLBs utilized / 400)

Table 1: Logic Resource Utilization of Embedded Controller -Type Applications

The area of the custom silicon implementations was obtained by synthesizing the minimal VHDL code constructs of the controller-glue applications into a 0.8 µm BiCMOS standard cell library using the Synopsys Design Analyzer [27] along with an estimate for the routing area. The resource requirements of the FPGA implementations of the given applications are obtained by multiplying the total FPGA die area by the ratio of the number of actually required CLBs to the number of available CLBs.

Considering the logic optimization achievable through the use of reconfigurable resources, the implementation related area penalty can be reduced. A comparison of the areas of the unoptimized custom implementation of the parallel port versus the area of the optimized FPGA implementation of the same module results in the following calculation. The area penalty is evaluated to be 3.60, corresponding to 2.88 mm^2 divided by 0.80 mm^2. Clearly, the design flexibility introduced with the use of reconfigurable logic can be employed to significantly reduce the implementation technology related area penalty from a factor of 23.8 (19.1 / 0.80) to another factor as low as 3.60 with the benefit of being able to implement the exact logic interface required.

The third parallel port example represents a PP configuration that cannot be realized in the existing controller logic, showing the advantage of the reprogrammable *OneChip* approach.

5.2. Performance Enhancement Applications

In this section it will be shown that the tight integration of reconfigurable resources into *OneChip*'s architecture makes it useful for enhancing the performance of even those applications in which the required communication bandwidth is relatively high and the grain size is relatively small.

The sample application chosen for implementation is the Discrete Cosine Transform (DCT). In its one-dimensional form (1-D DCT), it performs several sequential add and multiply operations on eight-bit quantities, requires multiple clock cycles to complete and never fully utilizes the 32-bit datapath found in *OneChip*'s core processor. The convolution of the two-dimensional DCT (2-D DCT) may be separated and implemented as two sets of eight 1-D DCTs separated by a transpose [28], an operation exceeding the amount of local store provided by the register file of *OneChip*'s core. The 2-D DCT is responsible for most of the computations performed in typical JPEG coders and still amounts to about 40% of the computations performed during MPEG playback [29].

Clearly, the DCT represents an application that can benefit from customized hardware constructs. Furthermore, the grain sizes of the one-dimensional and the two-dimensional versions represent a small grain and a large grain problem in one application, respectively. The former can be used to demonstrate that *OneChip*'s tightly integrated architecture does not suffer from an I/O bandwidth bottleneck and the latter can be used to demonstrate significant performance enhancements, which are achievable with customized execution units of larger functional grain sizes.

For comparison purposes, it is desirable to evaluate the performance of hardware-only *OneChip* DCT variants with the equivalent software-only DCT implementations. Realizing that the TM-1 based *OneChip* system with its extremely slow processor clocking frequency does not represent a commercially viable system, the following comparison setup will be used:

- a MIPS R4400-based machine enhanced with reconfigurable features configured for DCT computations versus a fixed MIPS R4400 machine

Such a system is expected to employ an existing, realistically fast MIPS R4400 processor that has been augmented with the reconfigurable structures used by the *OneChip* architecture.

The pertinent performance information required for the comparisons outlined above are the execution times of both a memory access and the actual DCT operation of systems. The following setups were used to obtain the individual times:

- the prototype *OneChip* environment configured with a hardware DCT PFU image (to determine execution time of actual DCT operations)
- an SGI Indy workstation using a 150Mhz R4400 CPU with 512 MB main memory, 16 kB instruction and data caches and a 1MB unified L2 cache (to determine execution time of memory accesses)

With the above execution times, system B can be compared to system A (see Table 2).

System		**Memory Access**	**DCT Computation**
A	commercial R4400	R4400 specific (fast)	R4400 general-purpose ALU (slow)
B	envisioned commercial OneChip	R4400 specific (fast)	TM-1 DCT-optimized PFU (fast)

Table 2: Overview Of CCM Systems Used In Performance Evaluation

Several different DCT variants were included in the comparison study to uniquely identify the effects of two other optimization features besides the already mentioned hardware evaluation of the DCT algorithm. These additional features include loading or storing numerous bytes per memory operation (four bytes on the 32-bit prototype *OneChip* system, eight bytes on the 64-bit R4400 based systems) and the use of 512 (= eight rows times eight columns of eight-bit intermediate picture element data) PFU internal registers to reduce memory access requirements of the 2-D DCT. The core processor's register file is not large enough to locally store this intermediate data. The resulting register spill would require several memory accesses. Incorporating the two new features, the following five unique DCT versions can be studied:

- a one-dimensional DCT that loads or stores one byte per memory access (Version I)
- a one-dimensional DCT that loads or stores numerous bytes per memory access (four bytes for a 32-bit system, eight bytes for a 64-bit system) (Version II)

- a two-dimensional DCT that loads or stores one byte per memory access and performs the transpose operation using the regular processor registers (Version III)
- a two-dimensional DCT that loads or stores numerous bytes per memory access (four bytes for a 32-bit system, eight bytes for a 64-bit system) and performs the transpose operation using the regular processor registers (Version IV)
- a two-dimensional DCT that loads or stores numerous bytes per memory access (four bytes for a 32-bit system, eight bytes for a 64-bit system) and performs the transpose operation using the PFU internal registers (Version V)

It is expected that improvements in FPGA technology in conjunction with the architectural optimizations discussed in Section 3.3 will allow the DCT PFU pipeline to be clocked at faster frequencies than the lower bound of 1.5 Mhz quoted for the interconnect limited TM-1 prototype. PFU clocking frequencies chosen for this study include 2.5, 25 and 50 Mhz, representing rates achievable on the TM-1, in a single present-day technology FPGA with an unpartitioned design, and in an anticipated future technology FPGA, respectively. Figure 8 presents the speedup results obtained with the envisioned commercial *OneChip* CCM relative to the software-only R4400 execution of the various DCT versions.

Using the speedup results just calculated, several interesting observations can be made. Clearly, the clocking frequencies of the fixed and the reconfigurable parts of a *OneChip*-like CCM system must be matched. It makes little sense to augment a 150 Mhz CPU with extra computational hardware operating only at 2.5 Mhz. The speedups of the *OneChip* system with the slow processor are limited and may only be improved upon by increasing the clocking frequencies of both the core processor and the PFU logic. The performance gains of the enhanced R4400, *OneChip*-like machine, however, can be further raised as faster programmable logic resources become available. Figure 8 suggests diminishing returns as the operating frequencies of the core processor and the reconfigurable structures get closer to one-another.

A more significant observation concerns the grain size of the enhanced application. It is noted that even for the one-dimensional DCT, which represents the small grain application, speedups of more than a factor of ten are achieved with the tightly integrated *OneChip* architectures. Optimizations, such as the use of multiple data packed in a single memory word, that are only realizable with a tight reconfigurable logic interfacing scheme make the PFU model interesting for the speedup of even relatively small size problems. The extension of the DCT PFU with local store for intermediate results brings the achievable speedup to a factor of more than forty

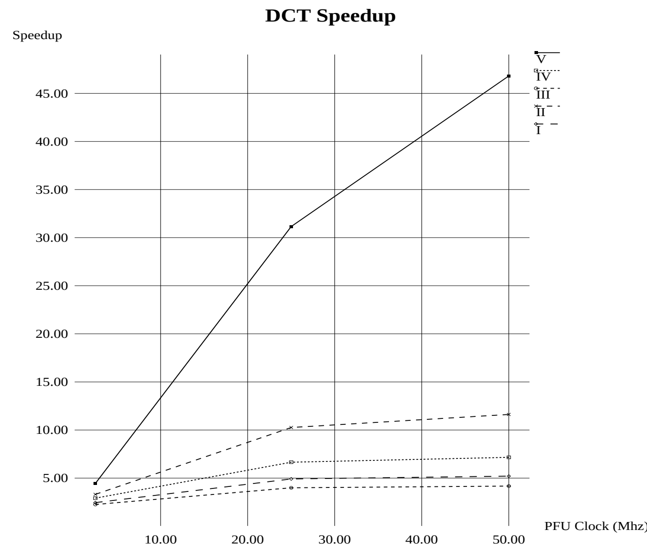

Fig. 8: DCT Speedup Using Envisioned Commercial OneChip System

while the granularity of the given problem is increased. Overall, the complexity of the hardware of the application presented in this section, represents a much smaller grain size than required by typical applications of more loosely coupled systems [4][6].

6. Conclusions

Early work in the area of reconfigurable computing presented promising speedup figures for large grain applications with low processor to reconfigurable logic communication requirements. The *OneChip* architecture overcomes these bandwidth limitations by directly linking reconfigurable structures into a functional block called a programmable functional unit (PFU). The PFU is tightly integrated into the processor pipeline in parallel to the basic functional unit (BFU) and can operate with an arbitrary latency.

Sample PFU images have been crafted to demonstrate the advantages of the *OneChip* architecture. For embedded controller applications the silicon area penalty to be paid for the added interface flexibility can be kept at a factor of less than 3.5 times the area of custom silicon implementations for fixed logic applications. For performance enhancement applications the *OneChip* architecture is also successful at overcoming bandwidth limitations of earlier CCM systems. In an example using the DCT as the application, it was shown that a speedup of close to 50 could be achieved using state-of-the-art technology to implement the *OneChip* system.

7. Future Work

Our paper has only investigated one small part of the area of reconfigurable computing. Before a user-friendly *OneChip* processor will be found in the heart of future commercial products the following issues must be addressed.

- The architectural requirements of *OneChip*'s core processor should be analyzed in greater detail. It has not been studied how much functionality of the fixed part of the *OneChip* system is actually used. A simpler, more compact core processor than the MIPS-like CPU employed in this work might be sufficient.
- The presented architecture has yet to be implemented in custom silicon to overcome the limitations of the inherently slow TM-1 prototype.
- It can be expected that sharing of custom hardware constructs across a set of processes will reduce hardware resource requirements.
- For the application of the *OneChip* architecture in multitasking environments some means of handling the computational state of several processes sharing one particular PFU must be provided.
- The software environment required for the programming and execution of application programs on the *OneChip* system has yet to be developed. It will be nontrivial.
- The design of a run-time operating system that handles dynamic PFU image compilation and automated context switching amongst multiple PFU images on a superscalar, time-shared system will be challenging.

Acknowledgments

Ralph D. Wittig was supported by an NSERC Postgraduate Scholarship and a UofT Open Fellowship. We also acknowledge NSERC and Bell-Northern Research who support the Transmogrifier project and Xilinx for their ongoing support of FPGA work at UofT.

References

[1] J. A. Hennessy, D. L. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kauffmann Publishers, San Mateo, CA, 1990.

[2] R. Jeschke, "An FPGA-Based Reconfigurable Coprocessor for the IBM PC", M.A.Sc.Thesis, University of Toronto, 1994.

[3] P. M. Athanas and H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis", *Computer*, March 1993, pp. 11-18.

[4] J. Arnold et al., "The Splash 2 Processor and Applications", *Proceedings International Conference on Computer Design*, October 1993.

[5] D. Van den Bout et al., "Anyboard: An FPGA-Based Reconfigurable System", *IEEE Design and Test of Computers, September 1992*, pp. 21-30.

[6] P. Bertin, D. Roncin, J. Vuillemin, "Introduction to Programmable Active Memories", *DEC Paris Research Laboratory Report #3*, 1989.

[7] M. Gokhale et al., "SPLASH: A Reconfigurable Linear Logic Array", *Proceedings International Conference on Parallel Processing*, August 1990, pp. 526-532.

[8] R. Razdan, M. D. Smith, "A High-Performance Microarchitecture with Hardware-Programmable Functional Units", *Micro 27*, November 1994, pp. 172-180.

[9] A. DeHon, "DPGA-Coupled Microprocessors: Commodity ICs for the Early 21st Century", *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94)*, April 1994.

[10] M. J. Wirthlin, B. L. Hutchings, K. L. Gilson, "The Nano Processor: a Low Resource Reconfigurable Processor", *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'94)*, Napa, CA, April 1994.

[11] M. J. Wirthlin, B. L. Hutchings, "A Dynamic Instruction Set Computer", *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'95)*, Napa, CA, April 1995.

[12] S. Guccione, "List of FPGA-based Computing Machines", Hyper text link: http://www.io.com/~guccione/HW_list.html

[13] M. Bolotski, A. DeHon, T. F. Knight Jr., "Unifying FPGAs and SIMD Arrays", *2nd International ACM/SIGDA Workshop on FPGAs (FPGA'94)*, February 1994.

[14] Xilinx Inc., *XC6200 FPGA Family*, San Jose, CA, 1995.

[15] D. Cherepacha, "A Field-Programmable Gate Arrary Architecture Optimized For Datapaths", M.A.Sc. Thesis, University of Toronto, 1994.

[16] J. Davidson, "FPGA Implementation Of A Reconfigurable Microprocessor", *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93)*, Napa, CA, April 1993.

[17] P. Chow, *The MIPS-X RISC Microprocessor*, Kluwer Academic Publishers, MA, 1989.

[18] D. Galloway, D. Karchmer, P. Chow, D. Lewis, J. Rose, "The Transmogifier: The University of Toronto Field-Programmable System", *Second Canadian Workshop on Field-Prgrammable Devices*, Kingston, ON, June 1994. Available as CSRI Technical Report 306 via anonymous ftp as ftp://ftp.csri.toronto.edu/csri-technical-reports/306/.

[19] D. Jones, D. M. Lewis, "A Time-Multiplexed FPGA Architecture for Logic Emulation", *Proceedings IEEE Custom Integrated Circuits Conference (CICC'95)*, Santa Clara, CA, May 1995.

[20] Altera Corporation, *Data Book*, San Jose, CA, 1995.

[21] Viewlogic Systems Inc., *ViewSynthesis User's Guide*, Marlboro, MA, August 1994.

[22] D.R. Coelho, *The VHDL Handbook*, Kluwer Academic Publisher, Norwell, MA, 1989.

[23] D. A. Patterson, J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Morgan Kauffman Publishers, San Mateo, CA, 1993.

[24] J. Babb, R. Tessier, A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators", *Proceedings IEEE Workshop on FPGAs for Custom Computing Machines (FCCM'93)*, Napa, CA, April 1993.

[25] Motorola, Inc, *MC68306 Integrated EC000 Processor User's Manual*, Motorola Literature Distribution, Phoenix, AZ, 1993.

[26] Paul Chow and Rob Jeschke, *CMC/University of Toronto Rapid-Prototyping Board Users Guide*, ICI-068, Canadian Microelectronics Corporation, 1995.

[27] Synopsys Inc, Synopsys Online Documentation V3.1, Mountain View CA, 1993.

[28] W. B. Pennebaker, J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, NJ, 1993.

[29] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors", *IEEE Micro*, April 1995, pp 22-32.