Compiling to FPGAs via an EPIC Compiler's Intermediate Representation

Zhiguo Ge Jirong Liao Weng-Fai Wong
Department of Computer Science
National University of Singapore
3, Science Drive 2
Singapore 117543

Abstract

The increasing density and speed of modern field programmable gate arrays offer the reconfigurable systems using them greater capability and flexibility, in particular for more complex computation. However, there remains a very important problem of how to design on a more abstract level to manage the vast hardware resource and shorten the design time. This paper presents an approach to compile a system level description to hardware through a conventional software intermediate representation (IR) of a state-of-the-art optimizing compiler for Explicitly Parallel Instruction Computing (EPIC) processors. The front end compiles C programs into an intermediate representation for an infinite resource EPIC processor. The intermediate representation contains all the information of control flow graph of basic blocks. It is from this intermediate representation that we have devised means to generate synthesizable Register Transfer level (RTL-level) Verilog description that can be mapped into the reconfigurable HW device. We will describe the details of the translation process and the performance on actual FPGA hardware.

1. Introduction

The development of the FPGA has led to the emergence of *reconfigurable computing systems* that mixes the properties of hardware and software. Typically, the hardware for such systems is typically made up of a FPGA component coupled tightly with a general-purpose processor [2,3]. Reconfigurable computing architecture provides the means for exploiting fine grain parallelism in computation, achieving significant speed-ups in many applications [1, 2].

With the significant increases in density and complexity of reconfigurable devices, the design methodology used for realizing solutions on such devices has become a serious issue. At the core of this issue is controlling the escalating non-recurrent engineering cost needed to quickly and correctly prototype a solution. It is necessary to design the system more abstract high level specification. Over the last decade, there has been much research in High Level Synthesis with a lot of encouraging results [5]. C and C++ have been popular choices as a basis for development as system level specification [6,7,8,13].

The inherent attraction of C is that its widespread use. One is almost certain to find a C implementation of just about any algorithm. This ready access to a large pool of tried and tested programs is good for reuse and rapid prototyping. However, C is inherently a serial programming language. Most of the constructs introduced in hardware flavors of C has to do with parallelism.

We took a different approach to the problem. Instead of having the programmer indicate parallelism and possibly re-work his algorithm, we used a state-of-the-art compiler infrastructure that is targeted at instruction level parallelism as our starting point. The input to the compiler is a standard C program. It is the compiler's responsibility to discover instruction level parallelism. In a previous paper[12], we reported on how we managed to obtain Handel-C hardware descriptions from the intermediate representation of the Trimaran instruction level parallel compiler called Elcor [10].

In this paper, we will describe how we transformed the Elcor intermediate representation to synthesizable RTL Verilog code. Handel-C is easy to learn and design, however, one has less freedom in direct manipulation of the cells in the FPGAs available with low-level hardware description languages such as Verilog and VHDL. Also, designing by Handel-C often result in lower clock frequencies. Our intention of changing from Handel-C to RTL Verilog is to hopefully achieve better performance by directly exploiting low level features of the FPGAs.

2. Hardware Platform

The reconfigurable computing platform we used in our development is the Celoxica RC1000-PP board [11], which connects to a PC host via the PCI bus. The RC1000-PP is a PCI bus plug-in card designed for Handel-C. It has one Xilinx Vertex XCV1000-4 FPGA [4] with four banks of SRAM memory. The XCV1000 FPGA is a SRAM-based FPGA that has 64×96 control logic blocks and a total system gate count that exceeds one

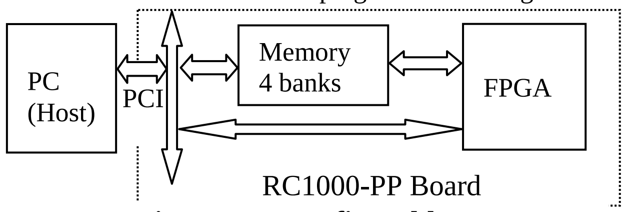

Figure 1. Reconfigurable System

million [4]. All four SRAM memory banks of 2MByte each are accessible by both the FPGA and any device on PCI bus. The PC host controls and programs the FPGA via PCI bus.

There are three possible means of communications between the host and the FPGA: (a) bulk data transfers via the memory banks, (b) communicating via two unidirectional 8-bits ports, and (c) two I/O pins provided for single-bit communication [11].

3. Design Flow

We used the Trimaran compiler infrastructure [10]. Trimaran is designed for compiler and architectural research in instruction level parallelism. The front-end of Trimaran compiles C programs into the Elcor intermediate representation. Control flow, data and control dependences, as well as many other attributes of the individual operators are captured in Elcor representations. We essentially wrote a new back-end that translates the Elcor output of the frontend to synthesizable RTL Verilog code. The design flow is shown in Figure 2.

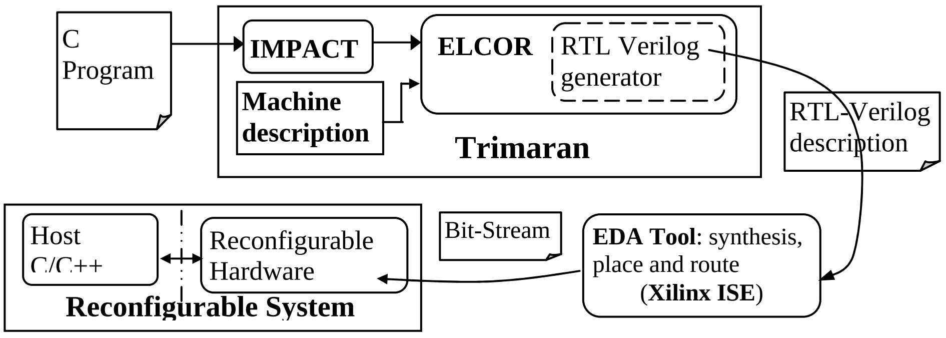

Figure 2. Design Flow

Trimaran is designed for compiling and simulating a parametric Explicitly Parallel Instruction Computing (EPIC) processor. The compiler performs a large suite of traditional high level compiler optimizations and VLIW-style instruction scheduling, register allocation as well as software pipelining. For our purpose, we used a Trimaran configuration for an infinite resource EPIC machine.

In the next section, we will describe the details of how we translate Elcor intermediate representations into synthesizable RTL Verilog.

4. Hardware Compilation via HFSM

4.1 Modeling the Elcor IR with Hierarchical Finite State Machines

There is a significant semantic gap when a software oriented programming language is used to describe hardware. For example, while jump instructions are very common in software, there are no jumps in hardware. Also while the processor fetches and executes the instructions sequentially (at least in semantically) from the instruction stream, execution in hardware is inherently parallel and spatial in nature. There is no instruction fetch and execution in hardware and the circuit implements all computations. Temporal scheduling must be done with great care for computation performed in hardware.

To overcome the gap, we need to model the software intermediate representation with the appropriate representation suitable for hardware. *Hierarchical Finite State Machine* (HFSM) is an ideal tool to model the Elcor intermediate representation. We use a two-level HFSM, shown in Figure 3, to model the program unit of Elcor IR. Note that this is method of translation is different from that used by Page and Luk in translating Occam and Handel-C to hardware [9]. They made use of the high-level structure of the code that is absent at the Elcor level.

The entire control flow graph consisting of basic blocks is mapped into the HFSM. Each node of the control flow graph, i.e. each basic block, is mapped into a unique second level FSM in the HFSM, as shown in Figure 3. Each basic block is a sequence of instructions and these instructions in the basic blocks are mapped into different states of second level FSM.

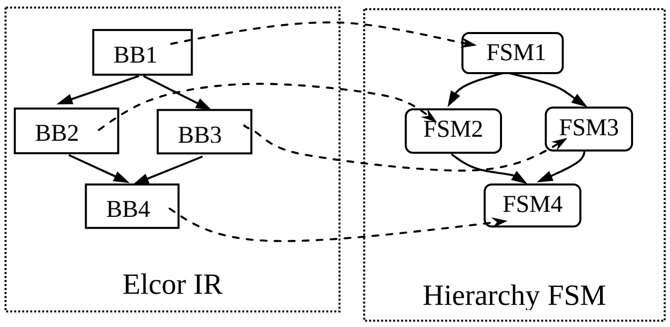

Figure 3. Modeling Elcor IR with HFSM

4.2 Instructions Scheduling:

Example 1 illustrates how instructions in a basic block of an Elcor intermediate code, listed in Figure 4, can be mapped into a second level FSM. In Figure 4, the important quantity to note is `s_time()`. `s_time()`is the scheduled time for a particular operation. This is computed by the Trimaran compiler for every instruction. Instructions that have same `s_time()` are to be executed in parallel. To convert the schedule into a FSM, we group those instructions that have the same `s_time()` into the same state of second level FSM. As a result, the scheduling of instructions in Figure 4 is as follows:

ADD_W.0, ADD_W.1	**: state 0**
L_W_C1_C1.0	**: state 1**
S_W_C1.1	**: state 2**
CMPP_W_LEQ_UN_UN.0	**: state 3**
BRCT.0	**: state 4**

Example 1: Mapping Elcor IR into FSM

```
bb 5 (
        weight(1)
        entry_ops(89) exit_ops(27)
        entry_edges(ctrl ^32 ctrl ^33) exit_edges(ctrl ^58 ctrl ^41)
        flags(sched)
        attr(lc ^112)
        subregions(
         op 89 (C_MERGE [] [] s_time(0) s_opcode(C_MERGE.0) ... )
         op 75 (ADD_W [br<52:i gpr 2>] [l:g_abs<_i> i<0>] p<t> s_time(0)  ...)
         op 74 (ADD_W [br<51:i gpr 3>] [l:g_abs<_j> i<0>] p<t> s_time(0) ...)
         op 76 (PBRR [br<53:b btr 2>] [b<7> i<0>] p<t> s_time(0) ...)
         op 26 (L_W_C1_C1 [br<15:i gpr 4>] [br<52:i gpr 2>] p<t> s_time(1) ...)
         op 25 (S_W_C1 [] [br<51:i gpr 3> i<0>] p<t> s_time(1) ...)
         op 77 (CMPP_W_LEQ_UN_UN [br<54:p pr 2> u<>] [br<15:i gpr 4>
i<10>] p<t> s_time(3) ...)
         op 27 (BRCT [] [br<53:b btr 2> br<54:p pr 2>] p<t> s_time(4) ...)
           )
          )
```

Figure 4 Elcor Intermediate Code for Example 1

Note that we have ignored pseudo instructions (such as the MERGE instruction) that are used by Elcor to make certain notes in the code.

4.3 Accessing external SRAM and other board resources

The RC1000-PP we used was designed for Handel-C. Using the board's resources, such as the external SRAM, the clock and reset signals, outside the context of Handel-C was not well documented. Direct communication with the FPGA was also not documented. To overcome these problems, we created a Handel-C module to take care of interfacing between the Verilog module and the host. This Handel-C intermediary is compiled into EDIF, which can then be connected with our Verilog module. The Handel-C module provided the necessary services for Verilog module by acting as its proxy. The interface between the Handel-C module with host and Verilog module is shown in Figure 5.

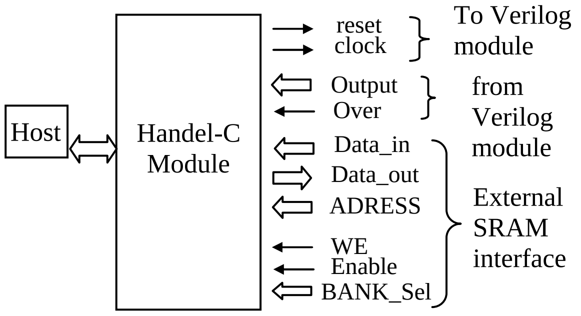

Figure 5: Interface with Handel-C

There are three groups of signals in the interface between the Handel-C and the Verilog modules. The first group provides the indispensable reset and clock signals to the Verilog module. The Handel-C module starts the computation circuit in FPGA using reset port. The second group consists of the output signals from the Verilog module. "Signal Over" indicates whether the Verilog module has completed its computation while the "Output" port is a 8-bits vector through which Verilog output a byte of data to the host by Handel-C module. The third group serves as the proxy for the external SRAM banks, making them directly accessible to the Verilog module.

Both the Handel-C and Verilog code are compiled to EDIF files so that the vendor EDA tool can generate the configuration bit-stream file.

4.4 Message chart between Host, Handel-C and Verilog:

Figure 6 shows how everything is put together for execution on the FPGA board.

First, the host loads the bit configuration file into the FPGA and starts the computation. The Handel-C proxy then starts the Verilog module, handling any memory access requests that the Verilog module may make. At the end of the computation, the Verilog module signals the Handel-C proxy module which then returns the result it receives back to the host.

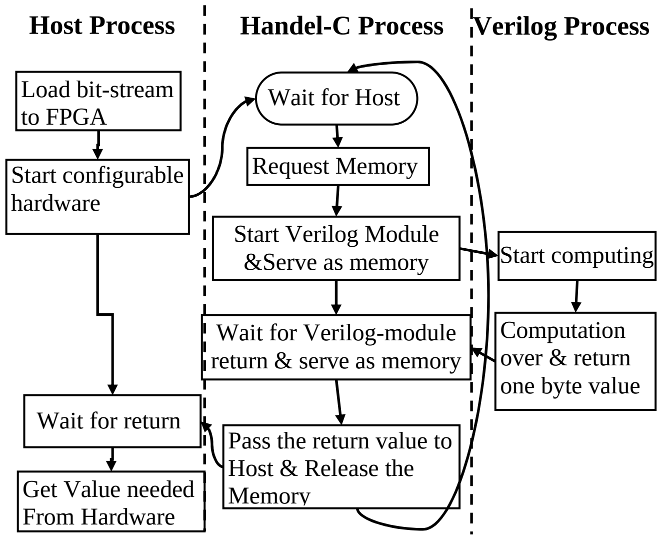
Host Process
Handel-C Process
Verilog Process
Load bit-stream to FPGA
Start configurable hardware
Wait for return
Get Value needed From Hardware
Wait for Host
Request Memory
Start Verilog Module &Serve as memory
Wait for Verilog-module return & serve as memory
Pass the return value to Host & Release the Memory
Start computing
Computation over & return one byte value

Figure 6: Message Chart for communication between the modules

5. Benchmarks

We used four benchmarks to compare the effectiveness of translating Elcor intermediate representation to RTL Verilog and Handel-C. These are: Finite impulse response (FIR), Matrix Multiplication (MM), Livermore Loop 1 (Lm1) and Histogram (Histo).

We use two parameters reported by the vendor's (Xilinx) EDA tool to evaluate the performance of the two schemes. The first parameter is the number of FPGA slices consumed by a design. This essentially shows the FPGA resource consumption of the code. The other parameter is the effective clock frequency at which the FPGA can run. The numbers of clock cycles taken in both cases are mainly dependent on the Elcor schedule of the code with some differences caused by the translation scheme. In any case, we measured the execution time using the host PC's timer. The results, measured in milliseconds, are shown in Table 1.

	Slice		**Exec. Time in msec (Frequency in MHz)**	
	Verilog	**Handel-C**	**Verilog**	**Handel-C**
MM	706	1016	486.3 (32.482)	743.4 (10.355)
FIR	585	914	0.765 (33.992)	1.45 (10.056)
Lm1	829	1098	0.588 (34.470)	0.886 (10.845)
Histo	264	332	0.329 (35.0)	0.312 (17.525)

Table 1: Performance comparison between translating to RTL-Verilog and to Handel-C

However, we have to point out some differences between the two output methods.

a) In the Verilog version, only one memory bank was used to accommodate variables. Different variables have different offsets in the same memory bank. In the Handel-C version, four memory banks are used.

b) The Handel-C version of the framework is able to translate Elcor hyperblocks [10] with predicated operations into Handel-C while the current Verilog version can only handle basic blocks. In Table 1, the MM benchmark uses hyperblocks while the others use basic blocks. Hyperblocks in EPIC machines expose more instruction level parallelism opportunites.

From Table 1, we can see an improvement of performance when Elcor intermediate representation is compiled into Verilog rather than Handel-C. On the average, the execution time of the Verilog version is about 70% that of the Handel-C version.

6. Conclusion

In this paper, we presented an approach of compiling the system level description into hardware via the intermediate representation of a compiler for an infinite resource EPIC machine. Through this design flow, a high level executable specification written in C that one can test and debug can be implemented quickly in reconfigurable hardware. We also showed that compiling Elcor IR to RTL-Verilog can achieve better performance in frequency and resource consumed than compiling Elcor IR into Handel-C HDL.

Using the framework we have constructed, we hope to conduct research in the following areas in the near future:

a) We wish to investigate various optimizations specific to reconfigurable computing systems.

b) By changing machine configurations to find a suitable balance between resource consumption and available instruction level parallelism, we believe it is possible to use the compiler as the vehicle to perform hardware-software partitioning.

7. Acknowlegement

This project is funded by A*STAR research project 012-106-0046.

References

[1] André DeHon. "The Density Advantage of Configurable Computing." *IEEE Computer*, vol.33, pp.41-49, 2000.

[2] André DeHon, John Wawrzynek. "Reconfigurable Computing: What, Why, and Implications for Design Automation." *Design Automation Conference (DAC)*, pp. 610 – 615, 1999.

[3] John R.Hauser, John Wawrzynek. "Garp: A MIPS Processor with a Reconfigurable Coprocessor." *Proc. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 16-18, 1997.

[4] Xilinx Corporation (http://www.xilinx.com), California.

[5] Raul Camposano, Wayne Wolf. *High-Level VLSI Synthesis*. Kluwer Academic Publishers, 1992.

[6] Giovanni De Micheli. "Hardware Synthesis from C/C++ Models." *Proc. of DATA'99*, p. 382.

[7] Donald Soderman. *Implementing C Designs in Hardware*. www.asicdsn.com/c2hw.pdf .

[8] David C. Ku, Giovanni De Micheli. *High Level Synthesis of ASICs Under Timing and Synchronization Constrains.* Kluwer Academic Publishers, 1992.

[9] Ian Page, and Wayne Luk. "Compiling Occam into Field-Programmable Gate Arrays." W. Moore and W. Luk (editors), Abingdon EE&CS Books, 1991, pp. 271-283

[10] Trimaran ILP Research Infrastructure, 1998. http://trimaran.org.

[11] Celoxica Company. http://www.celoxica.com/

[12] Jirong Liao, Weng-Fai Wong, and Tulika Mitra. "A Model for Hardware Realization of Kernel Loops." *Proc. of 13th International Conference on Field-Programmable Logic and Application (FPL 2003)*, pp. 334-344.

[13] Celoxica. *Handel-C Language Reference Manual* Version 3.1. 2002.