LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems

Andrew Canis[1], Jongsok Choi[1], Mark Aldham[1], Victor Zhang[1], Ahmed Kammoona[1], Jason Anderson[1], Stephen Brown[1], and Tomasz Czajkowski[‡]

[1]ECE Department, University of Toronto, Toronto, ON, Canada
[‡]Altera Toronto Technology Centre, Toronto, ON, Canada

legup@eecg.toronto.edu

ABSTRACT

In this paper, we introduce a new open source high-level synthesis tool called *LegUp* that allows software techniques to be used for hardware design. LegUp accepts a standard C program as input and automatically compiles the program to a hybrid architecture containing an FPGA-based MIPS soft processor and custom hardware accelerators that communicate through a standard bus interface. Results show that the tool produces hardware solutions of comparable quality to a commercial high-level synthesis tool.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

General Terms

Design, Algorithms

1. INTRODUCTION

Two approaches are possible for implementing computations: software (running on a standard processor) or hardware (custom circuits). A hardware implementation can improve speed and energy-efficiency versus software implementation (e.g. [3]). However, hardware design requires writing complex RTL code, which is error prone and difficult to debug. Software design, on the other hand, is more straightforward, and mature debugging and analysis tools are freely accessible. Despite the potential energy and performance benefits, hardware design is too difficult and costly for most applications, and a software approach is preferred.

In this paper, we propose *LegUp* – an open source high-level synthesis (HLS) framework we have developed that provides the performance and energy benefits of hardware, while retaining software ease-of-use. LegUp automatically compiles a C program to target a hybrid FPGA-based software/hardware system, where some program segments execute on an FPGA-based 32-bit MIPS soft processor and other program segments are automatically synthesized into FPGA circuits – *hardware accelerators* – that communicate and work in tandem with the soft processor. Since the first FPGAs appeared in the mid-1980s, access to the technology has been restricted to those with hardware design skills. However, according to labor statistics, software engineers outnumber hardware engineers by more than 10X in the U.S. [10]. An overarching goal of LegUp is to broaden the FPGA user base to include software engineers, thereby expanding the scope of FPGA applications and growing the size of the programmable hardware market.

LegUp includes a soft processor because not all program segments are appropriate for hardware implementation. Inherently sequential computations are well-suited for software (e.g. traversing a linked list); whereas, other computations are ideally suited for hardware (e.g. addition of integer arrays). Incorporating a processor also offers the advantage of increased high-level language coverage. Program segments that use restricted C language constructs can execute on the processor (e.g. recursion).

LegUp is written in modular C++ to permit easy experimentation with new HLS algorithms. The LegUp distribution includes a set of benchmark C programs [6] that can be compiled to pure software, pure hardware, or a hybrid system. In this paper, we present an experimental study demonstrating that LegUp produces hardware implementations of comparable quality to a commercial tool [13], and we give results demonstrating the tool's capabilities for hardware/software co-design.

2. RELATED WORK

Among prior academic work, the *Warp Processor* proposed by Vahid, Stitt and Lysecky bears similarity to our framework [12]. The Warp Processor profiles software running on a processor. The profiling results guide the selection of program segments to be synthesized to hardware. Such segments are disassembled from the software binary to a higher-level representation, which is then synthesized to hardware [9]. We take a somewhat similar approach, with key differences being that we compile hardware from the high-level language source code and our tool is open source.

On the commercial front is Altera's C2H tool [1]. C2H allows a user to partition a C program's functions into a hardware set and a software set. The software-designated functions execute on a Nios II soft processor, and the hardware-designated functions are synthesized into custom hardware accelerators. The C2H system architecture closely resembles that targeted by LegUp.

3. LEGUP FLOW AND ARCHITECTURE

The LegUp design flow comprises first compiling and running a program on a standard processor, profiling its execution, selecting program segments to target to hardware, and then re-compiling the program to a hybrid hardware/software system. Fig. 1 illustrates the detailed flow. Referring to the labels in the figure, at step ①, a C compiler compiles a program to a binary executable [7]. At ②, the executable

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FPGA'11, February 27–March 1, 2011, Monterey, California, USA.
Copyright 2011 ACM 978-1-4503-0554-9/11/02 ...$10.00.

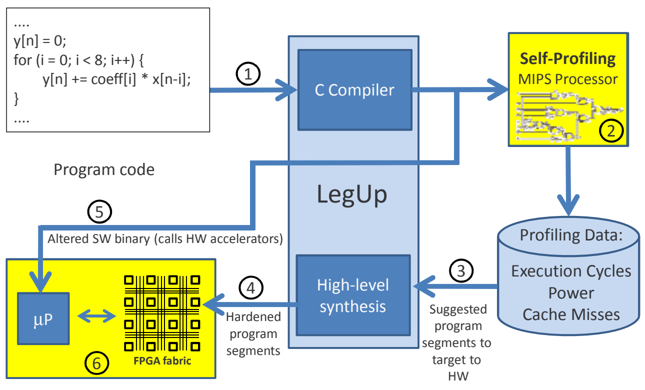
....
y[n] = 0;
for (i = 0; i < 8; i++) {
y[n] += coeff[i] * x[n-i];
}
....
Program code
C Compiler
Self-Profiling MIPS Processor
LegUp
Profiling Data:
Execution Cycles
Power
Cache Misses
Altered SW binary (calls HW accelerators)
High-level synthesis
Suggested program segments to target to HW
Hardened program segments
µP
FPGA fabric
1
2
3
4
5
6

Figure 1: Design flow with LegUp.

runs on an FPGA-based MIPS processor. We evaluated several publicly-available MIPS processor implementations and selected the Tiger MIPS processor from the University of Cambridge [11], based on its full support of the MIPS instruction set, established tool flow, and well-documented modular Verilog.

The MIPS processor has been augmented with extra circuitry to profile its own execution. Using its profiling ability, the processor is able to identify sections of program code that would benefit from hardware implementation. Specifically, the profiling results drive the selection of program code segments to be re-targeted to custom hardware from the C source. Profiling a program's execution in the processor itself provides the highest possible accuracy. Presently, we profile program run-time at the function level.

Having chosen program segments to target to custom hardware, at step ③ LegUp is invoked to compile these segments to synthesizeable Verilog RTL. LegUp's hardware synthesis and software compilation are part of the same compiler framework. Presently, LegUp HLS operates at the function level: entire functions are synthesized to hardware from the C source. The RTL produced by LegUp is synthesized to an FPGA implementation using standard commercial tools at step ④. In step ⑤, the C source is modified such that the functions implemented as hardware accelerators are *replaced* by wrapper functions that *call* the accelerators (instead of doing computations in software). This new modified source is compiled to a MIPS binary executable. Finally, in step ⑥ the hybrid processor/accelerator system executes on the FPGA.

Our long-term vision is to fully automate the flow in Fig. 1, thereby creating a *self-accelerating adaptive processor* in which profiling, hardware synthesis and acceleration happen transparently without user awareness. In the first release of our tool, however, the user must manually examine the profiling results and place the names of the functions to be accelerated in a file that is read by LegUp.

Fig. 2 elaborates on the target system architecture. The processor connects to one or more custom hardware accelerators through a standard on-chip interface. As our initial hardware platform is the Altera DE2 Development and Education board (containing a 90 *nm* Cyclone II FPGA), we use the Altera Avalon interface for processor/accelerator communication [2]. A shared memory architecture is used, with the processor and accelerators sharing an on-FPGA data cache and off-chip main memory. The on-chip cache memory is implemented using block RAMs within the FPGA fabric (M4K blocks on Cyclone II). Access to memory is handled by a memory controller. The architecture in Fig. 2 allows

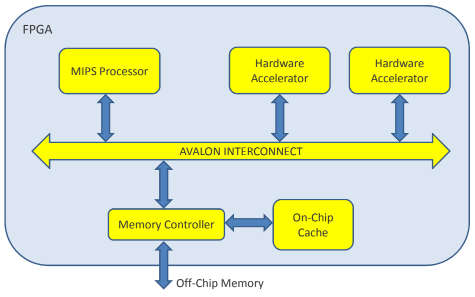

Figure 2: Target system architecture.

processor/accelerator communication across the Avalon interface or through memory.

The architecture depicted in Fig. 2 represents the target system most natural for an initial release of the tool. The architecture of processor/accelerator systems is an important direction for future research.

4. DESIGN AND IMPLEMENTATION

4.1 High-Level Hardware Synthesis

High-level synthesis has traditionally been divided into three steps [4]: allocation, scheduling and binding. Allocation determines the amount of hardware resources available for use, and manages other hardware constraints (e.g., speed, area, and power). Scheduling assigns each operation in the program being synthesized to a particular clock cycle (state) and generates a finite state machine. Binding saves area by sharing functional units between operations, and sharing registers/memories between variables.

LegUp leverages the low-level virtual machine (LLVM) compiler framework. At the core of LLVM is an intermediate representation (IR), which is essentially machine-independent assembly language. C code is translated into LLVM's IR then analyzed and modified by a series of compiler optimization passes. LLVM IR instructions are simple enough to directly correspond to hardware operations (e.g., an arithmetic computation). Our HLS tool operates directly with the LLVM IR, scheduling the instructions into specific clock cycles. LegUp HLS algorithms have been implemented as LLVM passes that fit neatly into the existing framework. Implementing the HLS steps as distinct passes also allows easy experimentation with different HLS algorithms; for example, one could modify LegUp to "plug in" a new scheduling algorithm.

The initial release of LegUp uses as-soon-as-possible (ASAP) scheduling [5], which assigns an operation to the first state after its dependencies are available. In some cases, we can schedule an instruction into the *same* state as one of its dependencies. This is called *operation chaining.* Chaining can reduce hardware latency (# of cycles for execution) without impacting the clock period.

Binding consists of two tasks: assigning operators from the program being synthesized to specific hardware units, and assigning program variables to registers (register allocation). When multiple operators are assigned to the same hardware unit, or when multiple variables are bound to the same register, multiplexers are required to facilitate the sharing. We make two FPGA-specific observations in our approach to binding. First, multiplexers are relatively expensive to implement in FPGAs using LUTs. A 32-bit multiplexer implemented in 4-LUTs is the same size as a 32-bit adder. Consequently, there is little advantage to sharing all but the largest functional units, namely, multipliers and dividers. Likewise, the FPGA fabric is *register rich* and sharing registers is rarely justified. The initial relase of LegUp uses a weighted bipartite matching heuristic to solve the binding problem [8]. We minimize the number of multiplexer inputs required, thereby minimizing area.

4.2 Processor/Accelerator Communication

Functions selected for hardware implementation are automatically replaced with a wrapper by the LegUp framework. The wrapper function passes the function arguments to the corresponding hardware accelerator, and receives the returned data over the Avalon interconnect. While waiting for the accelerator to complete its work, the MIPS processor can do one of two things: 1) continue to perform computations and periodically poll a memory-mapped register whose value is set when the accelerator is done, or, 2) stall until a done signal is asserted by the accelerator. The advantage of polling is that the processor can execute other computations while the accelerator performs its work. The advantage of stalling is reduced energy consumption – the processor is in a low-power state while the accelerator operates. In our initial LegUp release, both modes are functional; however, we use stalling for the results in this paper.

4.3 Language Support and Benchmarks

LegUp supports a large subset of ANSI C for synthesis to hardware including: control flow statements, all integer arithmetic and bitwise operations, and integer types. Program segments that use unsupported language features are required to remain in software and execute on the MIPS processor. LegUp also supports functions, arrays, structs, global variables and pointer arithmetic. Dynamic memory, floating point and recursion are unsupported in the initial release.

With the LegUp distribution, we include 13 benchmark C programs. Included are all 12 programs in the CHStone high-level synthesis benchmark suite [6], and Dhrystone – a standard integer benchmark. A key characteristic of the benchmarks is that inputs and expected outputs are included in the programs themselves. The presence of golden outputs for each program gives us assurance regarding the correctness of our synthesis results.

5. EXPERIMENTS

The goals of our experimental study are three-fold: 1) to demonstrate that the quality of result (speed, area, power) produced by LegUp HLS is comparable to that produced by a commercial HLS tool (eXCite [13]), 2) to demonstrate LegUp's ability to effectively explore the hardware/software co-design space, and 3) to compare the quality of hardware vs. software implementations of the benchmark programs.

We map each benchmark program using 5 different flows, representing implementations with increasing amounts of computation happening in hardware vs. software: 1) A software only implementation running on the MIPS soft processor (*MIPS-SW*); 2) A hybrid software/hardware implementation where the *second most* compute-intensive function (and its descendants) in the benchmark is implemented as a hardware accelerator (*LegUp-Hybrid2*); 3) A hybrid software/hardware implementation where the *most* compute-intensive function (and its descendants) is implemented as a hardware accelerator (*LegUp-Hybrid1*); 4) A pure hardware implementation produced by LegUp (*LegUp-HW*); 5) A pure hardware implementation produced by eXCite (*eXCite-HW*). The two hybrid flows correspond to a system that includes the MIPS processor and a single accelerator, where the accelerator implements a C function and all of its descendant functions. For the back-end of all flows, we use Quartus II ver. 9.1 SP2 to target the Cyclone II FPGA.

Three metrics are employed to gauge quality of result: 1) circuit speed, 2) area, and 3) energy consumption. For circuit speed, we consider the cycle latency, clock frequency and total execution time. Cycle latency refers to the number of clock cycles required for a complete execution of a benchmark. Clock frequency refers to the reciprocal of the post-routed critical path delay reported by Altera timing analysis. Total execution time is simply the cycle latency multiplied by the clock period. To measure energy, we use Altera's PowerPlay power analyzer tool, applied to the routed design. We use switching activity data gathered from a full delay simulation with Mentor Graphics' ModelSim.

Table 1 presents speed performance results for all circuits and flows. Three data columns are given for each flow: *Cycles*, *Freq* in MHz, and *Time* in μS ($Cycles/Freq$). The second last row of the table contains geometric mean results for each column. The *dhrystone* benchmark was excluded from the geomean calculations, as eXCite was not able to compile this benchmark. The last row of the table presents the ratio of the geomean relative to the software flow (*MIPS-SW*).

For the *MIPS-SW* flow, Table 1 indicates that the processor runs at 74 MHz on the Cyclone II and the benchmarks take between 6.7K-29M cycles to complete their execution. In the *LegUp-Hybrid2* flow, the number of cycles needed for execution is reduced by 50% compared with software, on average. The *Hybrid2* circuits run at 6% lower frequency than the processor, on average. Overall, *LegUp-Hybrid2* provides a 47% (1.9×) speed-up in program execution time vs. software (*MIPS-SW*). In the *LegUp-Hybrid1* flow, cycle latency is 75% lower than software alone. However, clock speed is 9% worse for this flow, which results in a 73% reduction in program execution time vs. software (a 3.7× speed-up over software). Looking broadly at the data for *MIPS-SW*, *LegUp-Hybrid1* and *LegUp-Hybrid2*, we observe a trend: execution time decreases substantially as more computations are mapped to hardware.

Benchmark programs mapped using the *LegUp-HW* flow require 12% of the clock cycles of the software implementations, on average, yet they run at about the same speed in MHz. Benchmarks mapped using *eXCite-HW* require even fewer clock cycles – just 8% of that required for software implementations. However, implementations produced by eXCite run at 45% lower clock frequency than the MIPS processor, on average. LegUp produces heavily pipelined hardware implementations, whereas, we believe eXCite does more operation chaining, resulting in fewer cycles yet longer critical path delays. Considering total execution time of a benchmark, LegUp and eXCite offer similar results. *LegUp-HW* provides an 88% execution time improvement vs. software (8× speed-up); *eXCite-HW* provides an 85% improvement (6.7× speed-up).

It is worth highlighting a few results in Table 1. Comparing *LegUp-HW* with *eXCite-HW* for the benchmark *aes*, LegUp's implementation provides a nearly 5× improvement over eXCite in terms of execution time. Conversely, for the *motion* benchmark, LegUp's implementation requires nearly 4× more cycles than eXCite's implementation. We believe such differences lie in the extent of pipelining used by LegUp vs. eXCite.

Average area results are provided in Table 2. For each flow, three columns provide the number of Cyclone II logic elements (*LEs*), the number of memory bits used (*# bits*), as well as the number of 9x9 multipliers (*Mults*). The numbers in parentheses represent ratios relative to the *MIPS-SW* flow. The hybrid flows include *both* the MIPS processor, as well as custom hardware, and consequently, they consume considerably more area. The *LegUp-HW* flow implementa-

Table 1: Speed performance results.

	MIPS-SW			LegUp-Hybrid2			LegUp-Hybrid1			LegUp-HW			eXCite-HW		
Benchmark	**Cycles**	**Freq.**	**Time**	**Cycles**	**Freq.**	**Time**	**Cycles**	**Freq.**	**Time**	**Cycles**	**Freq.**	**Time**	**Cycles**	**Freq.**	**Time**
adpcm	193607	74.26	2607	159883	61.61	2595	96948	57.19	1695	36795	45.79	804	21992	28.88	761
aes	73777	74.26	993	55014	54.97	1001	26878	49.52	543	14022	60.72	231	55679	50.96	1093
blowfish	954563	74.26	12854	680343	63.21	10763	319931	63.7	5022	209866	65.41	3208	209614	35.86	5845
dfadd	16496	74.26	222	14672	83.14	176	5649	83.65	68	2330	124.05	19	370	24.54	15
dfdiv	71507	74.26	963	15973	83.78	191	4538	65.92	69	2144	74.72	29	2029	43.95	46
dfmul	6796	74.26	92	10784	85.46	126	2471	83.53	30	347	85.62	4	223	49.17	5
dfsin	2993369	74.26	40309	293031	65.66	4463	80678	68.23	1182	67466	62.64	1077	49709	40.06	1241
gsm	39108	74.26	527	29500	61.46	480	18505	61.14	303	6656	58.93	113	5739	41.82	137
jpeg	29802639	74.26	401328	16072954	51.2	313925	15978127	46.65	342511	5861516	47.09	124475	3248488	22.66	143358
mips	43384	74.26	584	6463	84.5	76	6463	84.5	76	6443	90.09	72	4344	76.25	57
motion	36753	74.26	495	34859	73.34	475	17017	83.98	203	8578	91.79	93	2268	42.87	53
sha	1209523	74.26	16288	358405	84.52	4240	265221	81.89	3239	247738	86.93	2850	238009	62.48	3809
dhrystone	28855	74.26	389	25599	82.26	311	25509	83.58	305	10202	85.38	119	-	-	-
Geomean:	**173332.0**	**74.26**	**2334.1**	**86258.3**	**69.98**	**1232.6**	**42700.5**	**67.78**	**630.0**	**20853.8**	**71.56**	**291.7**	**14594.4**	**40.87**	**357.1**
Ratio:	**1**	**1**	**1**	**0.50**	**0.94**	**0.53**	**0.25**	**0.91**	**0.27**	**0.12**	**0.96**	**0.12**	**0.08**	**0.55**	**0.15**

Table 2: Area results (geometric mean).

Flow	#LEs	# bits	Mults
MIPS-SW	12243 (1)	226009 (1)	16 (1)
LegUp-Hybrid2	27248 (2.23)	258526 (1.14)	43 (2.68)
LegUp-Hybrid1	33629 (2.75)	261260 (1.16)	51 (3.18)
LegUp-HW	15646 (1.28)	28822 (0.13)	12 (0.72)
eXCite-HW	13101 (1.07)	496 (0.00)	5 (0.32)

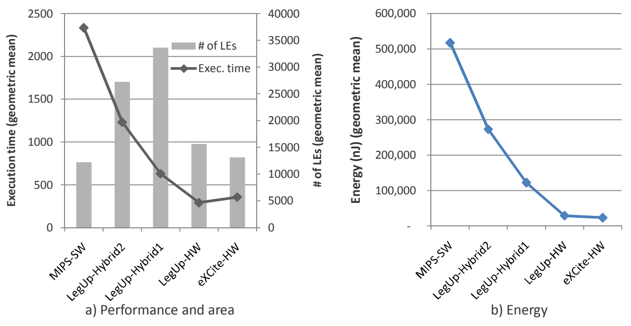

Figure 3: Performance, area and energy results.

tions require 28% more LEs than the MIPS processor on average; the *eXCite-HW* implementations require 7% more LEs than the processor. In terms of memory bits, both the *LegUp-HW* flow and the *eXCite-HW* flow require much fewer memory bits than the MIPS processor alone. For the benchmarks that require embedded multipliers, the *LegUp-HW* implementations use more multipliers than the *eXCite-HW* implementations, which we believe is due to more extensive multiplier sharing in the binding phase of eXCite.

Fig. 3(a) summarizes the speed and area results. The left vertical axis represents geometric mean execution time; the right axis represents area (number of LEs). Observe that execution time drops as more computations are implemented in hardware. While the data shows that pure hardware implementations offer superior speed performance to pure software or hybrid implementations, the plot demonstrates LegUp's usefulness as a tool for exploring the hardware/software co-design space. One can multiply the delay and area values to produce an *area-delay product*. On such a metric, *LegUp-HW* and *eXCite-HW* are nearly identical ($\sim$4.6M μS-LEs vs. $\sim$4.7M μS-LEs) – *LegUp-HW* requires more LEs vs. *eXCite-HW*, however, it offers better speed, producing a roughly equivalent area-delay product.

Fig. 3(b) presents the geometric mean energy results for each flow. Energy is reduced drastically as more computations are implemented in hardware vs. software. The *LegUp-Hybrid2* and *LegUp-Hybrid1* flows use 47% and 76% less energy than the *MIPS-SW* flow, respectively. With *LegUp-HW*, the benchmarks use 94% less energy than if they are implemented with the *MIPS-SW* flow (an 18× reduction). The eXCite energy results are similar to LegUp.

6. CONCLUSIONS

In this paper, we introduced LegUp – a new high-level synthesis tool that compiles a standard C program to a hybrid processor/accelerator architecture. Using LegUp, one can explore the hardware/software design space, where some portions of a program run on a processor, and others as custom hardware circuits. LegUp, along with its suite of benchmark C programs, is a powerful open source platform for HLS research that we expect will enable a variety of research advances in hardware synthesis, as well as in hardware/software co-design. LegUp is available for download at: `http://www.legup.org`.

7. REFERENCES

[1] Altera, Corp. *Nios II C2H Compiler User Guide*, 2009.

[2] Altera, Corp. *Avalon Interface Specification*, 2010.

[3] J. Cong and Y. Zou. FPGA-based hardware acceleration of lithographic aerial image simulation. *ACM Trans. Reconfigurable Technol. Syst.*, 2(3):1–29, 2009.

[4] P. Coussy, D. Gajski, M. Meredith, and A. Takach. An introduction to high-level synthesis. *IEEE Design Test of Computers*, 26(4):8 – 17, jul. 2009.

[5] D. Gajski and et. al. Editors. *High-Level Synthesis - Introduction to Chip and System Design*. Kulwer Academic Publishers, 1992.

[6] Y. Hara, H. Tomiyama, S. Honda, and H. Takada. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Journal of Information Processing*, 17:242–254, 2009.

[7] http://www.llvm.org. *The LLVM Compiler Infrastructure Project*, 2010.

[8] C.Y. Huang, Y.S. Che, Y.L. Lin, and Y.C. Hsu. Data path allocation based on bipartite weighted matching. In *Design Automation Conference*, volume 27, pages 499–504, 1990.

[9] G. Stitt and F. Vahid. Binary synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 12(3), 2007.

[10] United States Bureau of Labor Statistics. *Occupational Outlook Handbook 2010-2011 Edition*, 2010.

[11] Univ. of Cambridge, http://www.cl.cam.ac.uk/teaching/0910/ECAD+Arch/mips.html. *The Tiger "MIPS" processor.*, 2010.

[12] F. Vahid, G. Stitt, and Lysecky R. Warp processing: Dynamic translation of binaries to FPGA circuits. *IEEE Computer*, 41(7):40–46, 2008.

[13] Y Explorations (XYI), San Jose, CA. *eXCite C to RTL Behavioral Synthesis 4.1(a)*, 2010.