Automatic Compilation of C Applications for FPGA-based Hardware Acceleration

Lieu My Chuong[1], Yan Lin Aung[2], Siew-Kei Lam[2], Thambipillai Srikanthan[2], Lim Chai Soon[3]

[1]PixelMetrix Corporation, Singapore
[2]Centre for High Performance Embedded Systems,
Nanyang Technological University, Singapore
[3]School of Engineering, Republic Polytechnic, Singapore
jeff@pixelmetrix.com, {layan, assklam, astsrikan}@ntu.edu.sg, lim_chai_soon@rp.sg

Abstract

Advancement in design tools is necessary to bridge the widening productivity gap between hardware design and software development in state-of-the-art Field Programmable Gate Arrays (FPGA). We present a design exploration framework that automatically compiles C applications to realize efficient custom co-processor structures for hardware acceleration on the reconfigurable logic. We show that the proposed design exploration framework can automatically generate Register Transfer Level (RTL) codes from C-functions that outperform the commercial Altera C2H RTL generator by about 40% in terms of average area-time product.

1. Introduction

FPGAs are increasingly being adopted in embedded systems due to their ability to meet the technological and market uncertainties [1]. Modern FPGA architectures incorporate a multitude of Intellectual Property cores that include soft and hard processors. The additional processing options available on FPGAs have increased the complexity of design space explorations and have become unmanageable in traditional design methodologies especially for large applications.

In light of this, system-level design methodologies are expected to play a central role in the design success of current and future embedded products. System-level design methodologies often incorporate High-Level Synthesis (HLS) that allow automatic compilation of algorithm description in high-level languages such as C, C++ and SystemC to RTL code. In addition, one of the key tasks in modern system-level design is design space exploration for evaluating possible candidate design instances with varying design trade-offs on performance, hardware area, power, etc. in order to determine an optimal solution.

In this paper, we present a design exploration framework that can rapidly generate custom co-processor structures for FPGAs from C functions. The proposed framework is part of a larger co-design methodology that is capable of generating optimal hardware and software implementations for modern FPGA platforms.

The proposed framework relies on the Trimaran compiler infrastructure [2] for its advanced scheduling schemes to expose inherent parallelism in C-based algorithms. We have employed a co-processor template to assist in data and control path generation. The original Very Long Instruction Word (VLIW) machine model in Trimaran was modified to incorporate heterogeneous Functional Units (FUs) that will be bounded to the co-processor template.

The remainder of the paper is organized as follows. In the next section, we will discuss commercial tools and related work in automatic hardware generation for FPGAs, as well as the main contribution of our work. In Section 3, we describe an overview of the proposed framework and co-processor template. Section 4 discussed the bit-width optimization technique that has been employed for high-level synthesis in the proposed framework. Section 5 presents experimental results demonstrating the benefits of the proposed framework. Finally, Section 6 concludes the paper.

2. Related Work

A number of commercial tools that synthesize high-level languages to FPGA have been developed to expedite the development of complex applications in hardware. These tools differ in high-level language support, optimization capabilities and the target device. For example, Mitrion-C from [3] and ImpulseC [4] support a subset of the ANSI-C language that is extended with constructs for specifying the hardware definitions. These tools cannot be directly employed for most embedded applications that are typically represented using ANSI-C. C2H tool from [5], Catapult from [6] and Trident [7] support pure ANSI-C applications. Although these tools can efficiently convert C-level algorithm into gates, they do not provide accurate high-level estimations to facilitate hardware-software partitioning in FPGA-based system.

HLS approaches vary widely from hardware only implementations of high-level applications to processor-accelerator systems to heterogeneous multi-core systems. Some HLS tools have been developed for domain specific applications such as GAUT [8], ROCCC [9]. Quality of results and usability are two key criteria to measure capability and effectiveness of HLS tools. Recent BDTI evaluation results on two commercial HLS tools: AutoPilot from AutoESL and Synphony C from Synopsys show that current state-of-the-art HLS tools are capable of achieving both criteria. Such advancement plays a pivotal role in design space exploration for FPGA-based embedded systems.

2.1. Main Contribution

Unlike existing commercially available C-to-FPGA tools from Mitronics and Impulse Accelerated Technologies, the proposed framework supports applications represented in pure ANSI-C language, which is widely used in embedded applications. In addition, unlike C-to-FPGA tools from Altera, Mentor Graphics and Trident, the proposed framework incorporates reliable high estimation techniques to facilitate rapid design exploration for hardware-software partitioning for FPGA systems. Existing works also often neglect the effects of logic synthesis when translating the high-level input specification to an intermediate form for estimation. The proposed framework overcomes this problem by incorporating efficient bit-width optimization strategies. We will show that RTL implementations generated from the proposed framework provide higher area-time gains than those generated using the commercial Altera C2H RTL generator for almost all the benchmarks considered.

3. Overview of Proposed Framework

Figure 1 describes an overview of the proposed design exploration framework. The open-source Trimaran compiler infrastructure, which supports state of the art compiler research in instruction level parallelism based architectures, is relied upon to expose the hidden parallelism in the sequential C functions, and to perform high-level optimizations and scheduling. This front-end process typically takes less than 10 seconds for compiling a single C-function.

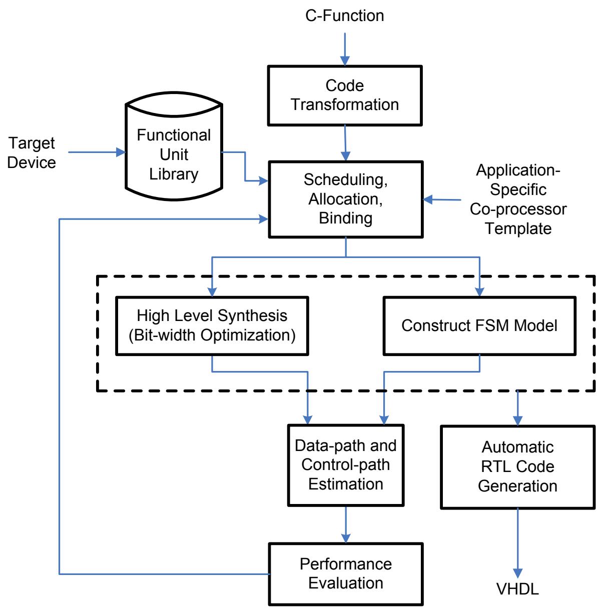
C-Function
Code Transformation
Functional Unit Library
Target Device
Scheduling, Allocation, Binding
Application-Specific Co-processor Template
High Level Synthesis (Bit-width Optimization)
Construct FSM Model
Data-path and Control-path Estimation
Automatic RTL Code Generation
Performance Evaluation
VHDL

Figure 1: Proposed design exploration framework

Based on the available Function Units (FUs) specified in the Functional Unit Library, Trimaran outputs an application schedule (i.e. the type of FUs that will be executed in each clock cycle and the data-dependency between them). This schedule information and the co-processor template are used in a simple hardware binding process to bind FUs with the most common input-outputs in order to reduce the interconnect complexity between registers and FUs. Bit-width optimization and constant propagation are then performed to prune off unnecessary logic.

Although not the focus of this paper, data-path and control-path estimation is then performed to evaluate the performance of the FPGA implementation. The process is repeated for different sets of hardware operators in order to populate the exploration space. The framework also incorporates a process to automatically generate RTL codes of the controller and data-paths. The final RTL code can subsequently be used as inputs to the FPGA implementation tool. Details of data-path and control-path estimation can be found in [10][11].

3.1. Co-processor Template

Figure 2 shows the application-specific co-processor template, which is used to facilitate automatic generation of the controller and data-paths. The controller is a Finite State Machine (FSM) comprising of the following components: 1) Next State Decoding Logic, which computes the next-state of the FSM based on the current state and inputs, 2) Control Signal Decoding Logic, which decodes the control signals to the data-path (i.e. register enable signals, multiplexer select signals, FU enable signals), and 3) state registers, that hold the current state of the FSM. Details of control-path generation can be found in [11].

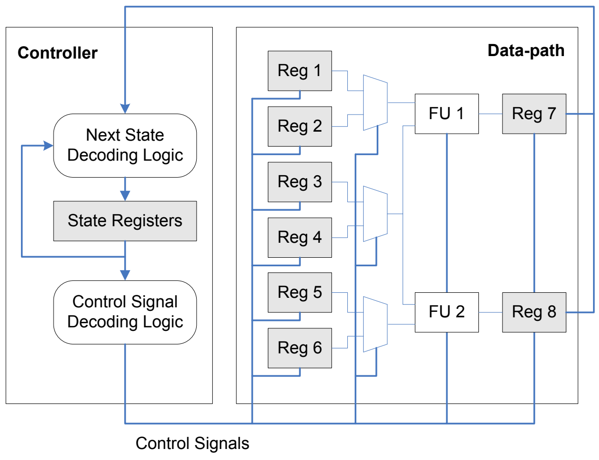

Figure 2: Co-processor template

We have adopted the data-path model that is similar to the one proposed in [12]. It is worth mentioning that the co-processor template can be adapted for pipelined or non-pipelined data-paths by configuring the application-specific interconnection. The Trimaran machine description is augmented with a range of heterogeneous FUs that can be part of the co-processor template. Each FU performs a dedicated operation (e.g. addition, shift, multiply, multiply-accumulate, logic operation, comparison, multiplexers, memory access operations, etc.). Only FUs that are required for a particular application will be incorporated in the application-specific co-processor.

4. Bit-Width Optimization

Bit-width optimization aims to automatically derive the minimum bit-width of FUs, while maintaining the functional correctness of the high-level specification. We have adopted the work in [13] to propagate the bit-width of each variable through the FUs. The work in [13] performs iterative constraint propagation to repeatedly refine the bit-width of the variables.

In addition to this, we have devised a simple algorithm to identify unused and constant bits that could lead to further bit-width refinement of the FUs. The algorithm repeatedly propagates the unused and constant bits through the data-path to refine the bit-width of the FUs based on the rules shown in Figure 3. These rules are used to determine the resulting bit *i* of a particular operator with bit operands m_i and n_i. Note that the operations in Figure 3, with the exception of the shifter, are commutative. The algorithm terminates when the bit-widths of the FUs remain unchanged in a particular iteration.

	m_i, n_i					
Operator	**0, 0**	**0, 1**	**1, 1**	**0, U**	**1, U**	**U, U**
Logical AND	0	0	1	0	U	U
Logical OR	0	1	1	U	1	U
2-1 Multiplexer	0	U	1	U	U	U
Adder/Subtractor *	0	1	0	U	U	U
Multiplier **	0	0	U	0	U	U
n-bit Left Shifter	Least significant n bits = 0					

* only applies when m_x AND y_x are 0s, where $x < i$
** only applies when m_x or y_x are 0s, where $x < i$

Figure 3: Bit-width inference rules

We will describe a single iteration of the algorithm using the data-path example in Figure 4. In the example, the constant bits (i.e. ‘0’s and ‘1’s) are propagated from the source registers (i.e. Reg 1 and Reg 2), through the multiplexer M1, arithmetic operators (i.e. multiplier and left shifter by 2), multiplexer M2 and finally to the destination register Reg 3. The output values of each register and operator is shown, where ‘U’ represents a bit value that is unknown before application runtime. Let us assume that the last two bits of the source registers are constants.

Based on the rules in Figure 3, it can be inferred that the LSB output of M1 is ‘0’ as both inputs of the multiplexer has a LSB of ‘0’. This constant ‘0’ is then propagated to the output of the multiplier. At the same time, the last two bits of the 2-bit left-shifter’s output are always ‘0’s. The ‘0’s at shifter’s output and multiplier’s output are then propagated to Reg 3, where the LSB is inferred to be a constant ‘0’. The knowledge of the unused and constant bits enables us to determine the necessary bit-widths for the FUs and registers in the data-path.

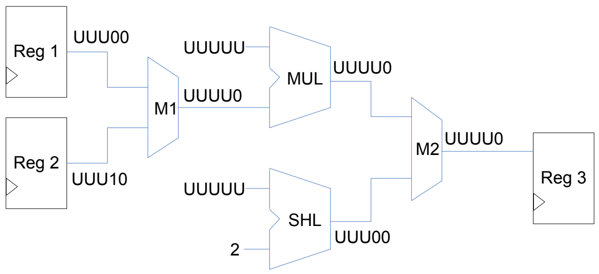

Figure 4: Refining bit-width of operators

5. Experimental Results

In this section, we compare the implementation results of RTL codes generated using the proposed design exploration framework with those that are generated from the commercial Altera C2H RTL generator. Thirteen embedded functions from EEMBC [14] and Trimaran benchmark suite, as shown in Table 1, were used to evaluate the effectiveness of the proposed framework.

Table 1. C-Functions considered in the experiments

	FUNCTIONS		FUNCTIONS
1	AutoCorrelation	**8**	CjpegV2_FixpointDCT_17mul
2	Adpcm_Coder	**9**	CjpegV2-RGB_YCC_Convert
3	Adpcm_Decoder	**10**	Convolutional Encoding
4	Comb_Sort	**11**	Viterbi_ACS
5	IDCT_Col	**12**	Viterbi_FindMetics
6	IDCT_Row	**13**	Viterbi_All
7	Sha		

These thirteen functions were first compiled using Trimaran to produce the IR. The framework shown in Figure 1, which incorporates the techniques discussed in this paper, is used to compile the C-functions to RTL, which were then implemented using Altera Quartus II 7.2. Note that in these experiments, we do not employ hardware estimation and performance evaluation as we are only interested to evaluate the quality of the RTL codes produced by the proposed framework.

We have evaluated the proposed method on Altera Cyclone II (EP2C8) and compared our results with the RTL generated using Altera C2H software. Altera C2H software is an EDA tool that is used for automatically translating algorithms written in ANSI-C into RTL codes so that they can be implemented as a hardware accelerator using Altera Quartus II FPGA tool. The advantage of this tool is that the hardware accelerator can be generated with minimum modifications in the C codes. This section compares the RTL design generated by the proposed design exploration framework and RTL generated by Altera C2H. Figure 5 and Figure 6 shows the area and critical path delay comparison of the RTL codes generated by the proposed design exploration framework and Altera C2H tool.

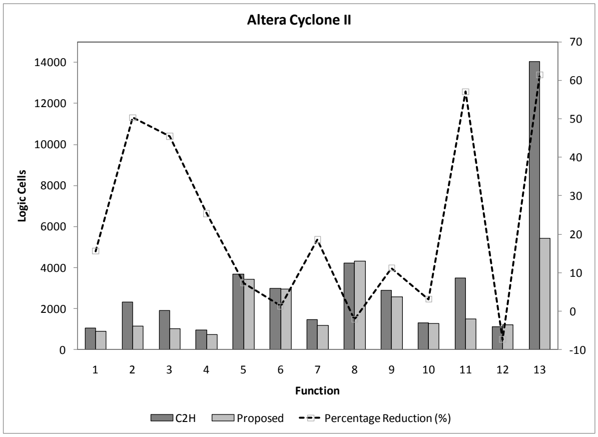

Figure 5: Area comparison

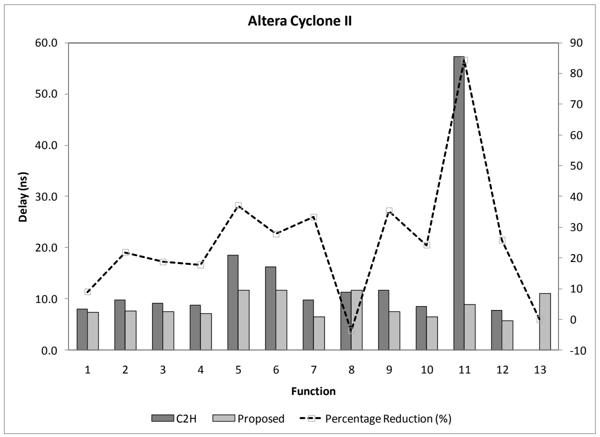

Figure 6: Delay comparison

It can be observed that the RTL codes generated from the proposed design exploration tool outperforms the RTL codes of Altera C2H tool in both area-utilization and performance for almost all the functions considered. In particular, when compared to the RTL codes of Altera C2H tool, the RTL codes generated by the proposed framework has an average area reduction of 22.1% and 27.6% for the area and critical path delay respectively. In addition, the Altera C2H tool is unable to produce a solution that can be fitted onto the Altera device for large functions e.g. Viterbi_All. The proposed framework uses Trimaran's in-lining optimization to flatten the Viterbi_All function in order to perform more optimizations on the function.

Figure 7 shows area-time product of the RTL codes generated by the proposed framework and C2H's RTL, which is computed by multiplying the circuit's area (logic cells) with the circuit's minimum clock period (ns). It can be observed that the area-time product of the hardware generated by the proposed framework is smaller for almost all cases. In particular, when compared to the RTL codes of Altera C2H tool, the RTL codes generated by the proposed design exploration framework has an average area-time product reduction of 39.3% and a maximum area-time product reduction of over 93% (for the Viterbi_ACS function). Finally, the runtime time of the proposed design exploration to generate RTL codes is much faster than Altera C2H (less than a second compared to tens of minutes).

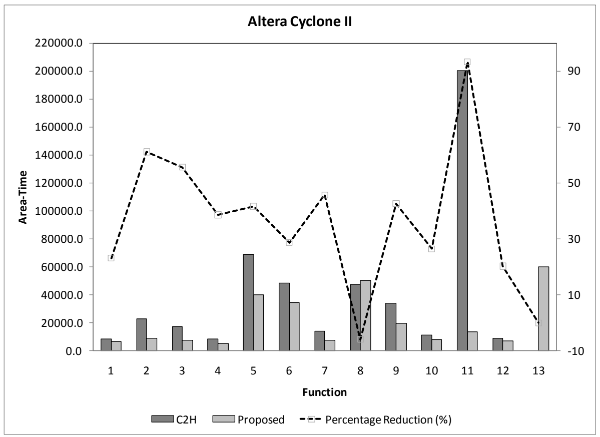

Figure 7: Area-time product comparison

6. Conclusion

In this paper, we proposed a design exploration framework to facilitate FPGA porting of algorithms represented in C. The proposed framework employs bit-width optimization to automatically derive the minimum bit-width of FUs. A simple algorithm is used to identify unused and constant bits that could lead to further bit-width refinement of the FUs. Results show that our RTL implementations are superior to those generated using the commercial Altera C2H RTL generator in almost all cases considered.

7. References

[1] P. Garcia, K. Compton, M. Schulte, E. Blem and W. Fu, "An Overview of Reconfigurable Hardware in Embedded Systems", EURASIP Journal on Embedded Systems, Vol. 1, 2006, pp. 1-19.

[2] Trimaran, "An Infrastructure for Research in Backend Compilation and Architecture Exploration", Available: http://www.trimaran.org/.

[3] Mitrionics, "The Mitrion Accelerated Computing Platform", Available: http://www.mitrionics.com/?page=mitrion-software-development-platform.

[4] Impulse Accelerated Technologies, Available: http://www.impulseaccelerated.com/index.htm.

[5] Altera, "Design Software", Available: http://www.altera.com/products/software/sfw-index.jsp.

[6] Mentor Graphics, "Catapult C Synthesis". Available: http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/.

[7] Trident Compiler, Available: http://trident.sourceforge.net/.

[8] P. Coussy and A. Morawiec, "GAUT: A High-Level Synthesis Tool for DSP Applications", High-level Synthesis: From Algorithm to Digital Circuit, Springer, 2008, pp. 147-170.

[9] J. Villarreal, A. Park, W. Najjar, and R. Halstead, "Designing Modular Hardware Accelerators in C with ROCCC 2.0", 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2010, pp. 127-134.

[10] M.C. Lieu, S.K. Lam, and T. Srikanthan, "High-level Delay Estimation Technique for Porting C-based Applications on FPGA", IEEE International Symposium on Industrial Electronics, June/July 2008, pp. 1991-1996.

[11] M.C. Lieu, S.K. Lam, and T. Srikanthan, "Area-Time Estimation of Controller for Porting C-based Functions onto FPGA", IEEE/IFIP International Symposium on Rapid System Prototyping, June 2009, pp. 145-151.

[12] R. Schreiber, S. Aditya, B.R. Rau, V. Kathail, S. Mahlke, S. Abraham and G. Snider, "High-Level Synthesis of Nonprogrammable Hardware Accelerators", 12th IEEE International Conference on Application-Specific Systems, Architectures and Processors, July 2000, 113-124.

[13] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber and T. Sherwood, "Bitwidth Cognizant Architecture Synthesis of Custom Hardware Accelerators", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, No. 11, 2001, pp. 1355-1371.

[14] EEMBC: The Embedded Microprocessor Benchmark Consortium. Available: http://eembc.org.