A Method To Derive Application-Specific Embedded Processing Cores

Olivier Hébert[1], Ivan C. Kraljic[2], Yvon Savaria[1,2]
[1] Electrical and Computer Engineering Dept. École Polytechnique de Montréal, Montréal, Qc, Canada
[2] MiroTech MicroSystems Inc. Saint-Laurent, Qc, Canada
{ hebert, kraljic, savaria }@grm94.polymtl.ca

ABSTRACT

The concept of system-on-a-chip is becoming increasingly popular for the integration of complex systems. New types of processor cores are now available that enable the designer to customize their processors for the target applications. These soft cores are not tightly coupled with the target application, and this leads to processing cores sub-optimal for their specific applications. This paper proposes a method to derive application-specific embedded processors from soft processor cores. The derivation process involves an analysis of the resources of the processing core used by the target application. Then a series of optimizations based on the analysis results are performed on an optimizable model of the processor core. We present the tool used to perform the analysis of the resources used by an application, and results from a real-world case. Then, various optimization methods are described.

Keywords

Embedded core, custom core, configurable processor, soft core, system-on-a-chip.

1. Introduction

Modern embedded systems typically include a microprocessor or a digital signal processor (DSP). With the recent availability of multi-million gate silicon, both in the form of application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs), many designs are implemented as a system-on-a-chip (SOC). The processors, which are often used to execute a fixed small size program repeatedly, are usually implemented from IP (intellectual property) cores, provided by various vendors.

Although logic gates are nowadays considered plentiful, the requirements of embedded systems regarding integration, performance, power and especially production cost are still often a challenge to meet. This is even more difficult in the case of SOC using traditional design techniques with hard IP cores, as their use wastes some of the available resources. Indeed, most applications do not make use of all the components of a hard processor core. This happens because the synthesis of the processor core is not coupled to the target application software that will be executed on it.

New types of cores, also known as soft cores, have started to appear on the market over the last two years. These new cores try to overcome some problems encountered with hard cores by letting designers customize the processors for their specific applications. Still, the coupling between the software application and the core itself is too loose to avoid wasting resources in the physical implementation.

To improve the efficiency of soft processor cores, an extra step of core optimization could be introduced prior to synthesis. This optimization would take into account the application software that will run on the processor to be synthesized, thus leading to application-specific derivation of embedded cores (ASDEC).

This paper presents the resource wasting problem associated with the direct implementation of a core for SOC, and a set of experiments on application-specific derivation of cores on which this kind of optimization may be carried out.

2. Motivations

- Design time is the bottleneck. Designing from scratch, or even from pre-designed library modules takes more time than deriving (automatically) an application-specific core from a general-purpose processor core.
- Validation is costly. It can take up much of the time spent on the design. Derivation from a validated general-purpose processor yields cores that are correct by construction and thus are easier to validate.
- Application-specific cores are simpler than generic processing cores. This is because unused functionality is removed from the generic core during the derivation process. Simpler circuits will lead to better performance, and less gate usage.

Previous research [9] has demonstrated the validity of the derivation approach to core design in the domain of hardwired data-flow vision automata. In [9], a vision application was firstly implemented and validated in real-time on a 1024-processor vision emulator, then an RTL VHDL description of a minimal architecture implementing the application was derived from the application's processor graph.

This research improves upon [9] by deriving cores directly from a validated VHDL description of the initial general-purpose processor. Also, no specific architecture is targeted, although SIMD-type cores will be targeted initially. Hence, a far larger domain of applications than data-flow image processing can be attacked. Finally, no emulator - costly and prone to obsolescence - is required, only the traditional design simulation tools. The additional tools required are fairly straightforward, and are not difficult to develop.

3. Traditional Embedded System Design

Traditional design of embedded systems using IP cores for SOC involves two distinct and almost independent phases: the design of the hardware and the development of the firmware (or software). Usually, the hardware part involves only the choice of the processor core that will have the necessary instruction set, processing power and peripherals for the target application. The processor model (also known as a hard core) is generally described with a synthesizable subset of a hardware description language (HDL). The software development is done afterwards, according to the choices made during the hardware phase. Thus the firmware can take advantage of the particularities of the selected processor core.

Once the design of the hardware and software are done, and both components have been thoroughly simulated and validated, the system is fully synthesized and integrated. This is usually done with synthesis and place & route tools for the hardware part, and a compiler and assembler (and possibly a linker) for the firmware. Both processes are done separately, as shown in Figure 1.

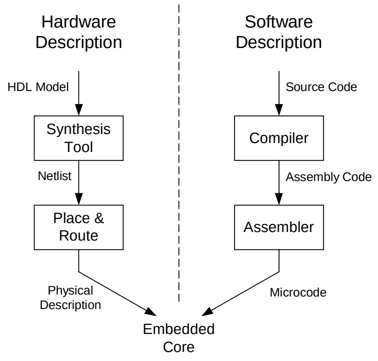

Figure 1 : Traditional synthesis of an embedded core for system-on-a-chip

This division of the design into software and hardware parts leads to the synthesis of a processor core that is not optimized for the target application. Indeed, the synthesis tool does not exploit any information regarding the microcode that will be executed on the processor core, and it cannot prune parts of the circuit that are not used for a particular application.

Modern synthesis tools are quite efficient at removing unused logic or simplifying circuits when given constant values as input. However, these tools cannot optimize circuits for which they do not know in advance the values that will be fed as input. By analyzing the input to the core, we can determine which signals can lead to optimizations of the circuits, and if the optimizations are worthwhile.

4. Microcode analysis tool

To determine whether the optimizations provided by tightly coupling the synthesis of the core with the application-specific software are possible, an analysis tool was needed. This tool, once completed, could also be used in the front-end of a core-optimizer toolset. The tool takes advantage of the predictability of the input signals to perform its analysis. It exploits knowledge of the microcode that will be executed on the processor, because firmware development is done prior to synthesis of the processor core, with a full processor core as the target. Thus, when the firmware is done, it can be analyzed to determine which components of the processor are used. The processor can then be optimized for that specific firmware, prior to synthesis.

4.1 Instruction Decoding

To identify the resources not used by a specific application, a microcode analysis tool has been created. The tool was developed for a specific processor architecture [1], but it applies to any similar architecture which fits the general model shown in Figure 2. This tool decodes the instruction microcode the same way it is done inside the processor. Thus, the various instructions are decoded into a list of bit fields according to their instruction encoding pattern. Then, the field values are fed to another set of decoders, which emulate the multiple ROMs that are embedded in the processor's controller. Most of the signals that will be fed to the other sections of the processors come from the output of these ROM emulator decoders. Note that the proposed optimization would be equally efficient if the decoding is implemented with hardwired logic instead of ROMs even though the discussion assumes a ROM based implementation.

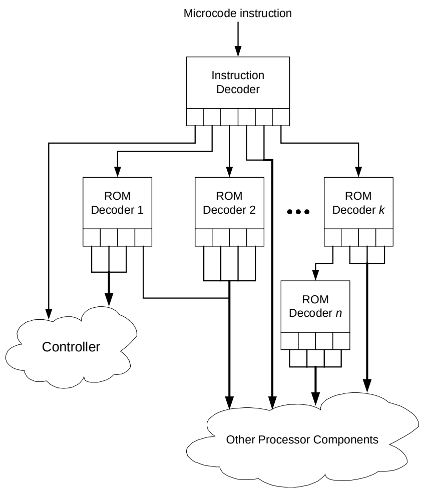

Figure 2 : Microcode decoder

4.2 Usage statistics

The analyzer has been designed to be as generic and architecture-independent as possible. All the information regarding the structure of the instruction decoders, the decoding ROMs, and the different fields are defined in tables contained in text files. As for the ROM tables themselves, they are extracted directly from the synthesizable VHDL source code to ensure proper decoding. Therefore, porting the tool from one processor architecture to another requires only rewriting of the different description tables, and possibly some modifications to small parts of the source code.

The second function of the analyzer is to accumulate the various values taken by the decoded signals for all the instructions of a particular program. When an entire program has been decoded and processed, the analyzer can determine which values of every signal are actually used. Thus, for each decoded field (or signal), the analyzer outputs the number of values actually used, the values themselves and the number of occurrences for each. It also calculates the minimum number of bits required to code a particular field, as determined by the number of values used.

The tool also enables the analysis of groups of signals. Thus, statistics regarding all the signals controlling a particular component, such as an ALU, can be generated. It quickly gives the designer a good idea of which resources are used in his application. These statistics can later be used to optimize the structure of the processor core, as explained in section 6.

5. Case Analysis : The PULSE V1 Processor

To perform actual measurements of a real-world processor, the microcode analyzer was customized for the PULSE V1 processor [1]. PULSE (Parallel Ultra Large Scale Engine) is a SIMD processor optimized for real-time digital signal processing applications, in which computations and data I/O are performed in parallel. This particular processor was selected because the synthesizable VHDL model had been verified and used successfully to build a prototype chip that was available. The PULSE V1 prototype was a fairly large design, with about 250k equivalent gates.

5.1 Test Applications Suite

A series of applications were written during the development of the PULSE V1 processor. A set of these typical programs were selected to perform usage measurements of different parts of the processor. The applications are as follows :

- CONV3X3 : A 3×3 two-dimensional generic convolution
- CONVMED : A 3×3 two-dimensional generic convolution followed by median filtering
- EDGE : An edge-detection image filter
- HISTO : Histogram image analysis
- IDEA : The International Data Encryption Algorithm (IDEA), used in PGP [2]
- MEDIAN3X3 : 3×3 median image filter
- RSA : RSA encryption and decryption
- SHRINK : Mathematical morphology : binary shrinking of an image

5.2 Test Applications Suite Results

Two sets of signals are measured in our test applications : the microcode and the control signals for various modules in the processing elements. For the microcode, the results shown in Table I indicate the minimum bit width of the microcode necessary to include all the information to execute the application, but using all the optimisations described in section 6, where addresses and indexes are recoded to minimize microcode word size.

For the other control signals, there are no addresses or indexes in the lists. These signals control the behavior of the various modules. The number indicated for each element is the total number of control signals that change during the execution of the application. A signal that does not change indicates that a feature or an operating mode of a module is not used.

First of all, we can see that the microcode can be recoded to a much smaller width when the proposed optimisations are applied. This leads to an overall microcode width of about one third of the original size.For the control signals of the various processing element modules, we can see that not all applications use all components, as many signals (if not all) remain constant. For example, the accumulator is only used in three out of nine applications. And for the modules that are actually used, it is rare that all the available operating modes and features are used. We can determine that for all the test applications, there are many parts of the processing element components that are not used. This indicates that these parts could be pruned without modifying the behavior of the processor for an application running on it. In the case of the PULSE V1 processor, this can lead to some interesting optimisations, as there are four processing elements (PEs) in the architecture. Thus, the removal of a component in the datapath is actually done four times.

5.3 Initial Manual Optimizations

A port of the PULSE V1 architecture towards a Xilinx Virtex FPGA [3] was done. The controller used up 3934 slices and the datapath 6772 slices. Manual optimizations were tried to reduce the complexity of the processor architecture when used only as an 8-bit MAC (multiply-accumulate) machine to do DSP processing such as convolutions and FIR filtering. Simple optimizations were tried, such as fitting the datapath width to the data to be processed, removing unused operating components in the datapath and reducing the ROM depths. Although these optimizations were rather simple, the resulting controller was smaller (1330 slices) and the datapath was much smaller (1066 slices). These results led the authors to believe that through automatic optimizations, we could get much better results with far more complex optimizations that could be difficult to do

Application	Microcode (66)	Accumulator Control (5)	ALU Control (27)	Barrel Shifter Control (8)	IO Channel Control (14)	Multiplier-Adder Control (6)	Memory Control (17)	Register Files Control (11)
CONV3X3	17 (26 %)	2 (40 %)	7 (26 %)	3 (38 %)	6 (43 %)	2 (40 %)	2 (12 %)	3 (28 %)
CONVMED	24 (37 %)	2 (40 %)	20 (75 %)	7 (12 %)	6 (43 %)	4 (67 %)	11 (65 %)	11 (100 %)
EDGE	16 (25 %)	0 (0 %)	9 (33 %)	0 (0 %)	4 (29 %)	0 (0 %)	2 (12 %)	10 (91 %)
HISTO	14 (22 %)	0 (0 %)	7 (26 %)	0 (0 %)	3 (22 %)	0 (0 %)	5 (30 %)	0 (0 %)
IDEA	20 (31 %)	0 (0 %)	15 (66 %)	6 (75 %)	6 (43 %)	4 (67 %)	14 (83 %)	10 (91 %)
MEDIAN3X3	22 (33 %)	2 (40 %)	19 (71 %)	5 (63 %)	6 (43 %)	4 (67 %)	11 (65 %)	11 (100 %)
RSA	20 (31 %)	0 (0 %)	12 (45 %)	5 (63 %)	4 (29 %)	5 (84 %)	10 (59 %)	10 (91 %)
SHRINK	22 (33 %)	0 (0 %)	6 (23 %)	0 (0 %)	4 (29 %)	0 (0 %)	14 (83 %)	0 (0 %)

Table I : Control Signals usage for application suite

manually.

6. Core Optimizations

With SOC used for integrating and optimizing performance and power requirements of embedded systems, and the availability of multi-million gate silicon (both in ASIC and FPGA form), new options have emerged for the embedded system designers. Among the new offerings is hardware configurable via software implementation of peripherals. This architecture uses the processing power now available in current processors to emulate functions traditionally done in hardware. Also available are configurable processors, which are a single-chip combination of a traditional microprocessor core with programmable logic and memory [4].

Another important new type of cores is now available: configurable processor cores, also known as soft cores [5][6][7]. These cores enable the designers to customize their processor by selecting the instruction set, the cache and register file sizes. They also enable the addition of new instructions, special registers, local RAM, conditional codes and multimedia extensions. Although these new cores and toolsets offer the designer greater flexibility, they still do not enable an optimization of the synthesized core according to the target application. By using the statistics regarding a specific application, such as those generated by the tool described in section 4, we can perform optimizations to the HDL model and the microcode prior to final synthesis, thus leading to application-specific synthesis of an embedded core (ASDEC). Figure 1 illustrates this new step in the synthesis flow.

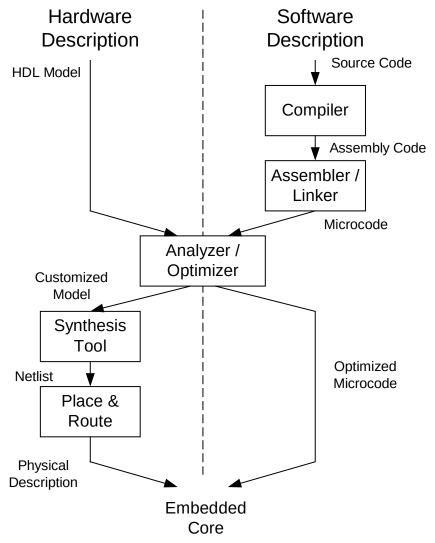

Figure 3 : Application-specific derivation of an embedded core (ASDEC) targeting a SOC

This approach to system synthesis diverges from the traditional cosynthesis methods. Usually, a system is described using a behavioral description, then a circuit is synthesized using elements from a library of predesigned components, and software is compiled to run on the generated hardware [8]. In our approach, the application is described by writing software for a complete processor core. Then, during optimization prior to synthesis, the subset of the entire core needed to run the application software is extracted and synthesized, thus leading to application-specific synthesis. A simple form of application-specific synthesis has been proposed in the context of fault-tolerant processors [9]. This approach was used to generate simpler processors which are more reliable.

The statistics described earlier can lead to a number of easy optimizations, some of which are described here, and will be included in a customizable version of the PULSE core which is in development.

6.1 Resource Elimination

When it can be determined that a particular component is not used in a specific application, it can be removed from the processor description prior to synthesis. For example, if the write enable signal of a MAC unit to the destination bus of the ALU is always off, then the MAC unit can be eliminated, thus saving resources. There are two ways to do this: the first one is described in the next section and basically involves letting the synthesis tool eliminate the unused parts. The second strategy is used when the synthesis tool cannot automatically do the optimization. This can occur under various conditions, one of which being a tri-state bus driver always in a high-impedance state: some tools cannot remove the logic that feeds the input of the bus driver. The optimizer must then remove these sections, before sending the model to the synthesis tool. This can be easily accomplished using inclusion clauses in the HDL model (for example, using *if […] generate* in VHDL) which can be set by the optimizer. This requires a properly coded HDL model which enables the removal of sections of the circuits without any impact to the rest of the circuit's functionality.

6.2 Constant Signals Propagation

As specified earlier, modern synthesis tools are quite efficient at removing unused logic or simplifying circuits when given constant values as input. So, when signals are found to remain constant for an application during the analysis phase, constant values can be fed to the input of the appropriate modules. The synthesis tool will then proceed to replace all the unused modules with constant values at the output. This is an automatic form of resource elimination, as described in the previous section, with the exception that this one does not require a model coded especially with the optimization in mind, because the synthesis tool is handling all the work when signals that do not change have been suitably detected and specified.

6.3 Local Constant Tables

Constant values are often used in the microcode, to enable immediate addressing modes and compare operations. The width of constants is usually large (i.e. 16 or 32 bits), while the number of values actually used is small. This implies that wide busses have to be routed from the source of the constants to the appropriate modules that use them, even if a small number of values will actually be used on those busses. While not a problem for most cores, it can become a serious routing hazard in the case of an SIMD processor, where these constants have to be routed to a large number of processing elements. The problem can be even more serious when programmable logic devices, such as FPGAs, with limited routing resources, are used as the physical support for the core. To alleviate this problem, constant values can be recoded into indexes of a constant table. In that case, the source of the constant sends only the index (which can be only a few bits wide) to the destination module. At the destination module, the index is used to decode the actual value from a constants table, and processing is done with the value as before. Although the implementation of the local tables requires more logic resources, the reduced requirements for routing resources can make this optimization very interesting. Of course, the microcode must be adapted to support this modification in the core.

6.4 Field Recoding

An other optimization involves recoding bit fields that are not fully used. For example, if a processor has an instruction word of 8 bits, and a specific application only uses 30 different instructions, the instruction codes could be recoded to a word size of only 5 bits. This optimization can be applied to a large number of signal types, such as instruction words, addressing modes codes and ALU control words. It is easy to do if the model is coded using custom hierarchical types, and by using constants instead of hardcoded values throughout the code. Thus, modifying a specific field only involves modifying the field's width and the constant table associated with it. As before, the microcode must be adapted in conjunction with the core.

6.5 Other Optimizations

The removal of a pipeline level might be possible if there are no actual operations involved in the level (i.e. only routing). This might happen when components are removed from the datapath, thus leaving the level useless. Retiming could also be performed.

A final optimization would be the fitting of the datapath bit-width with the actual data. Thus, we could easily adapt the core for 8-bit, 12-bit or even binary inputs. Having a datapath fitted to the width of the data we are processing would greatly reduce the amount of logic used to implement the datapath.

7. Current Developments

A team is currently working on the creation of a new SIMD architecture that will implement application-specific synthesis, using the optimisations described earlier. This architecture is based on the PULSE V1 core VHDL model and the PULSE V2 specifications [1]. The analysis tool is completed and will be used as part of the front-end for the microcode and processor model optimizer. New tools are also planned to enable further customization of the processor by modifying the instruction set.

8. Conclusion

Systems-on-a-chip are becoming increasingly popular for the design of embedded systems. The design of the hardware part of those systems is usually done using IP cores to minimize the development cycle. Some cores that are now available enable the designer to customize the processor for his application. However, the synthesis of those cores usually produce a processor which is not fully coupled with the target application.

This leads to a sub-optimal use of the available resources, as not all the features and parts of a processor are used in a specific application, but will be synthesized anyway. We have shown that using a microcode analyzer, we can determine which parts of the processor will be used for a specific program. We used the analyzer to get results for typical applications running on a specific processor (PULSE V1), and have shown that many components are not fully (if at all) used with several applications.

Thus, to help meet the requirements of embedded systems with respect to integration, power, performance, and production cost, we introduced an extra optimization step prior to synthesis. This new step leads to application-specific synthesis of processor cores. The synthesized processor is now tightly coupled and optimized for the application that will be executed on it. To execute this step, we presented a few possible optimisations that could be applied to soft processor cores.

We are working on a new SIMD architecture, based on an existing functionnal design (PULSE V1), that will enable the implementation of all the optimizations described in this paper. We are also working on the toolset that will perform the optimizations on the microcode and the processor model. In our future work, we will complete the design and implementation of the application-specific derivation concepts applied to that specific architecture.

9. Acknowledgements

This work was supported by the National Sciences and Engineering Research Council of Canada, grant number # STP 0201872.

10. References

[1] P.Marriott, I.C. Kraljic, Y. Savaria. "Parallel Ultra Large Scale Engine SIMD Architecture For Real-Time Digital Signal Processing Applications." In *International Conference on Computer Design ICCD'98,* Austin, October 1998, pp. 482-487.

[2] B. Schneier. *Applied Cryptography*. New York : Wiley, 1995.

[3] *1999 Data Book*, Xilinx Inc., San Jose, CA, 1999.

[4] *Configurable processor: An emerging solution for embedded system design*. Triscend Corporation, 301 North Whisman Road, Mountain View, CA 94043, 1998. http://www.triscend.com/products/IndexWhitePaper.html

[5] M. Levy. "Customized processors : have it your way". *EDN*, January 7, 1999, pp. 97-104.

[6] D. Bursky. "Tool Suite Enables Designers To Craft Customized Embedded Processors". *Electronic Design*, volume 47, number 3, February 8, 1999.

[7] *Xtensa Synthesizable Processors Unleash the Potential of System-On-Chip Designs*. Sales brochure. Tensilica Inc, 3255-6 Scott Blvd. Santa Clara, CA 95054.

[8] R.K. Gupta, G. De Micheli. "Hardware-Software Cosynthesis for Digital Systems". *IEEE Design & Test of Computers,* volume 10, number 4, September 1993, pp. 29-40

[9] M. Pflanz, H.T. Vierhaus, "Generating Reliable Embedded Processors". *IEEE Micro,* September-October 1998, pp.33-41

[10] I.C. Kraljic, G.M. Quénot, B. Zavidovique. "From Real-Time Emulation to ASIC Integration for Image Processing Applications". *IEEE Transactions on VLSI Systems*, September 1996, pp.391-404.