An Accelerated Datapath Width Optimization Technique for Area Reduction of Embedded Systems

Mesbah, Uddin M.
Department of Computer Science and Communication Engineering, Kyushu University

Cao, Yun
Department of Computer Science and Communication Engineering, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Kyushu University

出版情報：Proc. of IEEE/ACM International Symposium on System Synthesis (ISSS'02), pp.32-37, 2002-10. Association for Computing Machinery
バージョン：
権利関係：

An Accelerated Datapath Width Optimization Scheme for Area Reduction of Embedded Systems

Mohammad Mesbah Uddin Yun Cao Hiroto Yasuura
Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6–1 Kasuga-koen, Kasuga, 816-8580 Japan

ABSTRACT

Datapath width optimization is very effective for reducing the area of a custom-made embedded system. The trivial way of optimization is to iteratively customize, evaluate, and redesign a system to reach near an optimal value. The resulting effect is a long design time. In this paper, we introduce an effective scheme that accelerates design. A system-level pruning of design exploration space speeds up the optimization process. Through a single-pass simulation for a reference customization and a model for estimating and evaluating the system's performance, pruning of design space is achieved. Experimental results show that a substantial reduction in design time is possible.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Styles; B.7.2 [**Integrated Circuits**]: Design Aids; C.3 [**Special-purpose and Application-based Systems**]; C.5.4 [**Integrated Circuits**]: Computer System Implementation—*VLSI systems*; D.3.2 [**Programming Languages**]: Language Classifications—*specialized application languages*; D.3.4 [**Programming Languages**]: Processors—*code generation, compilers, optimization, retargetable compiler*; J.6 [**Computer Applications**]: Computer-aided Engineering—*computer-aided design (CAD)*

General Terms

Design, Measurement, Performance

Keywords

design of custom embedded systems, pruning of design exploration space

1. INTRODUCTION

An embedded system is a digital system which is tailored to realize a specific function. Such a system is generally constructed of several LSI-s (large scale integrated circuits) such as a processor core, memory cores and ASIC-s (application specific integrated circuits). Embedded systems are used extensively in everyday use. Reducing the area of these systems is a major design issue. Moreover, time-to-market design is very important for these systems.

In this study, we focus on the datapath width of the core processor. The datapath width has a strong effect on the area of the system. Datapath width optimization, proposed by Shackleford et al [2], explores an optimal embedded system by varying the datapath width 1-bit at a time. The key idea to obtain an optimized system is to fix the computational precision first, and then tune the underlying hardware, while preserving the computational precision as much as possible. Using datapath width optimization, a substantial reduction of system area (including memories) is reported in [3] [4] [5]. Thus, considering the datapath width is of paramount importance.

However, in order to optimize a system by synthesizing the datapath width, it requires – (1) a processor supporting this type of modification, (2) a high-level language to specify the computational precision of the application program, and, (3) a compiler for the system. As an example, to support the above requirements, a customizable processor (Bung-DLX) [4], and a high level language, called Valen-C along with the retargetable Valen-C compiler [5] is developed.

Nevertheless, the concept of datapath width optimization is quite new and little emphasis is given to the design methodology. The trivial approach to find an optimal datapath width for cost reduction is an exhaustive synthesis for all possible customizations. [2] [3], for instance, adopts the naive method. An exhaustive synthesis necessitates a number of iterative simulations for the family of custom-made systems obtained by modifying the datapath width. Each simulation requires processor customization, compiler generation, and compilation along with an evaluation for area and performance (in number of execution cycles). The repetitive and slow simulations result in a long turn-around-time. In order to reduce design time, we propose a scheme that accelerates design by a system-level pruning of the design exploration space.

The task of synthesizing hardware requires the solution of complex optimization problem. Problems arise when operations of varying width are assigned to a heterogeneous set of system components such as processor or memories. A key goal for our work is to provide a hardware solution where processors are synthesized so as to be adequately powerful to process data at a given computation rate yet minimum in area. Low-performance solutions should have less area

ISSS'02, October 2–4, 2002, Kyoto, Japan.

while high performance solutions having more. Therefore, it is worthwhile to consider performance as well as area to achieve our solution. In order to estimate performance, we customize a reference system and collect bitwidth related information from it. Performance for the other candidates are derived from the profile data and a model for estimation. Pruning of design exploration space is achieved by a high-level evaluation of system performance to check whether the constraints on performance are met.

The organization of this paper is as follows. Section 2 describes datapath width optimization and also defines the problem. Section 3 presents the proposed approach. Section 4 shows experimental results. Section 5 briefly describes some of the related work, and section 6 summarizes this paper.

2. DATAPATH WIDTH OPTIMIZATION

2.1 System Requirements

For this study, we assume a cache-less, non-pipelined system integrating a core processor, instruction memory and data memory. ROM and RAM are used as instruction memory and data memory, respectively. In addition, we assume that:

- computational precision of the application program can be determined,
- parameters of the core processor is customizable,
- performance constraints are given,
- the clock frequency of the core processor is fixed, and,
- cycle penalty ratios are known.

Cycle Penalty Ratio: Cycle penalty occurs when an operation instance becomes multiple precision because the operation needs more execution cycles than the single precision case. For an operation instance with exec_{sp} instruction(s) for single precision and exec_{pr} instructions for precision pr, we define the cycle penalty ratio as,

$$cpr_{pr} = \lceil \frac{exec_{pr}}{exec_{sp}} \rceil$$

If, for example, an operation executes 4 machine instructions for its single precision instance, and its cpr_2 is 2, then a double precision instance of the operation would require 2×4 = 8 machine instructions.

2.2 Effect of Datapath Width Reduction

2.2.1 Effect on CPU Area

The area of the processor almost linearly decreases with the reduction of the datapath width. Generally, the reduction of the datapath width makes processor size small, but causes loss of performance. This is because the number of execution cycles increase since some of the single precision operations become double or triple precision one. By a single precision operation, we mean an operation whose required maximum width is not larger than the datapath width. A multiple precision operation requires a maximum width larger than the datapath width. For example, an addition of two 28-bit data is a single precision operation on a 32-bit processor, but is a double precision operation on a processor whose datapath width is 20 bits.

2.2.2 Effect on RAM Area

Changing the datapath width affects the size of the data memory (RAM) as well as processor. Let us consider a program including two variables in Figure 1(a), and assume that two variables x and y require at most 18 bits and 26 bits, respectively. When the datapath width is 32 bits, two words are necessary to store those two variables. Therefore the amount of the data memory is 32 × 2 = 64 bits (Figure 1(b)). Since the minimum size required to store them is only 18 + 26 = 44 bits, 20 bits of them are unused (about 30%). When the datapath width is 9 bits, two words and three words are necessary for x and y, respectively, and the unused area is only 1 bit (Figure 1(b)). However, the RAM size does not decrease monotonically with the reduction of the datapath width. Quite many unused bits can be eliminated by determining the datapath width appropriately.

2.2.3 Effect on ROM Area

The ROM size, which is calculated (in bits) by multiplying the instruction word length by the number of instructions stored in the ROM, is also affected by the reduction of datapath width. When the datapath width is reduced, the number of instructions in the ROM increases. For example, an addition of the data whose maximum width is 26 bits (Figure 1(a)), is executed by only one instruction on a 26-bit processor (Figure 1(c)). When the datapath width is 16 bits, two instructions, namely, additions of lower 16 bits and upper 10 bits (2 bits for x) are required as shown in Figure 1(c). Furthermore load and store instructions may be required because of the storage of registers.

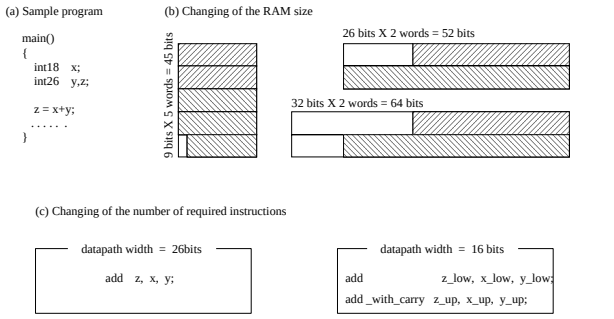

Figure 1: Effect of width reduction – (a) sample program, (b) effect on data memory, (c) effect on instruction memory.

2.2.4 Effect on Total System Area

The total system area depends on the individual area of the processor, RAM and ROM. Since any change in the datapath width imposes different behavior in the individual area changes, the total system area does not monotonically change for the datapath width.

2.3 Datapath Width Optimization

When designing processor systems, it is sufficient to set the datapath width equal to the largest bit-width required by the application software. Having the datapath wider only results in extra size, but no increased performance. Shackleford et al. indicates that there is an area reduction opportunity where the datapath width is smaller than the maximum size of variables in the application program [2].

Criteria for optimality varies with design. Some applications may require small system area while some other requiring high performance circuitry. Processor and memories show different system/area characteristics on narrowing the datapath width or modifying other parameters. Moreover, it might be impossible to sacrifice performance over an acceptance level. Therefore a trade-off policy considering the performance or the area consumed by the processor, memories and the whole system becomes necessary.

2.4 Evaluate-and-Redesign Approach for Optimization

Although, there are efforts on how to gain advantage of the knowledge of bitwidth, design methodologies adopt a trivial evaluate-and redesign approach (Figure 2). Initially the design begins with a customizable processor and the target application source program written in a high-level language such as "C".

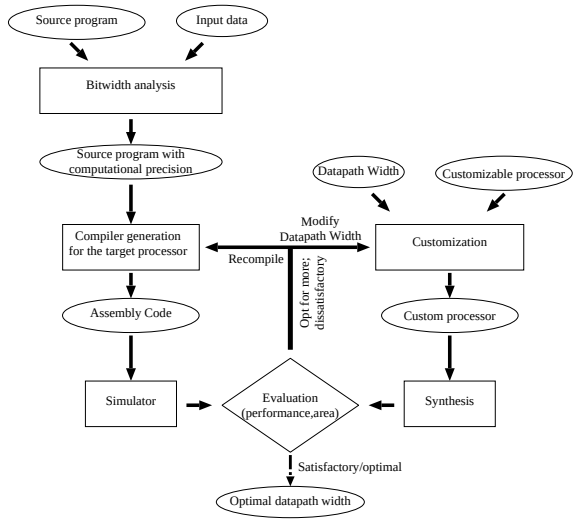

Figure 2: Design flow of a trivial approach.

The design flow consists of four phases:

- Phase 1: The original source program of the target application is rewritten in a language that can specify computational precision. Bitwidth analysis technique (*e.g.* [6]) is used in this phase.
- Phase 2: The processor is customized for a datapath width.
- Phase 3: A compiler is generated and the rewritten program is compiled for the customized processor.
- Phase 4: A low-level evaluation (based on logic-synthesis and simulation) checks whether the generated system satisfies design constraints.

After phase 4, if the design constraints are not met, or, in order to opt for a better suit, phase 2 through phase 4 are repeated by changing the datapath width 1-bit at a time. However, where a family of customized systems should be synthesized for optimization, the repetition becomes a bottleneck in design time. To address this, we define the problem as:

"How the area of an embedded system can be minimized speedily by datapath width optimization?"

We show that using a one-pass simulation, it is possible to develop an estimator function to generate performances for a set of systems with different datapath width. Then pruning of the design exploration space is done by a system-level evaluation of system performance to check whether the constraints on performance are met.

3. AN ACCELERATED SCHEME FOR DATAPATH WIDTH OPTIMIZATION

In order to solve our optimization problem, it is necessary to estimate the performance of the system as a function of datapath width at an early stage of system design. Given a target processor along with a set of compiler specifications, the number of execution cycles is directly related to the code sequence of the application software. With an insight into the software and the way of its implementation on the hardware, it is possible to estimate the performance of the system through a system-level simulation.

As an illustration, let us consider the following program [1] that will be executed on two different processors with datapath width of 32 bits and 21 bits, respectively.

```
main()
{
  int16  x;  /* x requires a maximum of 16 bits */
  int26  y;  /* y requires a maximum of 26 bits */
  int30  z;  /* z requires a maximum of 30 bits */

  z = x + y;
}
```

When the target processor has a datapath width of 32 bits, the compiler should convert the above program into the machine code sequence as shown in Table 1. Table 1, however, also presents the machine code sequence for a processor having a datapath width of 21 bits.

32-bit processor		21-bit processor	
Instruction	Description	Instruction	Description
load x,R1	$\leftarrow x_{15} \sim x_0$	load x,R1	$R1 \leftarrow x_{15} \sim x_0$
load y,R2	$R2 \leftarrow x_{25} \sim y_0$	load y_{low},R2	$R2 \leftarrow y_{20} \sim y_0$
add R2,R1	$R1 \leftarrow R1 + R2$	load y_{up},R3	$R3 \leftarrow y_{25} \sim y_{21}$
store R1,z	$z_{29} \sim z_0 \leftarrow R1$	add R1,R2	$R2 \leftarrow R1 + R2$
		addc #0,R3	$R3 \leftarrow R3$ + carry
		store R2,z_{low}	$z_{20} \sim z_0 \leftarrow R2$
		store R3,z_{up}	$z_{29} \sim z_{21} \leftarrow R3$

Table 1: Instruction sequence of the sample program for two processors with datapath width of 32-bits, 21-bits.

Now, let us focus on the original program. For the assignment operation, we can easily determine the required bitwidth for each of its operators, operands and results:

Operator	Width	Operand	Width	Operand	Width
+	26	x	16	z	30
=	30	y	26		

Table 2: Analysis of an operation instance.

For a 32-bit processor, all of the necessary operations are of single precision (Table 3). However, for the 21-bit processor case, the number of multiple precision operations increases (Table 3).

operation	32-bit processor	21-bit processor
load(x)	1 *sp* load (*cpr = 1*)	1 *sp* load (*cpr = 1*)
load(y)	1 *sp* load (*cpr = 1*)	1 *dp* load (*cpr = 2*)
add(+)	1 *sp* add (*cpr = 1*)	1 *dp* add (*cpr = 2*)
store(=)	1 *sp* store (*cpr = 1*)	1 *dp* store (*cpr = 2*)

Table 3: Precision instances of operations on two processors with datapath width 32-bits and 21-bits, respectively. *sp* indicates single precision operation, *dp* indicates double precision operation. *cpr* stands for cycle penalty ratio.

Assuming that each single precision operation requires a single instruction, we get

$$\sum(cpr) \cdot (number\ of\ instructions) = 4,$$

which is exactly equal to the total machine instructions generated by the compiler for a 32-bit processor. For a 21-bit

[1] In Valen-C, the C programming language is augmented to support this kind of declarations [5].

processor,

$$\sum(cpr) \cdot (number\ of\ instructions) = 7.$$

This also exactly matches the compiler generated code for a 21-bit processor. The use of cycle penalty ratio is important because an n^{th}-precision instance of an operation may not be simply n-times of the single precision instance.

Thus, it is possible to predict the total number of execution cycles for any operation. Consequently, if we count the number and necessary width of all the referenced operations, we can estimate the total number of execution cycles for any application. Hence, it is possible to estimate the performance (in number of execution cycles) at an early stage of design.

3.1 Design Flow of the Proposed Approach

The design flow of the proposed approach is shown in Figure 3. The flow consists of the following phases:

- Phase 1: The original source program of the target application is rewritten in a language that can specify computational precision. Bitwidth analysis technique, (*e.g.* [6]), is used in this phase.
- Phase 2: The system performance is estimated for an initial value of datapath width. This process consists of customizing the processor, generating compiler and compiling the application program for that processor, simulation, synthesis and so forth. Next, analysis for access-count of variables, and precision and access count of functions is done. The system performance is then estimated for a number of datapath widths with an estimator function.
- Phase 3: Datapath widths that fail to satisfy the performance constraints are excluded. Thus, we get a set of *candidate solutions* (accepted datapath widths).
- Phase 4: The processor is customized for a candidate datapath solution.
- Phase 5: The rewritten source program is compiled for the customized processor.
- Phase 6: A low-level evaluation checks whether the generated system satisfies the constraints for area. If the constraints are not met or in order to find a better system, phase 4 to phase 6 are repeated for the remaining candidates. However, with a reduced number of design parameters obtained at phase 3, the number of repetitive simulations is reduced.

In this section, we cover the main features of our proposed approach.

3.2 The Reference System

The reference system is assumed to have a datapath width equal to the largest variable in the program. However if the width exceeds the capability of the existing technology, the widest possible supported-configuration is used instead. Initially, the performance of the reference system is estimated (Phase 2). Estimation process includes customizing the core processor for the reference datapath width, compiling the application program for the customized processor, simulation and synthesis.

3.3 Analysis and Performance Estimation

The purpose of analysis phase is to take profile data for variable and operation instances. Analysis is done by executing the application program with a set of input data and with the reference system being the target. Profile data includes the number each variable is accessed throughout the program run. It also includes the precision and the

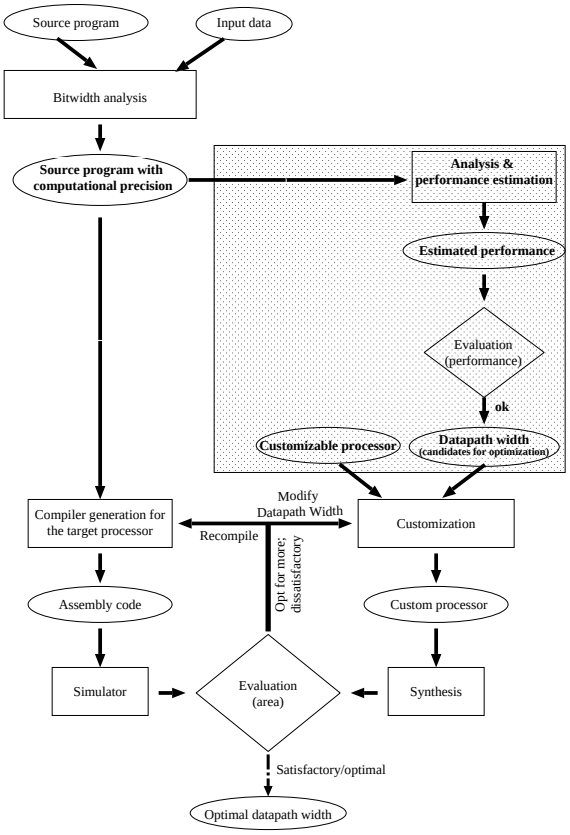

Figure 3: Design flow of the proposed approach which accelerates design time through a system-level pruning of design exploration space. The shaded portion outlines the pruning phase.

number of accesses to each operation (function) instances. Variables are divided into groups according to their effective bitwidth. Each function contains a subgroup, divided according to their precision on different instances.

3.3.1 Assumptions

The estimation model of our proposed approach utilizes the following assumptions:

- there is no register crowding, and,
- store operation is necessary for each assignment operation.

3.3.2 Estimation Model

System performance is related to the number of execution cycles under typical inputs. In this study, the performance (P_{func}) for a given datapath width ω, is estimated by

$$P_{func}(\omega) = \frac{\beta}{\aleph_{exec\_cycle}(\omega)}$$

where β is a constant and $\aleph_{exec\_cycle}(\omega)$ is the total number of execution cycles for the application program necessary for memory accesses for that datapath width. A measure of $\aleph_{exec\_cycle}(\omega)$ as a function of datapath width is given by

$$\aleph_{exec\_cycle}(\omega) = \sum_{op_i} N_{exec\_cycle}(op_i, \omega)$$

where *op* is operation (*suffix i* is used to denote instance), $N_{exec\_cycle}(op_i, \omega)$ is the number of execution cycles for that operation instance op_i on a processor/system with datapath width ω and is determined by

$$N_{exec\_cycle}(op_i, \omega) = cpr_{pr}(op_i, \omega) \cdot n_{exec\_cycle}(op_{sp}, \omega)$$

where, pr is the precision of the operation instance, cpr is the cycle penalty ratio, and $n_{exec\_cycle}(op_{sp}, \omega)$ is the number of execution cycles for a single precision execution. Note that, our analysis result is used to determine the sum of $N_{exec_cycles}(op_i, \omega)$.

However, an operation described in a high-level language may need arithmetic/logic instruction(s), load/store instruction(s), or both. *op* stands for such an operation and should be understood from the context. If the operation instance involves only load/store instruction(s) (i.e., *op* = load/store), pr is determined by the size of the variable to be loaded (or, stored) and the datapath width ω. For any operation instance involving arithmetic/logic instruction(s) (i.e., *op* = function), pr is determined by the operation itself, the size of its operands (variables), and the datapath width ω. Nevertheless, it is likely that an operation would contain both kind of operations and a combination of the above is necessary.

For a datapath width equal to the size of the application program's largest variable, all the operations of that program will be single precision. If we narrow the datapath width, some of the operations would become multiple precision ones. This will increase the number of execution cycles for those operations. The net effect is an increase in the total number of execution cycles.

Now, with all the necessary information described above, it is possible to determine system performance as a function of datapath width.

3.3.3 First-Order Compensation

An approximation compensates for the deviations of performance (P_{func}) introduced by our profile-data-based estimator function. The compensated estimated performance, $P_{est(\omega)}$, for a system with datapath width ω is given by

$$P_{est}(\omega) = \frac{P_{sim}(ref_\omega)}{P_{func}(ref_\omega)} \times P_{func}(\omega)$$

where, $P_{sim(ref_w)}$ is the performance for the reference system with datapath width ref_w, $P_{func(ref_\omega)}$is the estimated performance for the reference system, and, $P_{func(\omega)}$ is estimated performance for a system with datapath width ω.

3.4 Performance Evaluation and Pruning of Design Exploration Space

At this stage of design, (1) the performance constraints, and, (2) system performance as a function of datapath width are known. It is, therefore, easy to determine which of the systems satisfy the performance constraints. Datapath widths satisfying the performance constraints are accepted as candidates for further evaluation.

3.5 Detailed Design Phase

Detailed design is done with the Evaluate-and-Redesign approach for a range of accepted datapath width values. However, with a candidate datapath width, the core processor is customized and the application program is compiled for that processor. A simulation is used to determine the area of the system. Again, an evaluation checks whether the customized system satisfies the design goals. In case the design goals is not met, a repetition is necessary as before. Since, we evaluate and redesign a reduced number of of targets, an acceleration is gained.

4. EXPERIMENTAL RESULTS

In this section we will illustrate our experimental results. We used Bung-DLX as our target processor. Valen-C is used to describe the computational precision of the application program. Two applications are chosen for the experiment:

1. Lempel-Ziv data-compression algorithm [7], and,
2. CCITT adpcm g721 encoder.

Both the programs are originally written in C. They are tested against standard benchmark inputs. The system performance is assumed to lie within 5% of the reference case (32-bits for Lempel-Ziv, 19-bits for adpcm). However, the range of possible datapath widths is 32-bits $\sim$ 8-bits (8, because of a restriction of the Valen-C compiler).

In Figure 4, we show the estimated performance for the Lempel-Ziv program. Estimated performance is shown for both the naive approach and the proposed approach for datapath widths between 8-bits $\sim$ 32-bits.

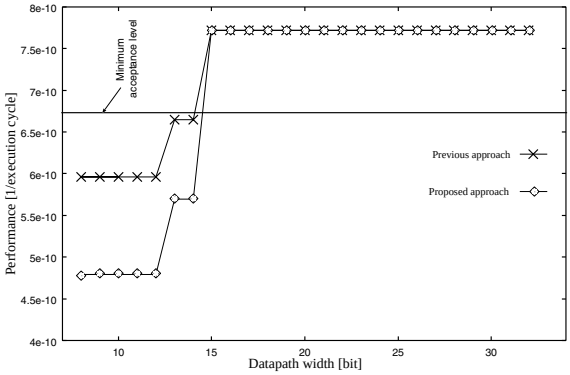

Figure 4: Estimated performance for the Lempel-Ziv program.

From Figure 4 it is easily verified that the acceptable range of datapath width is 32-bits $\sim$ 15-bits. That is, the reduction of design space is approximately 28%. We now continue with these values and opt for the optimal datapath width that minimizes area. The optimal datapath width that yields a minimum system area is determined to be 15-bits. Below, we show the required time for the naive and proposed approach (Table 4).

Design phase	Necessary time	
	Previous approach	Proposed approach
Analysis	α	α'
Processor customization, Compiler generation, & compilation	25β	1β

Table 4: Required design time for pruning of design exploration space (Lempel-Ziv).

At present, we do not have an automatic tool to use for our analysis. Assuming that our analysis scheme is included in the compiler specifications along with the bitwidth analyzer, α' will assume a very close value to α. For performance estimation, we have successfully reduced the number of simulations to 1. However, evaluation for area requires an area estimation model. Through processor synthesis, we estimated CPU area. Compiler generation and compilation is used to derive area for memories. Hence, at this stage we carried out another 17 simulations for the (Lempel-Ziv program) in order to estimates the system area for a set of candidates. Nevertheless, repetition introduced here can still be reduced by using a better model for evaluation. Time required for each of the simulations (β) is usually much greater than that of the analysis phase (α, α'). Neglecting the effect of α, and, α', design time reduction is approximately 28%.

For the adpcm encoder, The range of possible datapath widths is 19-bits $\sim$ 8-bits. Estimated performance for this case is shown in Figure 5. Estimations are shown for both the naive and the proposed approaches.

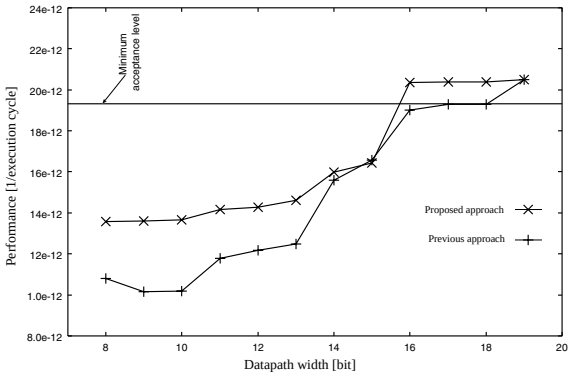

Figure 5: Estimated performance for the CCITT adpcm encoder program.

The range acceptable values of datapath width is 16-bits $\sim$ 19-bits. Thus, the reduction of design space is about 66%. The optimal datapath width that yields the minimum system area is determined to be 16-bits. The required time for the naive and proposed approach is shown in Table 5. Design time reduction is approximately 66%.

Design phase	Necessary time	
	Previous approach	Proposed approach
Analysis	α	α'
Processor customization Compiler generation & compilation	12β	1β

Table 5: Required design time for pruning of design exploration space (CCITT adpcm encoder).

4.1 Discussion

In order to achieve our goal, the performance estimated with our approach must be reliable – if the estimated performance is too much erroneous, we might exclude the space that contain the actual solution. To verify our approach, we correlate the estimated performance obtained by our approach and the other one (Figure 4, 5).

For the Lempel-Ziv program, estimations with our proposed approach is deviated 20% at the most. The minimum deviation is 0.03%. For datapath width equal to 15-bits or above, the error is below 0.06%, which is very substantial. However, for the adpcm encoder, the deviation rise up to 34%, at the narrower datapath widths. Deviation lies below 7% for datapth width 19-bits $\sim$ 14-bits.

Despite the deviation introduced, the curves for the estimated performance are almost alike – the one with our approach having a little more slope at some points. While it is clear that our approach correctly prunes the design space, we could improve reliability by adding a single or two to customization-targets for detailed design.

Our approach uses a very simple model with several restrictions. The effect of *constant propagation*, *spill-instructions*, or, *compiler directed optimizations* are not reflected adequately with those restrictions. For a narrower datapath width, the number of codes are likely to increase and the compiler specifications assume an important role. Therefore, consideration of compiler specifications need to be investigated for a better implementation of our approach.

5. RELATED WORK

Bitwidth has been exploited in a number of previous efforts. [9] emphasizes the careful treatment of possible value ranges. Their work targets FPGA implementation. [8] emphasizes sparse patterns of bits considering detailed information about each bit position separately. Considering the effects of quantization error for fixed point operations, low order bits are discarded in [10].

Our approach customizes a conventional processor with respect to bitwidth. As indicated in [2], detailed bitwidth information on operations is used to explore an optimal system from a family of processors obtained by varying the datapath width. Bitwidth information on operations allows the system to determine the precise number of computational steps required for each operation.

6. CONCLUSION

In this paper, we proposed an efficient scheme for determining an optimal datapath width which yields minimum system area. Our approach was to prune the design space at system-level and reduce the number of low-level synthesis and simulation targets. Thus, we could achieve acceleration in design time.

Although our final goal was area minimization, our approach can be enhanced for the minimization of power/energy consumption as well. Again, a combination of area and power/area consumption can also be our final minimization objective. Currently, we are working on developing a tool to support our analysis. Consideration of compiler specification is also under investigation. Analysis and estimation model for a real world system with cache memories and pipelines remains as our future work.

7. REFERENCES

[1] Scott Mahlke, *et al.*,"Bitwidth Cognizant Architecture Synthesis of Custom Hardware Accelerators," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Synthesis, Vol. 20, no. 11, pp. 1355-1371, November 2001.

[2] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, A. Inoue, and H. Yasuura, "Embedded system cost optimization via datapath width adjustment," IEICE Trans. Inf. & Syst., vol. E80-D, no. 10, pp. 974-981, Oct. 1997.

[3] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko, "Embedded System Design Using Soft-Core Processor and Valen-C," Journal of Information Science and Engineering, No. 14, pp. 587-603, August 1998.

[4] F. N. Eko, A. Inoue, H. Tomiyama, and H. Yasuura, "Soft-Core Processor Architecture for Embedded System Design," IEICE Trans. on Electronics, Vol. E81-C No. 9, pp. 1416-1423, Sep. 1998.

[5] A.Inoue, H. Tomiyama, T. Okuma, H. Kanbara, and H. Yasuura, "Language and Compiler for Optimizing Datapath Width of Embedded Systems," IEICE Trans. Fundamentals, Vol. E81-A, No. 12, pp. 2595-2604, Dec. 1998.

[6] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T. Okuma, and H. Yasuura, "Variable Size Analysis for Datapath Width Optimization," Proc. of Asia Pacific Conference on Hardware Description Languages (APCHDL '98), pp. 69-74, July 1998.

[7] http://www.data-compression.com/loosless.html

[8] M. Budiu, S. Goldstein, K. Walker, and M. Sakr, "Bitvalue inference: Detecting and exploiting narrow bitwidth computations," in Euro-Par 2000 Parallel Processing.

[9] M. Stephenson, J. Babb, and S.Amarasinghe, "Bitwidth analysis with application to silicon compilation," in Proc. SIGPLAN'00 Conf. Programming Language Design and Implementation, June 2000, pp. 108 - 120.

[10] K. I. Kum and W. Sung, "Word-length optimization for high-level synthesis of digital signal processing systems," in Proc. 1998 IEEE Workshop on Signal Processing Systems, Oct. 1998, pp. 142 - 151.