Abstract—Entire systems embedded in a chip and consisting of a processor, memory, and system-specific peripheral hardware are now commonly contained in commodity electronic devices. Cost minimization of these systems is of paramount economic importance to manufacturers of these devices. By employing a variable configuration processor in conjunction with a multi-precision compiler generator there are situations in which considerable system cost reduction can be obtained by synthesizing a CPU that is narrower than the largest variable in the application program.

I. INTRODUCTION

As recent papers addressing the hardware/software codesign dichotomy [1]–[3] will attest, cost reduction of embedded-system chips is of paramount economic importance due to their wide application in commodity electronic devices. Hardware/software codesign seeks first to insure adequate performance and then to achieve minimum cost.

The majority of these designs have been implemented with conventional, fixed-design microprocessor cores and megacells. The introduction of the “soft core” processor [5], the ASIP (application specific integrated processor) [6], and the variable configuration processor [4] have provided and additional avenue for embedded system cost-optimization.

We intend to demonstrate that, given a variable bit-width CPU synthesizer and a multi-precision-compiler generator, there is further opportunity for embedded system size optimization in the region where the CPU is narrower than the largest variable in the application code.

II. MEMORY-CPU SIZE TRADEOFFS

A. Scope and Assumptions

First, we should define the scope of this paper and how it relates to the definition of an embedded system. For the purpose of this paper we are interested in the effect on the cost of an embedded system that is caused by changing the size of a single parameter: the CPU bit-width. To this end, it is adequate to define an embedded system as a processor along with its data and instruction storage all implemented on a single chip.

The processor that we will use in this study is a variable configuration RISC processor that can be synthesized to any bit-width. It has separate instruction and data memories. The instruction memory is implemented in ROM and the data memory is implemented in RAM and is equal to width of the CPU data path. The instruction word length, although a function of synthesis parameters, remains invariant for the range of parameter variation in this study. We assume the existence of a multi-precision compiler generator (under development) that will automatically emit the necessary multi-precision instructions, according to the bit-width specification of each variable. As the CPU bit-width is varied, the high-level source code will remain unchanged, although the number of instructions emitted from the compiler will vary according to the number of multi-precision instructions.

B. The Effect of Narrowing the CPU

In the design of an embedded system employing a variable configuration processor, the first and most predictable stage of optimization is where the processor bit-width equal to that of the largest variable in the data memory. At this point, which we term the single-precision point, the CPU and data memory are equal in width to the largest variable and all instructions in the instruction memory are single-precision.

The single-precision point for the tiny example in Fig. 1 is where the CPU and data memory are 5 bits wide. At this point, the largest variable, A, requires exactly one word of data memory and the other variables, B–E, require less than a full word each. Reducing the CPU bit-width by 1 bit has several effects:

• the data memory becomes 1 bit narrower, but also 1 word larger (but still, a net reduction of memory area of 1 bit) due to variable A now requiring double precision
• the instruction memory, which does not change width, also becomes 1 word larger due to A becoming double precision.

Changing the CPU width from 4 bits to 3 bits brings about a similar change to variable B. Variable A remains
unchanged due to a surplus of unused bits. The data memory is again 1 word larger and 1 bit narrower, but now 3 bits smaller than the 4-bit case. The instruction memory is also one word larger due to the double-precision variable B.

The last point at which variable A (representing the largest variable) can remain double precision is a CPU bit-width of 3. We term this bit-width the **double-precision point**.

In general, as the CPU bit-width is reduced, the data memory size (measured in total bits of memory) will decrease, and the instruction memory size (measured in total words) will increase.

The problem that we wish to address is: what is the effect on system cost as the CPU bit-width is reduced to beneath that of the largest variable? Specifically, we are interested in:

- the relationships between the relative and absolute sizes of data and instruction memories and their effect on memory and system costs
- the distributions of lengths of variables and references to these variables within the data and instruction memories and the effect of these distributions on the system cost change
- the relative magnitude of the CPU cost decrease per bit-width reduction
- and finally, where, if at all, a subsequent minimal system cost occurs as CPU bit-width is decreased.

We will use the word **cost** to refer to chip area as measured in equivalent 2-input NAND gates and **size** to refer to linear measure such as bits and words.

### III. BACKGROUND AND RELATED WORK

#### A. Processor Synthesis System

The processor synthesis system used in this study is the Satsuki processor synthesis and compiler generator system [4]. Satsuki, which is built on top of the Tsutsuji logic synthesis system, is the combination of the synthesis of a variable configuration processor (similar to the “soft-core” processor concept proposed by Yasuura [5] and the CPU kernel of the PEAS-I system for ASIP development proposed by Sato [7]) with a C-compiler generator [6]. Parameters controlling the synthesis of the processor (CPU bit-width, register file size, data and instruction memory sizes) also drive the C-compiler generator to produce a custom-made C compiler for every possible processor instance.

The synthesis parameters determine the configuration, cost, and performance of the CPU. They are fed directly to the processor synthesis module. The module consists of various Tsutsuji components such as adders, registers, multiplexers, busses, etc., all of which are in turn parameter-driven by various secondary parameters, derived from the primary synthesis parameters.

#### B. Compiler Generator

The Satsuki compiler generator prototype (based upon the work of Tomiyama [6]) is driven by the processor synthesis parameters and produces a single-precision, C-subset compiler. We are currently developing a C compiler generator that will produce a compiler capable of issuing multiple-precision instructions when a referenced variable has a bit-size greater than the CPU bit-width. To complement multi-precision compiler, the C language will be extended to include bit-length specification for individual variables.

#### C. CPU Architecture

The CPU synthesized by Satsuki is a variable configuration RISC processor with separate data and instruction memories and a three-port register file. All instructions execute in one clock cycle and each instruction occupies a single word of instruction memory. The instruction bit-width is a function of the synthesis parameters. Data are accessed as words and are the same width as the CPU.

Whereas application specific integrated processors (ASICs) [7]–[11] target specific application areas and achieve their optimization through a combination of instruction set selection/implementation and architectural variation, we have chosen a general purpose architecture with an invariant instruction set. We balance the performance

---

1. Satsuki derives its name from a variety of Japanese azalea that blossoms later in the season than the ordinary Japanese tsutsuji (azalea). We chose this name because the processor synthesis portion of Satsuki is built on top of the Tsutsuji logic synthesis system.

2. Tsutsuji was previously marketed by Yokogawa Hewlett-Packard Ltd. and is now marketed by Zunken Incorporated as VPS.
requirements of the application with the cost of the processor by varying the CPU bit-width and the register file size.

IV. SYSTEM-COST MODEL

In this section we will describe the characteristics of the chip technology used in the experiments, the means by which the costs of the CPU and memories are calculated, and how the instruction and data memory word-count expansion ratios are computed as a function of the bit-width distributions of variables and instruction writes. As a chip area measure, we will use the cost in equivalent 2-input NAND gates.

A. Chip Technology

The target technology for mapping was a 0.8 micron CMOS gate array (Mitsubishi V080200) [15]. In addition to the normal complement of gate-level modules, the technology contains RAM and ROM modules that can be synthesized to any bit-width, but must be implemented in terms of modules whose addresses are powers-of-2. RAM module size ranges from 32 words to 8192 words and ROM module size ranges from 64 words to 2048 words. Figure 2 illustrates the relative cost growth rates of equal-sized RAM and ROM modules in this technology.

B. CPU Cost Model

The single-precision-point CPU has a 25-bit data path and 16-word register file. Its initial cost is 7.8k gates and reduces linearly at 283 gates per bit as the CPU bit-width is narrowed.

C. Bit-Width Distributions

For our software model, two aspects of the source code must be considered:

• The static distribution of bit-widths of the program variables. This determines the word-count expansion rate of the data memory.

• The static distribution of target bit-widths of instructions operating on or moving data. This determines the word-count expansion rate of the instruction memory.

For each of the above distributions, two distribution shapes were chosen, each with three size variations. The first shape is the linearly decreasing distribution shown in Fig. 3. The figure shows the distributions for 100-, 200-, and 400-variable sets. The linearly decreasing distributions for the instruction memory are derived from these distributions. The 200-instruction case is the same distribution as the 200-variable case. The 800-instruction case is scaled up from the 400-variable case by 2X.

The number of elements in each distribution represents the starting condition at the single-precision point. A subsequent reduction of the CPU bit-width by 1 will cause the variables at the right end of the distribution to require 2 words each and the instructions writing these variables to become double-precision pairs. The means of modeling the memory word-count expansion will be discussed in the next subsection.

D. Word Count Expansion

The expanded word-count for both data and instruction memories is given by the following equation:

\[
W_i = \sum \frac{n_{sp}}{n_{cpu}} \cdot (\ )
\]

where \( n_{sp} \) is the single-precision point, \( n_{cpu} \) is the bit-width of the CPU, and \( \hat{w}_j \) is the number of words in the distribution at bit-width \( i \).

E. ROM and RAM Cost Models

The cost per bit among all of the ROM modules is nearly
the same, so it is possible to model ROM cost by a single equation:

\[
W_i = \left[ \frac{198}{800} \right] + 3
\]

where \( W_i \) is the number of instruction words and \( n_i \) is the number of bits in the instruction. For the 21-bit instruction memory used by the CPU in this paper, every 64-word block of ROM is equivalent in cost to 1544 gates (Fig. 2).

The per-bit cost of RAM varies significantly among modules, so it is not possible to predict it accurately with a single, simple equation. Therefore, each RAM module is modeled separately with the manufacturer’s cost formula (Table 1). The data memory cost for a given word count is then taken to be the minimum of all aggregate module combinations whose total memory size is adequate to contain the word count.

The final memory cost (as a function of CPU bit-width) is obtained by applying the expanded word count size to the memory cost formula (or function in the RAM case). We will examine the cost variation according to CPU bit-width in the next Section.

### V. Experimental Results

To determine system cost as a function of CPU bit-width, we conducted a series of 12 experiments, divided into two groups:

- Linearily decreasing distributions for both variables and instruction targets (upper half of Fig. 4).
- Uniform distributions for both variables and instruction targets (lower half of Fig. 4).

Each group consisted of six cases, generated by the three variations of each of the two distributions:

- The length-of-variable distribution had counts of 100, 200, and 400 variables. This is to say that the data memory sizes at the single precision point were 100, 200, and 400 words respectively (all 25 bits wide).
- The target-bit-width distributions had counts of 200 and 800 instructions, representing instruction memory sizes of 200 and 800 words.

The smallest system consisted of 100 variables and 200 instructions at a cost of 26.2k gates. The largest system consisted of 400 variables and 800 instructions at a cost of 64.6k gates.

#### A. Memory Cost Variation Due to CPU Bit-Width Change

**ROM cost change.** As the CPU bit-width is reduced, ROM cost grows monotonically (Fig. 4). This is due to the linear nature of the ROM size-cost model and the fact that the ROM bit-width does not change. The growth rate due to the uniform distribution is greater than that of the linearly decreasing distribution because, with the uniform distribution there are initially more variables being converted to double precision than with the linearly decreasing distribution.

**RAM cost change.** In contrast to the ROM, where only the single action of the word count increasing happens as the CPU bit-width is reduced, two things happen to the RAM as the CPU bit-width is decreased: (1) the word count increases as in the case of the ROM, (2) the RAM bit-width is reduced in step with the CPU. This leads to a memory with fewer bits (which tends to reduce the cost) as the unused bits are squeezed out, but more words (which tends to increase the cost). These two actions, coupled with the nonlinear cost growth of the RAM yield a general downward, but non-monotonic, cost trend as shown in Fig. 4. The cost decrease is greater with the linearly decreasing distribution because this distribution has more unused bits than the uniform distribution.

#### B. System Cost Change Under Decreasing Distribution

Referring to the upper half of Fig. 4: In the 100-variable, 200-instruction case, the CPU and data memory provide roughly equal portions of the system cost reduction whose minimum occurred at a CPU bit-width 13. This represents a 28% cost reduction from the initial system cost of 26.2k gates. Although the other two cases in the 200-instruction row (200 and 400 variables), show larger magnitude gate count reductions, the initial systems are larger and the cost-reduction percentages (33% and 34%) are similar to the first case. It’s clear from the graphs in the upper row that the cost reduction process is being dominated by the RAM cost reduction.

Moving to the second row of Fig. 4 (800-instruction case), the cost growth of the instruction memory is now beginning to predominate, although moderate cost reductions of 12% and 20% can still be realized for the 200-variable and 400-variable cases.

---

**Table 1**: RAM module costs as function of bit-width \( n \).

<table>
<thead>
<tr>
<th>Size (words)</th>
<th>Cost (gates)</th>
<th>Max bit-width</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>1284 + 107n</td>
<td>72</td>
</tr>
<tr>
<td>64</td>
<td>2180 + 182n</td>
<td>72</td>
</tr>
<tr>
<td>128</td>
<td>3972 + 331n</td>
<td>72</td>
</tr>
<tr>
<td>256</td>
<td>3972 + 662n</td>
<td>36</td>
</tr>
<tr>
<td>512</td>
<td>3972 + 1324n</td>
<td>18</td>
</tr>
<tr>
<td>1024</td>
<td>3972 + 2648n</td>
<td>9</td>
</tr>
<tr>
<td>2048</td>
<td>3972 + 5296n</td>
<td>4</td>
</tr>
<tr>
<td>4096</td>
<td>4016 + 10710n</td>
<td>2</td>
</tr>
<tr>
<td>8192</td>
<td>26018</td>
<td>1</td>
</tr>
</tbody>
</table>
Fig. 4 Twelve cases resulting from the combination of linearly decreasing distributions (top half) and uniform distributions (bottom half) for instruction target operand lengths (rows) and variable lengths (columns). Plotted are the component cost changes vs. CPU bit-width. The resultant system cost change (i.e., the sum of all component cost changes) is plotted as a heavy solid line. The initial CPU cost is 7.8k gates.
C. System Cost Change Under Uniform Distribution

Referring to the bottom half of Fig. 4: We see the same trends that were evident in the upper half, except now the magnitude of cost reduction is not as great. This is due to the uniform distribution having more variables in the area near the single-precision point. These variables, when converted to double-precision words, cause an immediate and rapid growth in the sizes of both the instruction and data memories.

D. Discussion

Clearly, there is cost reduction opportunity in the area where the CPU is smaller than the single-precision point. However, the amount of cost reduction is highly dependent upon the distribution of variables and the relative sizes of the data and instruction memories.

In general, we can say that if the data memory is equal or greater in cost to the instruction memory, then there is a very good chance of further cost reduction below the single-precision point. This possibility is further enhanced by a distribution of variables favoring shorter bit-widths.

There is also a dynamic factor that becomes more pronounced as the bit-width is reduced: this is the “register crowding” effect. This effect occurs as multi-precision variables are loaded into the register file. For example, each double precision variable in the register file reduces the effective register count by one. As the number of registers is reduced, the effective lifetime of variables in the register file is reduced and the number of load and store instructions is increased.

So, for the cases presented here, we can say that they represent best cases for the distributions involved and that the certainty of reaching a particular limit would decrease as the CPU bit-width was reduced. However, this is entirely a function of the program characteristics.

VI. CONCLUSION AND FUTURE WORK

We have introduced a means for further cost reduction of embedded systems beyond what is normally achieved with the hardware/software codesign paradigm. It is intended to complement, not replace, hardware/software codesign. By means of a processor synthesis system and multi-precision compiler generator (compiler generator currently under development), it should be possible to achieve further embedded system cost reduction in the area where the CPU bit-width is less than the bit-width of the largest variable (single-precision point).

Initial experiments have shown when the data memory (RAM) is initially equal or greater in cost than the instruction memory (ROM) and the distribution of the bit-widths of variables favors smaller words that cost reductions of 30% or more can be achieved over the single-precision point cost.

However, initial experiments were not able to explore the effect of artificial register file size reduction caused by the presence of multi-precision variables. With the introduction of the multi-precision compiler generator we plan to study to what degree the instruction memory cost is increased by this effect and to what degree the cost increase can be ameliorated by increasing the processor register file size.

REFERENCES