A Cosynthesis Algorithm for Application Specific Processors with Heterogeneous Datapaths

Yuichiro Miyaoka† Nozomu Togawa†† Masao Yanagisawa‡ Tatsuo Ohtsuki‡

Waseda University The University of Kitakyushu Waseda University Waseda University

miyaoka@ohtsuki.comm.waseda.ac.jp togawa@env.kitakyu-u.ac.jp yanagi@yanagi.comm.waseda.ac.jp to@ohtsuki.comm.waseda.ac.jp

† Dept. of Electronics, Information and Communcation Engineering, Waseda University

3–4–1, Okubo, Shinjuku, Tokyo, 169–8555, Japan, Tel: +81–3–5286–3396, Fax: +81–3–3203–9184

†† Dept. of Computer and Media Sciences, The University of Kitakyushu

1–1, Hibikino, Wakamatsu, Kitakyushu, 808–0135, Japan, Tel: +81–93–695–3264, Fax: +81–93–695–3368

‡ Dept. of Computer Science, Waseda University

3–4–1, Okubo, Shinjuku, Tokyo, 169–8555, Japan, Tel: +81–3–5286–{3392,3387 }, Fax: +81–3–{3204–4875, 3203–9184 }

Abstract— This paper proposes a hardware/software cosynthesis algorithm for processors with heterogeneous registers. Given a CDFG corresponding to an application program and a timing constraint, the algorithm generates a processor configuration minimizing area of the processor and an assembly code on the processor. First, the algorithm configures a datapath which can execute several DFG nodes with data dependency at one cycle. The datapath can execute the application program at the least number of cycles. The branch and bound algorithm is applied and all the number of functional units and memory banks are tried. For an assumed number of functional units and memory banks, an appropriate number of heterogeneous registers and connections to functional units and registers are explored. The experimental results show effectiveness and efficiency of the algorithm.

I. Introduction

General DSPs such as TMS320C2x[14], DSP56300[12], DSP16xx[10], ADSP-21xx[3], and [9] have heterogeneous datapaths. Heterogeneous registers (accumulate registers, for example) can have flexible bit width, while general purpose registers must have single bit width. Heterogeneous registers can satisfy application requirements with less hardware costs. Sophisticated heterogeneous datapaths including heterogeneous registers can execute application programs fast. Therefore, processors with heterogeneous datapaths can have small costs and achieve high performance.

For processors with heterogeneous datapaths, code optimization or generation is a struggled problem, since, for example, it must be considered which of heterogeneous registers a variable is bound to. Several retargetable compilers to processors with heterogeneous datapaths have been reported as in [5, 6, 7, 11, 17, 18]. These retargetable compilers, however, cannot always make a sufficient application code, since datapath configuration of the target processor is not always suitable for a given application program. We think that an application specific processor should be synthesized, especially for a processor with a heterogeneous datapath. A *Hardware/software codesign* method can be effectively applied to processors with a heterogeneous datapaths. Several researches on a hardware/software codesign for microprocessors have been reported as in [1, 2, 4, 13, 15]. However, they does not focus on heterogeneous datapaths. In [8] a hardware/software codesign environment have been proposed. In [8], given an application program and datapath configuration, the codesign environment generates a processor hardware description and the object code on the processor. However, another datapath configuration must be manually designed when estimated area or the execution time of a given application program on a processor with a datapath configuration is insufficient. At the design of a processor with a heterogeneous datapath, it is difficult to find a performance bottleneck and to design another appropriate processor datapath. Exploring datapath configuration and compiling an application program must be close each other.

Therefore we propose a hardware/software cosynthesis algorithm for processors with heterogeneous datapaths. Given a CDFG corresponding to an application program and a timing constraint, the algorithm generates a processor configuration minimizing area of the processor and an assembly code on the processor. First, the algorithm configures a datapath which can execute several DFG nodes with data dependency at one cycle. The datapath can execute the application program at the least number of cycles. The branch and bound algorithm is applied and all the number of functional units and memory banks are tried. For an assumed number of functional units and memory banks, an appropriate number of heterogeneous registers and connections to functional units and registers are explored.

This paper is organized as follows: Section II defines a processor architecture. Section III proposes a hardware/software cosynthesis algorithm for processors with heterogeneous datapaths. Section IV shows experimental results. Section V gives concluding remarks.

II. Architecture Model

This section defines a processor architecture model for our synthesis algorithm. Our VLIW type processor has the 3 stage pipelines composed of `IF`, `ID`, and `EXE` stages. Immediate values are decoded on the `ID` stage and written in the `ID/EXE` pipeline registers. The processor can have one or two data memory banks. The data bus width of data memory is fixed to bit width of processor basic bit width b_{basic}. Besides the number of data memory banks, configuration of *general purpose registers*, *heterogeneous registers*, and *functional units* can be changed. Figure 1 shows our processor model.

0-7803-8175-0/04/$17.00 ©2004 IEEE.

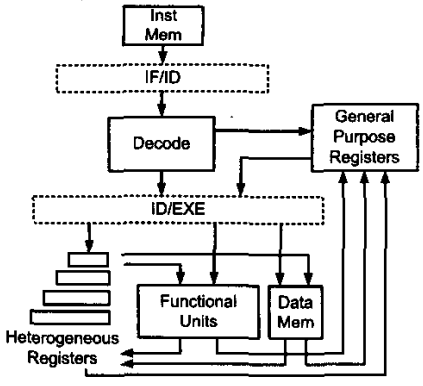

Fig. 1. Our pipeline architecture.

Bit width of *general purpose registers* is b_{basic}. In case that data of a general purpose register is read, the data is read at the ID stage and saved in the ID/EXE pipeline registers. In case that data is written in a general purpose register, data is generated at the EXE stage and written back to the register. The number n_g of general registers is given in advance.

Data of *heterogeneous registers* are read and written at the EXE stage. Data read from a heterogeneous register is (a) used for an input of a functional unit, (b) written back to a general purpose register, (c) written back to another heterogeneous register, or (d) written to data memory. To convert bit width of data in a heterogeneous register to bit width required for (a)-(d), a heterogeneous register can have a bit extended/extracted unit. (e) Output data of a functional unit, (f) data in a general purpose register, (g) data in another heterogeneous register, or (h) data from data memory is written to a heterogeneous register. The number of heterogeneous registers and bit width of each heterogeneous register can be changed. Which of (a)-(h) connections each register has and whether each register has a bit extended/extracted unit can be changed.

Functional units are such as ALUs, adding units, multiplying units, shift operation units, and bit extended/extracted units. For the same operation type, a functional unit can have different bit width of inputs and outputs. A processor has several functional units in a functional unit library $FU = \{f_1, f_2, \ldots f_u\}$. By connecting more than one functional units, a processor can have a datapath which can execute a operation and another operation continually. For example, by connecting an output port of a multiplying unit to an input port of an adding unit, multiplying and adding operation can be executed at one cycle.

We define minimum instructions to perform as a general processor. A processor must have a datapath which can execute (in one or several cycles) adding operation, logical and/or/xor operation, shift operation, and comparing operation for two variables a and b in the data memory mem_0, and write their operation results to mem_0. A processor must have a non-conditional branch operation and a branch-on-equal operation for a and b.

n_g, n_r and n_w, and n_h denote the number of general purpose registers, reading ports and writing ports of general purpose registers, and heterogeneous registers respectively. n_f denotes the number of functional units in a processor. $n_m \in \{1, 2\}$

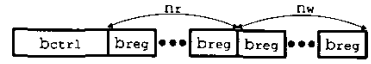

Fig. 2. Instruction format.

denotes the number of data memory banks and s_{mem_i} denotes the number of signals which can be written to data memory i. Let s_g be the number of signals which can be written back to general purpose registers and s_h^i be the number of signals which can be written back to a heterogeneous register R_h^i. Assume that a functional unit f_i should have n_{f_i} input ports and that the number of signals which connect a $j(0 \leq j < n_{f_i})$-th input of a functional unit should be $s_{f_i}^j$. Then an instruction set is composed of, at most,

$$n_{inst} = \prod_{i=0}^{n_m-1} s_{mem_i} \times s_g \times n_w \times \prod_{i=0}^{n_h-1} s_h^i \times \prod_{i=0}^{n_f-1} \prod_{j=0}^{n_{f_i}-1} s_{f_i}^j$$

instructions. Only the instructions which are required in order to execute a given application program and minimum instructions as mentioned above are synthesized. Bit length for an instruction code is $b_{ctrl} \leq \lceil \log n_{inst} \rceil$ and bit length for specifying general purpose registers is $(n_r + n_w) \times b_{reg}$, where b_{reg} is bit length to specify a general register and $b_{reg} = \lceil \log n_g \rceil$ (Fig. 2).

We estimate processor area by adding up area of (i) ID/EXE pipeline registers, (ii) general purpose registers, (iii) heterogeneous registers, (iv) functional units, and (v) multiplexers (for writing to general purpose registers, heterogeneous registers, and data memory banks, and input of functional units). Let a_{reg} and a_{mux} be area of a single bit register and a 2-1 multiplexer respectively. Area A_{pipe} of (i) is estimated as $A_{pipe} = a_{reg} \times (b_{ctrl} + n_r \times b_{basic} + n_w \times b_{reg})$. Area A_g is estimated as $A_g = a_{reg} \times n_g \times b_{basic} + a_{mux} \times (n_r \times b_{basic} \times (n_g - 1) + n_w \times b_{basic} \times n_g)$. Area A_h of (iii) is estimated as $A_h = a_{reg} \times \sum_{i=0}^{n_h-1} b_h^i$, where b_h^i is bit width of R_h^i. Area A_F of (iv) is estimated as $A_F = \sum_{fu \in F} a_{fu}$. Finally, area A_{mux} of (v) is estimated as $A_{mux} = a_{mux} \times \sum_{p \in \text{all}mux} (s_p - 1) \times b_p$, where s_p is the number of signals which can be input to a multiplexer p and b_p is bit width of the multiplexer p.

The clock period of a processor is estimated as critical path delay at the EXE stage, which is composed of delay of multiplexers, functional units, and writing back to registers. We assume that d_{mux} should be delay of a 2-1 multiplexer and we estimate delay of an s_p-1 multiplexer as $d_{mux} \times \lceil \log s_p \rceil$.

III. A HARDWARE/SOFTWARE COSYNTHESIS ALGORITHM FOR PROCESSORS WITH HETEROGENEOUS DATAPATHS

This section proposes an algorithm to synthesize a processor with a heterogeneous datapath.

A. Problem definition

A control flow graph (CFG) $G_c = (V_c, E_c)$ is defined as a graph representing control flow in a function. A CFG has no input and output edges. A basic block is a node of a CFG and has a data flow graph (DFG). A basic block has no branches and joins except for its starting and ending. A DFG $G_d = (V_d, E_d)$ is defined as a graph representing data flow. An ending node of a DFG may be connected to a starting node in another DFG. A DFG is a set of operations from a branch or

join to a next branch or join in an application. A node $v \in V_d$ is associated with an operation in a functional unit $f \in FU$ or a memory access operation. If the execution of a DFG node v_1 has data dependency to another DFG node v_2, a DFG has an edge (v_1, v_2).

T_{app}, the execution time of an application program, is calculated by a product of the number of total clock cycles to execute the CDFG and a clock period of the processor. Processor configuration includes the number of data memory banks, the number of heterogeneous registers, bit width of each heterogeneous register, the types and number of functional units, and the connections among data memory, general purpose registers, heterogeneous registers, and functional units.

Definition III.1 (Processor synthesis problem) *A processor synthesis problem is to determine processor configuration given a CDFG and a timing constraint in order to minimize processor area while the execution time of a given application program satisfies a timing constraint T_{max}, that is, $T_{app} \leq T_{max}$.*

B. Proposed algorithm

The proposed algorithm is composed of two phases. One determines initial processor configuration and the other explores processor configuration. Initial processor configuration has a sufficient number of functional units and heterogeneous registers, and can execute the application program in a small number of cycles, while processor area of the configuration is large. The branch and bound algorithm is applied and all the number of functional units and memory banks are tried. For an assumed number of functional units and memory banks, an appropriate number of heterogeneous registers and connections to functional units and registers are explored. In this way, we can obtain optimized processor configuration in a short time.

B.1 Initial processor configuration

To configure initial processor configuration, all the DFGs in a CFG are scheduled in the following method. To schedule a DFG $G_d = (V_d, E_d)$ associated with a CFG node $v_c \in V_c$ is to determine a pair (v, st) for all the nodes $v \in V_d$, $st \in \{1, 2, \ldots\}$, that is, to construct $S_{v_c} = \{(v, st) | {}^{\forall} v \in V_d\}$. A node $v \in V_d$ is a node executed by a functional unit (called *a functional node*) or a node accessing data memory (called *a memory accessing node*). A set of all the functional nodes in V_d is denoted as V_f. A set of all the memory accessing node in V_d is denoted as V_m, where V_f and V_m satisfy $V_d = V_f \cup V_m, V_f \cap V_m = \phi$. Our target processor can execute several functional nodes at one cycle which have data dependency to one another by connecting an output of a functional unit to an input of another functional unit. However, a processor cannot execute a node to load data from memory and a node to operate the loaded data at the same cycle and cannot execute a node to store data to memory and a node to operate the stored data at the same cycle. Therefore, if an edge (v_i, v_j) is included in E_d and v_i and $v_j \in V_d$ are assigned to a step st_i and st_j respectively, $st_i \leq st_j$ in $v_i, v_j \in V_f$ and $st_i < st_j$ in $v_i \in V_m$ or $v_j \in V_m$. Figure 3 shows the step assigning rule. DFG nodes are assigned to steps in a topological order. To assign a functional node and memory accessing node to a step st is realized by assigning all the functional nodes which can be assigned to st and memory accessing nodes satisfying resource constraints. Figure 4 shows this algorithm.

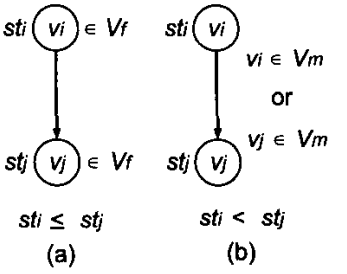

Fig. 3. A step assigning rule. (a) In case a predecessor and successor node of an edge is in V_f, (b) a predecessor or successor node is in V_m.

Inputs A CFG $G_c = (V_c, E_c)$ and DFG $G_d = (V_d, E_d)$.

Outputs A scheduling result $\{S_{v_c} | {}^{\forall} v_c \in V_c\}$.

Step 1. Select a CFG node $v_c \in V_c$. Execute $st \leftarrow 1$ and $S_{v_c} \leftarrow \emptyset$.

Step 2. Select $v \in V_d$ in $G_d = (V_d, E_d)$ corresponding to v_c in a topological order.

- **Step 2.1.** For $v \in V_f$, assign v to st if there are no v' where $v' \in V_m$ and $(v', v) \in E_d$. Execute $S_{v_c} \leftarrow S_{v_c} \cup (v, st)$.
- **Step 2.2.** For $v \in V_m$, assign v to st if a memory resource constraint is satisfied and a step st' assigned v' where $(v', v) \in E_d$ is less than st. Execute $S_{v_c} \leftarrow S_{v_c} \cup (v, st)$

Step 3. Go to **Step 4** if every node in V_d are assigned to a step. Otherwise, update $st \leftarrow st + 1$ and go to **Step 2**.

Step 4. S_{v_c} has been obtained. Select another CFG node v_c' and go to **Step 2**. Finish the algorithm when all the CFG nodes have been scheduled.

Fig. 4. Initial scheduling algorithm.

Based on an initial scheduling result, an initial processor configuration is configured. If a predecessor node and a successor node of an edge are assigned to different steps, the edge is assigned to a register. The lifetime of each variable is analyzed in advance.

First, we decide the number of functional units. Let $V_{v_c,st}$ be a set of nodes assigned to a step st in a node $v_c \in V_c$. $fu(v)$ denotes a functional unit or data memory corresponding to $v \in V_d$. The number of a functional unit f_i to execute $V_{v_c,st}$ in a cycle is $n_{f_i}(v_c, st) = |\{v \in V_{v_c,st} | fu(v) = f_i\}|$. Therefore, the number of a functional unit f_i which a processor has is,

$$n_{f_i} = \max_{{}^{\forall} v_c, {}^{\forall} st} \{n_{f_i}(v_c, st)\}$$

Similarly, the number of data memory banks is determined to be one or two.

Now, we propose an algorithm to configure the connections among functional units, data memory banks, and registers. We define a *processor configuration graph* in order to represent the connections. A processor configuration graph denotes $G_{pr} = (V_{pr}, E_{pr})$. A node in V_{pr} correspond to a pipeline register, a heterogeneous register, a functional unit, or data memory. A processor configuration graph has an edge (v_1, v_2) if there is a connection between hardware corresponding to v_1 and one corresponding to v_2. The number of nodes corresponding to functional units or data memory can be determined as mentioned above. Thus, there are n_{f_i} functional units associated with f_i in v_{pr}. There are one or two nodes associated with data memory banks. Corresponding to reading

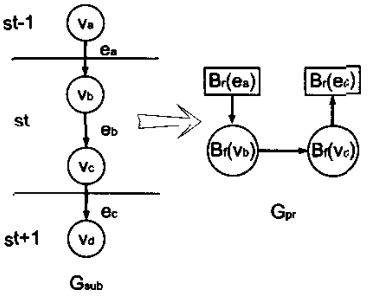

Fig. 5. An example of G_{pr} satisfying configuring conditions based on G_{sub}.

and writing general purpose registers, nodes associated with pipeline registers must be included in V_{pr}.

First, to perform as a general processor, our algorithm adds a node v associated with an ALU to V_{pr} and edges between v and nodes associated with pipeline registers.

The number of nodes associated with heterogeneous registers and a set E_{pr} of edges are decided by applying the following algorithm to all the steps of all the basic blocks and updating $G_{pr} = (V_{pr}, E_{pr})$.

Let E_{sub} be a set of edges where $(v_1, v_2) \in E_d$, $v_1 \in V_{v_c,st}$ or $v_2 \in V_{v_c,st}$. A subgraph $G_{sub} = (V_{sub}, E_{sub})$ of G_{pr} can be constructed, where V_{sub} is a set of all the nodes which are v_1 or v_2 of $(v_1, v_2) \in E_{sub}$. *Configuring conditions for G_{pr} in a step st* is defined as follows.

1. $v_d \in V_{v_c,st}$ is associated with $v_{pr} \in V_{pr}$. A mapping function is denoted as $B_f : V_{v_c,st} \mapsto V_{pr}$. If $v_{pr} = B_f(v_d)$, v_{pr} is corresponding to a hardware $fu(v_d)$. If $v_1, v_2 \in V_{v_c,st}$, $B_f(v_1) \neq B_f(v_2)$.

2. If $v_1, v_2 \in V_{v_c,st}$ for $e = (v_1, v_2) \in E_{sub}$, $(B_f(v_1), B_f(v_2))$ must be included in E_{pr}.

3. Let $E_r = \{(v_s, v_t) \in E_{sub}\}$ be a set of edges where $v_s \notin V_{v_c,st}$ or $v_t \notin V_{v_c,st}$. $e_r \in E_r$ is associated with $v_{pr} \in V_{pr}$. A mapping function is denoted as $B_r : E_r \mapsto V_{pr}$. If $v_{pr} = B_r(e_r)$, v_{pr} is corresponding to a heterogeneous register between $B_f(v_s)$ and $B_f(v_t)$. For $e_1 = (v_i, v_j), e_2 = (v_k, v_l) \in E_{sub}$, $B_r(e_1) = B_r(e_2)$ if $v_i = v_k$ and $B_r(e_1) \neq B_r(e_2)$ if $v_i \neq v_k$.

4. If $v_1 \in V_{v_c,st}$ and $v_2 \notin V_{v_c,st}$ for $e = (v_1, v_2) \in E_{sub}$, $(B_f(v_1), B_r(e))$ must be included in E_{pr}. Similarly, if $v_1 \notin V_{v_c,st}$ and $v_2 \in V_{v_c,st}$ for $e = (v_1, v_2) \in E_{sub}$, $(B_r(e), B_f(v_2))$ must be included in E_{pr}.

Figure 5 shows an example of G_{sub} and a graph satisfying configuring conditions for G_{pr} in a step st.

A mapping function B_f which corresponds $V_{v_c,st}$ to V_{pr} is decided so that the configuring condition 1 for G_{pr} is satisfied. Then, G_{pr} is updated as follows so that configuring conditions 2–4 for G_{pr} are satisfied.

1. If $(B_f(v_1), B_f(v_2))$ is in E_{pr} for $(v_1, v_2) \in E_{sub}$ where v_1 and v_2 are in $V_{v_c,st}$, E_{pr} is not updated. If $(B_f(v_1), B_f(v_2)) \notin E_{pr}$, $(B_f(v_1), B_f(v_2))$ is added to E_{pr}.

Inputs A scheduling result $\{S_{v_c} | \forall v_c \in V_c\}$ and a set of subgraph $G_{sub} = (V_{sub}, E_{sub})$.

Outputs A processor configuration graph $G_{pr} = (V_{pr}, E_{pr})$.

Step 1. Calculate the number n_{f_i} of f_i as

$$n_{f_i} = \max_{\forall v_c, \forall st} \{n_{f_i}(v_c, st)\}$$

Similarly, the number of data memory banks is determined to be one or two.

Step 2. Include nodes associated with functional units, data memory banks, and reading and writing general purpose registers. The numbers of nodes associated with functional units and data memory banks are decided at **Step 1**. $E_{pr} \leftarrow \phi$.

Step 3. Add a node v associated with an ALU to V_{pr} and edges between v and nodes associated with pipeline registers to E_{pr}.

Step 4. Pick up a CFG node v_c. $st \leftarrow 0$.

Step 5. Decide a mapping function B_f which associates functional nodes in G_{sub} at st to the nodes in G_{pr}. Try to update G_{pr} and calculate the G_{pr} cost.

Step 5.1. If $(B_f(v_1), B_f(v_2))$ is in E_{pr} for $(v_1, v_2) \in E_{sub}$ where v_1 and v_2 are in $V_{v_c,st}$, E_{pr} is not updated. If $(B_f(v_1), B_f(v_2)) \notin E_{pr}$, $E_{pr} \leftarrow E_{pr} \cup (B_f(v_1), B_f(v_2))$.

Step 5.2. Let us focus on an edge $e = (v_1, v_2) \in E_r$ where $v_1 \in V_{v_c,st}$ and $v_2 \notin V_{v_c,st}$. Let v_{reg} be a node which is associated with a heterogeneous register connected to an outgone edge of $B_f(v_1)$ and does not have st in lifetime. Update a mapping function B_r so that $B_r(e) = v_{reg}$. If there does not exist such v_{reg}, $V_{pr} \leftarrow V_{pr} \cup v_{reg}$ and $B_r(e) = v_{reg}$ is added to B_r.

Step 5.3. For $e = (v_1, v_2) \in E_r$ where $v_1 \in V_{v_c,st}$ and $v_2 \notin V_{v_c,st}$, $B_r(e)$ has been determined at an iteration by $st - 1$. If $(B_r(e), B_f(v_2))$ is not in E_{pr}, $E_{pr} \leftarrow E_{pr} \cup (B_r(e), B_f(v_2))$.

Step 6. Select B_f which has the minimum G_{pr} cost and update G_{pr}.

Step 7. $st \rightarrow st + 1$ and go to **Step 5**. if the steps for all the st are tried, go to **Step 4**. Finish the algorithm if all the CFG nodes are tried.

Fig. 6. A processor configuration algorithm.

2. Let us focus on an edge $e = (v_1, v_2) \in E_r$ where $v_1 \in V_{v_c,st}$ and $v_2 \notin V_{v_c,st}$. Let v_{reg} be a node which is associated with a heterogeneous register connected to an outgone edge of $B_f(v_1)$ and does not have st in lifetime. Update a mapping function B_r so that $B_r(e) = v_{reg}$. If there does not exist such v_{reg}, a new v_{reg} is included in V_{pr} and $B_r(e) = v_{reg}$ is added to B_r.

3. For $e = (v_1, v_2) \in E_r$ where $v_1 \in V_{v_c,st}$ and $v_2 \notin V_{v_c,st}$, $B_r(e)$ has been determined at an iteration by $st - 1$. If $(B_r(e), B_f(v_2))$ is not in E_{pr}, $(B_r(e), B_f(v_2))$ is included in E_{pr}.

The algorithm selects B_f which has the minimum G_{pr} cost and updates G_{pr}, where G_{pr} cost is defined by estimating processor area as mentioned in Sect. II. The processor configuration algorithm is shown in Fig. 6.

An initial processor configuration is obtained by applying the processor configuration algorithm to an initial scheduling result. An initial processor configuration has the minimum number of cycles executing the application program. Based on a model in Sect. II, area and a clock period of the processor is estimated.

Inputs A processor configuration graph $G_{pr} = (V_{pr}, E_{pr})$ obtained by means of applying the processor configuration algorithm.

outputs A updated processor configuration graph $G_{pr}{}' = (V_{pr}{}', E_{pr}{}')$.

Step. 1 Try to reduce for one of all the registers as following two reduction methods. Calculate processor area and the execution time of the application program.

1. Focus on two heterogeneous registers which have same bit width and have no overlap in their lifetime. If a register v_2 is , Update all the edges which have v_2 in either side of edges by replacing v_2 to v_1. $V_{pr} \leftarrow V_{pr} \setminus v_2$ and Update E_{pr}.
2. Reduce a heterogeneous register which has the same bit width as general purpose registers and Save a content of the heterogeneous register in a general register. Update all the edges which have the heterogeneous register v in either side of edges by replacing v_2 to a pipeline register. $V_{pr} \leftarrow V_{pr} \setminus v$ and Update E_{pr}.

Step 2. Actually update a processor configuration which has the minimum area under the timing constraint of all the candidates. New G_{pr} is obtained. If there are more than one such processor configurations, select the candidate in which the number of edges in E_{pr} including the node associated with the heterogeneous register is the most.

Step 3. Finish if there are no registers that can be reduced in **Step 1** or all the candidates do not satisfy a timing constraint.

Fig. 7. An algorithm to reduce heterogeneous registers.

B.2 Exploring a processor configuration

Based on an initial processor configuration, the maximum numbers of functional units and data memory banks are determined. We apply a branch and bound method which subproblems are constructed by branching about the number of functional units and data memory banks. When the numbers of functional units and data memory banks are determined, we have only to configure registers. Therefore we optimize processor configuration in a short time. We solve the sub-problem as follows. First we schedule a CDFG under a constraint of the numbers of functional units and data memory banks. Figure 4 can be easily applied to it by modifying that assigned nodes are restricted by resource constraints. The processor configuring algorithm is applied to the scheduling result. While a timing constraint is satisfied, a heterogeneous register reduction algorithm shown in Fig. 7 is applied.

IV. Experimental results

We have implemented the proposed algorithm in C++ on Sun Ultra Sparc 3 750MHz. We use gcc 2.95.3 as a compiler. To estimate area and delay of hardware, we use VDEC libraries (CMOS and 0.35 μm technology)[1]. We use area and delay in Table I for estimation of functional units. We assume that area and delay of 2-1 multiplexer are a_{mux} =167[μm^2] and d_{mux} =0.23[ns], respectively and that area and writing delay of a single bit register are a_{reg} =383[μm^2] and d_{reg} =0.40[ns], respectively. We assume that basic bit width of a processor is 16 bit.

The algorithm has been applied to an FIR filter and a DCT in which basic bit width of variables is 16 bit and 32 bit multiplying and accumulate operation are used. The results are shown in Table II. For different timing constraints, different processor configuration can be obtained. In order to compare our results with existing systems, we have picked up two processor synthesis systems (which are referred to as [15] and [16] in References. [15] synthesizes a simple processor with a homogeneous datapath. [16] synthesizes a processor which has two types of register files and a homogeneous datapath. Comparison results have been shown in Table III. It shows that our system can synthesize processors with less area than existing systems which synthesize processors with homogeneous datapaths. When the timing constraint of 60 μs is given, System 1 and 2 cannot output a processor configuration meeting the timing constraint.It is because processors synthesized by system 1 and 2 cannot have chains among the functional units and no processors satisfy the timing constraint. Our proposed algorithm synthesizes a processor with less area compared with System 1 because of the following reason: In a processor synthesized by system 1, all the registers and functional units must have bit width of 32 bits. In a processor synthesized by the proposed algorithm, however, registers and functional units can have flexible bit width. The processor can have both 16 bits and 32 bits resources. Our proposed algorithm synthesizes a processor with less area compared with System 2 because of the following reason: A processor synthesized by system 2 can have 32-bit and 16-bit registers and functional units. Since the synthesized processor, however, has a homogeneous datapath, it must have connections and multiplexers between all the registers and all the functional units. A processor synthesized by the proposed algorithm can have only connections and multiplexers required in order to execute a given application program. Therefore we can synthesize a processor with less area.

TABLE I
FUNCTIONAL UNITS.

Unit	Area [μm^2]	Delay [ns]
ADD16	25,259	1.44
ALU16	78,915	2.82
MUL16	356,948	5.71
ADD32	41,963	2.49
ALU32	151,194	3.25

V. Conclusion

In this paper, we proposed a hardware/software cosynthesis algorithm for processors with heterogeneous datapaths. In the future, we will incorporate SIMD functional units into the processor model and establish the algorithm to optimize the processor configuration.

[1]The libraries in this study have been developed in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration by Hitachi Ltd. and Dai Nippon Printing Corporation.

References

[1] H. Akaboshi, and H. Yasuura, COACH: A computer aided design tool for computer architectures, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E76-A, no. 10, pp. 1760–1769, 1993.

[2] A. Alomary, T. Nakata, Y. Honma, M. Imai, and N. Hikichi, "An ASIP instruction set optimization algorithm with functional module sharing constraint," in *Proceeding of 1993 IEEE/ACM International Conference on Comuputer-Aided Design*, pp. 526–532, 1993.

[3] Analog Devices, *ADSP-2100 Family User's Manual*, 1995.

[4] N. N. Binh, M. Imai, A. Shiomi, and N. Hikichi, "A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate count," in *Proceedings of 33rd Design Automation Conference*, pp. 527–532, 1996.

TABLE II
EXPERIMENTAL RESULTS.

App.	Const.[μs]	Area[μm^2]	Processor configuration
FIR	60	1,531,347	ALU16 × 1, MUL16 × 3, ADD32 × 2 16 bit heterogeneous registers× 7, 32 bit heterogeneous register × 1
	70	676,761	ALU16 × 1, MUL16 × 1, ADD32 × 1 16 bit heterogeneous registers× 5, 32 bit heterogeneous register × 2
DCT	60	656,057	ALU16 × 1, ADD16 × 3, MUL16 × 1, ADD32 × 1 16 bit heterogeneous registers× 6, 32 bit heterogeneous register × 1
	80	580,290	ALU16 × 1, MUL16 × 1, ADD32 × 1 16 bit heterogeneous registers × 6, 32 bit heterogeneous register × 1
	110	568,038	ALU16 × 1, MUL16 × 1, ADD32 × 1 16 bit heterogeneous registers × 4, 32 bit heterogeneous register × 1

TABLE III
RESULTS COMPARED TO EXISTING SYSTEMS.

FIR			
Const.	Area [μm^2]		
[μs]	Proposed	System 1 [15]	System 2 [16]
60	1,531,347	-	-
70	676,761	806,326	752,443

DCT			
Const.	Area [μm^2]		
[μs]	Proposed	System 1 [15]	System 2 [16]
60	656,057	776,326	705,471
80	580,290	720,323	649,468
110	568,038	693,918	643,340

[5] S. Fröhlich and B. Wess, "Integrated approach to optimized code generation for heterogeneous-register architectures with multiple data-memory banks," in *Proceedings of 14th Annual IEEE International ASIC/SoC Conference*, pp. 122–126, 2001.

[6] N. Ishiura, M. Yamaguchi, and T. Kambe, "A graph-based algorithm of operation binding for compilers targeting heterogeneous datapath," in *Proceedings of The 1998 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 395–398, 1998.

[7] N. Ishiura, T. Watanabe, and M. Yamaguchi, "A code generation method for datapath oriented application specific processor design," in *Proceedings of SASIMI2000*, pp. 71–78, 2000.

[8] N. Ishiura and T. Watanabe, "Datapath oriented codesign method of application specific DSPs using retargetable compiler," in *Proceedings of 2002 Asia-Pacific Conference on Circuits and Systems*, vol. 1, pp. 55–58, 2002.

[9] S. Kurohmaru, M. Matsuo, H. Nakajima, Y. Kohashi, T. Yonezawa, T. Moriiwa, M. Ohashi, M. Toujima, T. Nakamura, M. Hamada, T. Hashimoto, H. Fujimoto, Y. Iizuka, J. Michiyama, and H. Komori, "A MPEG4 programmable codec DSP with an embedded pre/post-processing engine," in *Proceedings of the IEEE 1999 Custom Integrated Circuits*, pp. 69–72, 1999.

[10] Lucent Technologies, *DSP1611/17/18/27/28/29 Digital Signal Processor Information Manual*, 1998.

[11] P. Marwedel, "Code generation for core processor," in *Proceedings of 34th Design Automation Conference*, pp. 232–237, 1997.

[12] Motorola, *DSP56300 24-bit Digital Signal Processor Family Manual (DSP56300FM/AD)*, 2000.

[13] Tensilica, *Xtensa Microprocessor: Overview Handbook*, http://www.tensilica.com.

[14] Texas Instruments, *TMS320C2x Datasheet*, 1998.

[15] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A hardware/software cosynthesis system for digital signal processor cores," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E82-A, no. 11, pp. 2325–2337, 1999.

[16] N. Togawa, M. Yanagisawa, and T. Ohtsuki, "A hardware/software cosynthesis system for digital signal processor cores with two types of register files," IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, vol. E83-A, no. 3, 2000.

[17] J. Van Praet, D. Lanneer, G. Goossens, W. Geurts, and H. De Man, "A graph based processor model for retargetable code generation," in *Proceedings of European Design and Test Conference*, pp. 102–107, 1996.

[18] M. Yamaguchi, N. Ishiura, and T. Kambe, "A binding algorithm for retargetable compilation to non-orthogonal DSP architecture," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E81-A, no. 12, 1998.