High-Level Synthesis Challenges and Solutions for a Dynamically Reconfigurable Processor

Takao Toi, Noritsugu Nakamura, Yoshinosuke Kato, Toru Awashima, and Kazutoshi Wakabayashi
System Devices Research Laboratories, NEC Corporation
Li Jing
NEC Informatec Systems, Ltd.
1753 Shimonumabe, Nakahara, Kawasaki, Kanagawa 211-8666, Japan
info@drp.jp.nec.com

Abstract

A dynamically reconfigurable processor (DRP) is designed to achieve high area efficiency by switching reconfigurable data paths dynamically. Our DRP architecture has a stand alone finite state machine and that switches "contexts" consisting of many operational and storage units in processing elements (PEs) and wires between them. Utilizing the resources not only in two spatial dimensions but also vertically (time-multiplexed) under accurate timing and area constraints imposes challenges for a high-level synthesizer for the DRP. We describe a C-based behavioral synthesis method which features data path generation with clock speed optimization. This is achieved by including the overhead of selectors in the scheduling algorithm, and considering a wire delay at each PE level. A new technique is introduced to achieve high area efficiency. It works by effectively allocating multiple steps into the context. From the original high-level synthesizer for application-specific integrated circuits, some of the basic rules such as operator and register sharing were completely changed due to the coarse grained and multi-context architecture. Experimental results show that the generated data paths are highly parallelized and well balanced between contexts. The delay controllability enables the highest throughput point to be found more easily.

Keywords: High-level Synthesis, Reconfigurable Processor, Dynamic Reconfiguration

1. Introduction

A dynamically reconfigurable processor (DRP) has a new programmable architecture that enables switching of time-multiplexed data paths. Each data path is configured as a "context" consisting of many operational and storage units and the wire connections between them. This enables the DRP to execute highly complex and parallel data paths.

The continuing growth in the demand for flexible, low power, and high-performance processors has led to the development of several new types of DRPs, such as Chameleon [1], IPFlex's DAP/DNA [2], Elixent's D-Fabrix, PACT's XPP [3], and others [4]. Note that there is another approach called configurable processor which adds customizable functions to prefixed CPU instructions such as Tensilica's Xtensa and Synfora's PICO. However, these new processors must be competitive with existing programmable chips, such as CPUs, digital signal processors (DSPs), and field programmable gate arrays (FPGAs).

The compiler plays an important role in this. There are two major challenges in developing compilers for DRPs. First, to compete with CPUs or DSPs, high programmability must be maintained while still having the strength of the parallelism of the wired logic. This is usually addressed during the design stage when developing a microprocessor. Consequently, the compiler for reconfigurable fabric plays a larger role because the boundary between the chip design and compiler shifts toward the compiler as mentioned in the report on Berkeley's BRASS project [5]. Second, to compete with FPGAs, the compiler must be convenient to use, while still achieving high area efficiency through the dynamic reuse of resources. While the area efficiency of DRPs depends on the nature of the chip, it largely depends on the compiler's performance. Compiler developers should be able to meet these challenges through a concerted effort with the new chip architecture development.

However, existing DRP compiles do not adequately meet these challenges. Some use schematic entry, one uses its own proprietary language at the register-transfer-level (RTL), and a limited "C language" can be used only for data path generation at the best [6]. A designer who uses these tools still has to take care of traditional problems such as meeting timing and area constraints, as well as the additional problem of utilizing the resources not only two dimensionally but also vertically (time-multiplexed).

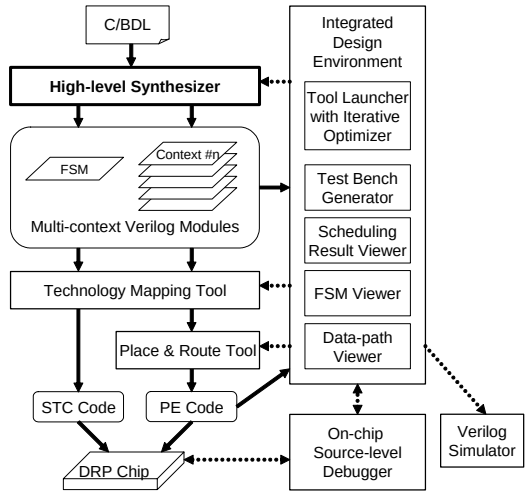

Figure 1. Compilation flow and design environment for DRP.

We have developed an integrated C compiler for the DRP. Like a C compiler for a microprocessor, our compiler includes the entire system environment. As shown in Fig. 1, it includes a high-

ICCAD'06, November 5-9, 2006, San Jose, CA

level synthesizer (HLS), a technology mapping tool, a place and route tool, an on-chip source-level debugger, and an integrated development environment (IDE) with a graphical user interface. In this paper, we mainly focus on area efficiency and complying timing requirement on the HLS but not on a special technique to gain performance.

Comparing multi-context type DRPs to single context type reconfigurable devices such as FPGAs, although these two architectures seem to be different, the performance would be not so different for two reasons. At first, custom arithmetic logic operations are only slightly faster than look-up table (LUT) based arithmetic operation with dedicated architectural features of FPGAs [7]. Secondly and more importantly, wire delay is dominant factor in this type of reconfigurable device. FPGAs and DRPs have the same wiring structure which has many switches between segmented wires. If there is no limitation on the area and using the same process technology, the performance of the same circuit design would not be the same but not quite different. Their performances mainly depend on parallelizing techniques used in a circuit design. These techniques are generally applicable to any wired logic devices including FPGAs and DRPs.

Although we don't focus on parallelizing technique, we can not neglect the timing issue. It is difficult to predict the delay at HLS level for fine grained architectures such as FPGAs. The delay on a critical path mainly depends on the level of look-up table but its level is subject to change in logic optimization. On the other hand, a coarse grained architecture has a chance to control the delay more accurately. In order to reduce the size of configuration memory, all the architectures [1]-[4] introduced here are coarse grained although their granularities depend on the application they focus on. The level of PE in HLS can be constrained since few logic optimization techniques can be applied. There are still error factors in placement and routing needless to add.

The main characteristic of the DRPs is its high area efficiency which is achieved by switching context dynamically. The compiler must fill the context not only in two spatial dimensions but also vertically (time-multiplexed). Ideally, all the resources are equally used in contexts. But in most cases, it is difficult to balance them. We solve this problem by combining contexts to maximize resources at any one context without exceeding allowable maximum for that context.

2. Dynamically reconfigurable processor

A DRP is a coarse-grained, multi-context, reconfigurable core that can be integrated with an application-specific integrated circuit (ASIC). Most other reconfigurable architectures, including the FPGA, use data path to synthesize a sequencer [8], making it difficult to control the sequencer from outside the chip. In some DRP architectures, an embedded CPU is used instead of a sequencer [2], but a CPU is too slow to handle state transition of the data paths. One architectural characteristic of our DRP [4] is that the sequencer is not synthesized using the data path; it is a stand-alone unit. It is thus fast enough and is controllable thorough a bus. Many processing elements (PEs) and memory units are arranged on the DRP. Both the operations to perform and the wires to use between the PEs and other resources, such as the on-chip memories and external ports, are selected based on configuration codes stored in each PE. The configuration is selected within one clock cycle (less than a nanosecond) by the sequencer, which is called a "state transition controller (STC)".

A primitive DRP unit is called a "tile", and a DRP core consists of an arbitrary number of tiles. In our prototype "DRP-1", there are eight tiles on the chip. As shown in Fig. 2, each tile consists of 64 PEs, an STC, and one- or two-port on-chip embedded memory units around the edge. A VMEM, for example, is an 8-bit, 256-word synchronized memory with one data input and two data outputs. Both the bit width and the depth can be expanded by using four units without occupying any PEs. The STC, located at the center of the tile, controls both the context and state transition based on an internal state transition table. Our DRP can execute multiple processes concurrently up to the number of tiles.

As mentioned in the introduction, in order to reduce the size of configuration memory, our DRP is 1/8 bit granularity architecture. As shown in Fig. 3, each PE has an 8-bit arithmetic logic unit (ALU), a data manipulation unit (DMU) for both 8-bit shift/mask operations and 1-bit logic operations, an 8-bit register file unit (RFU), an 8-bit flip-flop unit (FFU), and wire switches. Up to 16 different configuration codes are stored on-chip. Additional codes can be downloaded on-the-fly from external memory.

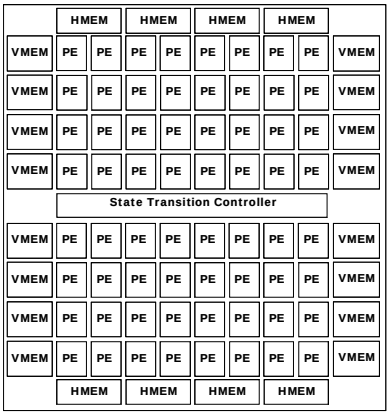

Figure 2. Structure of tile in DRP-1.

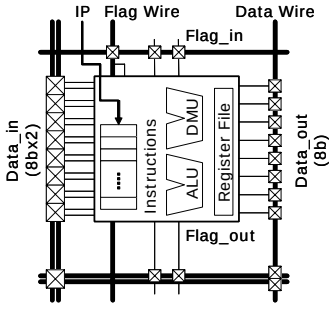

Figure 3. PE architecture.

3. High-level synthesis for DRP

As silicon process technology progresses, it has become challenging to design the complex logic circuits. High-level synthesis has attracted much attention for handling the complex design of reconfigurable chips [9][10][11]. Compared to a RTL design, there are several advantages to use C-based HLS in the compilation flow for the DRP. Designers can write a code more efficiently by using a higher abstraction level language. When a C compiler for the CPU is used during design verification flow, behavioral level simulation is faster than RTL simulation. The C-source code can be debugged functionally using a sophisticated

source-level debugger for a CPU. It is more important for reconfigurable chips to shorten the design turn-around time than ASICs which takes a longer time to be fabricated.

We developed a C-based HLS as a front-end tool for the compilation flow. It is based on our proprietary HLS for ASICs [12][13]. The tool extracts instances of parallelism by generating a control data-flow graph (CDFG) that splits up the description of each step based on given constraints. Unlike microprocessors, the clock frequency of the DRP varies since each resource has its own delay, and the PEs can be chained without inserting a register or memory. Therefore, HLS with an automatic scheduler is useful for controlling delays on the data paths.

Moreover, HLS usually extracts both data path and finite state machine (FSM) from the description. This corresponds to the DRP architecture in which the data paths and STC are handled separately. Right from the start, we designed the DRP architecture with this C-based HLS in mind.

3.1. Allocating multiple contexts

Our DRP compiler inputs C, or behavioral design language (BDL), and outputs downloadable configuration code (STC Code and PE Code shown in Fig. 1). The BDL is a subset/superset of standard C language. For example, hardware-specific notation, input and output port declarations, bit-level extraction, and concatenation are all BDL extensions. There are some restrictions which are difficult to realize on hardware such as recursive call and dynamic memory allocation. Some types of pointer are supported if they are statically analyzable.

Our HLS for the DRP is a front-end tool that generates "multi-context Verilog", in which the contexts are divided into separate modules. Although the generated Verilog code cannot be synthesized using generic logic synthesizers, it can be simulated using an RTL simulator. Figure 4 shows the communication path between the data-flow graph and the DRP resources. A finite state machine (FSM) generated by the HLS is mapped onto the STC. Basically, the steps have a one-to-one relationship with the contexts. We can treat the context switching mechanism as state transition, because the STC is fast enough. However, a function that combines multiple steps into a single context obtains better area efficiency, as described below.

An operator is mapped onto either the ALU or DMU as an instruction for that unit. A register is mapped onto either the RFU or FFU. An array is mapped to either the embedded memory unit (VMEM, HMEM) or the off-chip memory through an embedded memory controller, which is automatically selected based on its depth.

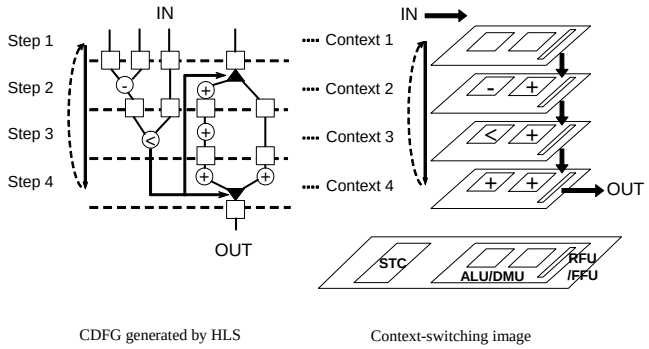

Figure 4. Basic relationships between steps and contexts.

3.2. High-level synthesis flow

A block diagram of our HLS is shown on the left in Fig. 5. In order to shorten synthesis time, the flow is straight forward. The C/BDL description is translated into a tree-structured control flow graph (tCFG). Some optimizations, such as constant propagation, common sub-expression elimination, loop unrolling, in-line procedure expansion, and dead code elimination, are applied to the tCFG. A CDFG is generated from the tCFG. Scheduling, data path allocation, module binding, and control synthesis are processed using the CDFG. A condition vector list scheduling (CVLS) algorithm is used to efficiently allocate resource under the given constraints [14][15]. Other optimizations, such as redundancy elimination, are applied to the RTL tCFG. Finally, the internal format is translated into an RTL language.

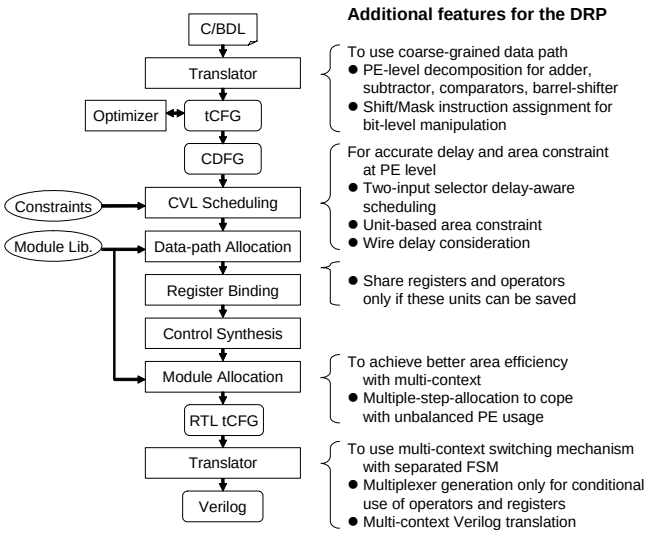

Figure 5. Flow of HLS and additional features for DRP.

4. New challenges in HLS

Some otherwise unimportant issues for ASICs could become critical problems for our DRP. Still, some problems must be anticipated. Table 1 summarizes the differences between HLS for ASICs and for the DRP.

4.1. Context compaction technique

Circuit-size variance between steps is one potential problem. While this is not an issue for ASIC designs, the variance in the number of PEs used among steps becomes a problem in the DRP, especially if we assume a one-to-one relationship between steps and contexts. This assumption increases the number of unused PEs in the context. We implemented a synthesis technique called multiple-step-allocation to achieve higher area efficiency by utilizing the benefit of having multiple contexts.

If a specified description is scheduled using only a delay constraint, the number of PEs used in the step could vary. If the upper limit of the number of the PEs is constrained in the step, an operation that exceeds the limit slides to the next step. However, the step might be changeable without reaching the limit. For instance, since only one access in a clock cycle for each port is permitted using synchronous memory, if any output data of an array has a data dependency on the same memory's input address, the step is changed because of the data dependency, even though

Table 1. Differences between HLS for ASICs and for the DRP.

	For ASICs	For the DRP
Data path	Single context	Multiple contexts (Must average the number of operational unit)
FSM	Wired logic (with variable delay)	STC code (with fixed delay)
Operator / Register sharing	By using multiplexer (Its delay is negligible. Must do scheduling and sharing simultaneously)	Basically no sharing (Share only if unit is saved)
Conditional expression		By using selector instruction in either ALU or DMU (Must consider delay)
Operand and wires (Granularity)	Any bits (Fine-grained)	8-bit: 2 / 3 input and 1 output, 1-bit: 1 / 3 input and 1 output (Coarse-grained)
Bit concatenation/ extraction	Wire connection	Shift/Mask instruction (Must consider delay)
Area constraint	Based on number of operators or memory elements	Based on number of units (ALU / DMU / VMEM / HMEM)

it does not push the limits of any of the constraints. In this case, the context might only have a few memory accesses, and most of the PEs are not used. There are several causes of state transition: (in order of descending priority) 1) control statements including loops, such as "while" or "goto"; 2) access to synchronous resources, such as memory or I/O ports; and 3) delays and area constraints.

The multiple-step-allocation technique helps reduce the number of contexts, in which several steps are combined based on the number of PEs in the step and other constraints. Similar to single-context devices, this technique uses a multiplexer for sharing resources, such as the register units, memory units, and the ports between the steps. A step activation signal for the multiplexer is sent from the STC, in which a table of the relationships between steps and contexts is stored. This technique enables the DRP to store more steps than the number of contexts, which is limited by the size of the configuration memory in a PE. Special care must be taken not to increase the delay of the circuit when inserting the multiplexer. It should not be inserted into a critical path. Other steps that are not on the longest path can be combined unless doing so creates a path with a delay that exceeds that of the critical path. Of course, the number of PEs has to be within available PEs in the context. Therefore, the number of multiplexers is estimated each time when combining steps into various possible combinations. To get the initial data path information, this technique is applied to the translator for the RTL tCFG after the shared resources are allocated (Fig.5).

There are two basic methods for combining the steps: 1) combine consecutive steps and change the context when the constraint limits are hit; 2) combine as many steps as possible until limits are hit, regardless of whether it is contiguous. The first method can reduce the amount of context switching, which consumes one third to half of the power for this kind of DRP. The second provides better area efficiency and better reduces the number of unused PEs than the first by optimizing the step selection process.

4.2. Wire delays and granularity consideration

In general, the wire delays of reconfigurable devices are much longer than the operational delays, because they include both metal wire delays and buffer and tri-state route-switching delays. Wire delays can account for up to 60% of the overall design delays in FPGA [16] and are a dominant factor in the performance of a DRP. Even in HLS for ASICs, wire delays cannot be ignored, especially for the deeper submicron processes. However, accurate wire delays can only be obtained after the place and route tool has finished the static timing analysis. Layout-driven scheduling-binding synthesis reportedly can avoid time consuming iterations of the design process [17]. IPFlex [2] avoids this issue architecturally by guaranteeing the worst-case delay between two operators in any location and prohibiting PE chaining. By utilizing coarse-grained data paths, our HLS for the DRP is able to control the delay of the circuit at the PE level.

Since wire delays are dominant, we focused on them. The typical wire delay between PEs was added to each operational delay. It was calculated based on previous experimental measurement using the place and route tool. Other pre-measured delays were added to the setup and delay of the embedded memory units and the delay of external ports because their placements on the chip are more restricted than those of the PEs.

PE-level decomposition during the pre-scheduling translation stage takes into account the course grain architecture of the DRP. Although the DRP basically has an 8-bit granularity architecture, both the ALU and DMU have 1-bit inputs and output. They are used for carry-in and carry-out flags, comparison result flags, logical operations, etc. A 32-bit adder, for instance, is decomposed into four cascaded 8-bit adders, each of which has the wire delay. Without this decomposition, the original 32-bit adder, which has longer wire delays for four wire legs cannot be scheduled unless it fits within the timing constraint. This decomposition is applied to C-operators, which are decomposed into more than one PE level during the pre-scheduling translation stage. These operators include an adder, a subtractor, comparators, and a barrel-shifter. With this decomposition, the scheduler can control the delay more precisely at each PE level, reducing the number of steps required.

Care must be taken when there is bit-level extraction or concatenation. Since these operations become the wiring connection in an ASIC, they do not have to be considered in the scheduler. The DRP needs a shift/mask instruction in the DMU, and the delay and area for this instruction must be carefully considered. In our HLS for the DRP, bit-level extraction or concatenation that does not align to one or eight bits is transformed into a shift/mask instruction during the pre-scheduling translation stage; this instruction is then treated the same as any other instruction by the scheduler.

4.3. Multiplexer delays in context switching

HLS for single-context devices, such as ASICs or FPGAs, treats the data path and FSM separately. A generated RTL FSM is merged into the data path using a logic synthesizer. On the other hand, multiple contexts must be considered in our HLS for the DRP, in which the STC is a stand-alone module that handles context transition and is outside the PE array. This separation enables the DRP scheduler to treat multiplexing with a completely new approach.

There are two types of multiplexers: those for resource sharing among steps and those for conditional branching. The two types are usually merged for a single-context device. But this can prevent accurate timing estimation in the scheduler because the number of inputs for a multiplexer is difficult to predict from only the CDFG, especially if resources are shared among several steps. Operator sharing is an effective technique for optimizing the data

path. Therefore, except for the primitive logical operators which are smaller than a multiplexer, operators are shared using the CVLS algorithm in our HLS for ASICs. Although the delay of a multiplexer with only few inputs is negligible in an ASIC, the scheduling of such multiplexers is difficult because the resources are actually shared among the steps during the "Data-path Allocation" or "Register Binding" stage (see Fig. 5, 6 and (a)).

For the DRP, a multiplexer for sharing resources among steps is hidden into the fast context switching mechanism with a fixed delay. Because both the RFU and FFU have their own write-enable flags as a configuration bit, a multiplexer for preserving the value of a flip-flop for more than one step is not necessary. The context is switched by changing both the wire connection and the operational code without using a selector instruction (SEL) in either the ALU or DMU. The SEL is conventionally used for selecting signals conditionally in a step.

In our HLS for the DRP, a SEL is treated the same as the other instructions. For example, although the operational delay of a SEL and an adder are not same, the delay for each PE level differs slightly since wire delay is the dominant factor in a reconfigurable device. From the perspective of PE usage, both of a SEL and adder are the same ALU instruction, although their actual areas on the silicon differ. For these reasons, the operators are shared in a context only if both the number and the level of the PE do not increase (see Fig. 6 and (b)). Eventually, only a SEL for conditional use of an operational unit, register unit, memory unit or port in a step must be scheduled. This helps the scheduler estimate the conditional delay more precisely. A SEL placed before either an operational unit or register unit is depicted as a node in the CDFG. Our HLS for the DRP treats this selector node as an operator node which has a delay (see Fig. 6 and (c)). A selector node with only one input can be ignored for a register or operator but is necessary for an external port or memory port. These ports are depicted as more than one node in a step on the CDFG when they are conditionally used. For these types of resources, we add the SEL delay into the node itself. We must carefully treat a SEL for the data input port of a memory unit, because the unit has a write or read enabling signal, which also works as a selector.

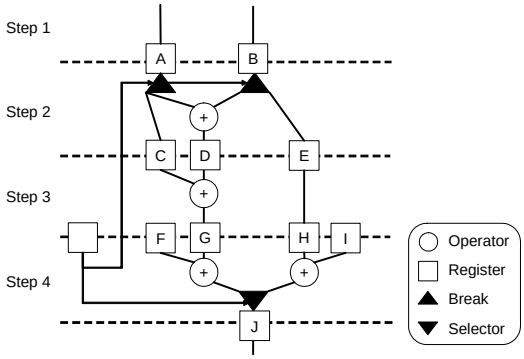

Figure 6. Example illustration of resource sharing.

(a) If the adders in steps 2 and 3 are shared, assuming registers A and C are the same, a two-input multiplexer is needed only if both registers B and D are not shared. If all adders including the two in step 4 are shared, a multiplexer with more input may be needed. However, the number of inputs is difficult to predict during the scheduling stage because registers must be optimally assigned simultaneously.

(b) In our DRP, all the adders are not shared. The fast context switching mechanism eliminates the need for a multiplexer for sharing resources among steps. Because the two adders in step 4 are exclusively used, they should be shared only if registers F and I are the same. Although this sharing does not reduce the PE level because it simply moves the multiplexer to be inserted from the input of the adder to the output, the number of operational units is reduced from three (two adders and one selector to select the adders) to two (one adder and one selector to select register G or H).

(c) If the total delay of both a selector node and the adders in step 4 is more than the designed delay, the selector node is moved to step 5, like the other operator nodes.

This new multiplexer handling also simplifies the work of the place and route tool. Because most of operational instructions are not shared in a context or among contexts, there is no restriction on the multiplexer location. Essentially, this means the operational instructions including the SELs are bound at the placement. Therefore, a faster data path with fewer wire connections can be synthesized.

4.4. Unit-based area constraint

An area-constrained algorithm is a suitable synthesis algorithm for the DRP for three reasons. Many pairs of the ALUs and DMUs are spread over the tiles of the chip, a DRP core consists of an arbitrary number of tiles, and the DRP is designed to get the maximum performance by using these operational units as much as possible but within their limited number.

The area constraint in our HLS for ASICs is given by the numbers of operators. If the number exceeds the constraint in a step, the data flow is divided into multiple steps. Basically, the designer sets the number of operators based on the size of the chip, the operator library for a specific process technology, and the design properties. For the DRP, because all the operators become instructions in either the ALU or DMU, the total number of either one is taken as an upper bound in the scheduler.

This unit-based constraint is also applied to the memory unit. The constraint in our HLS for the DRP is given by the number of the memory units, such as the HMEM or VMEM. The memory specifications, such as the number of ports, setup time, data delay, and total number of units, are specified in a memory constraint file provided by the compiler system.

5. Evaluation

We evaluated the performance of our HLS for the DRP by using a Viterbi decoder. Up to 16 add, compare, and select (ACS) operations can be executed in parallel from this code. The other specifications were a constraint length (K) of 9 and a code rate (R) of 1/2. It should be noted that we had compared the performance to CPUs and DSPs on several stream applications (including Viterbi decoder but not from the same source code) [18] and several encryption algorithms [19].

5.1. Effect of context compaction with area constraint

The numbers of operational units allocated in each context using the following four allocation rules were counted and are shown in Fig. 7. Because two operational units, the ALU and DMU, comprise a PE, we took the larger of these counts as a result. The circuit delay was not constrained.

- Single-step-allocation (SSA)
  Allocating following the single-step to single-context rule results in contexts 8, 9, and 10 having many operational units since the ACS operations are executed in these steps. The other contexts use only a few operators.
- Multiple-step-allocation (MSA)
  Allocating following the multiple-step to single-context rule results in a reduction in the total number of contexts to approximately half compared to the single-step-allocation rule although the total number of operational units slightly increases because selectors are inserted to share the resources. This rule produces the lowest context usage. DRP-1, our prototype, has two restrictions: 1) a context can collect up to four steps; and 2) up to four event signals per context can be fed back to the STC.
- Single-step-allocation with operational unit constraints (SSA+OCs)
  Limiting the number of operational units to 128 (the number of units in two tiles) per step under the single-step to single-context rule increases the number of contexts related to the ACS calculation (the context 8 to 12), compared to the above two methods, because the contexts hit the constraints although the peaks are diminished.
- Multiple-step-allocation with operational unit constraints (MSA+OCs)
  Limiting the number of units to 128 per context under the multiple-step to single-context rule averages the numbers of operational units without increasing the number of contexts. This rule offers the best from the stand point of "volume efficiency", as described below.

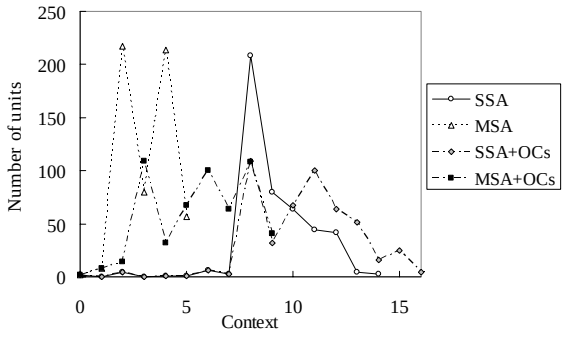

Figure 7. Number of operational units assigned in each Viterbi decoder context.

5.2. Area efficiency

Two criteria, context and area, are used to evaluate "area efficiency" for multi-context devices. Therefore, it should be renamed "volume efficiency" for this type of architecture.

To evaluate the previous context, we defined context usage rate $C_{rate} = C / 16$, where C is the number of contexts used and 16 is the number of contexts that can be stored in the configuration memory on the DRP-1. If C_{rate} exceeds 1, all the contexts cannot be stored simultaneously. Since the unused contexts are available for future extensions or for other applications, a lower rate is better. The MSA rule produced the lowest context usage rate (Fig. 8).

The area is primarily resolved using the maximum number of PEs in the context. Let the number of operational units allocated in context i be OPE_i. The filling rates of operational unit OPE_{rate} are defined as follows and are shown on the right in Fig. 8.

$$OPE_{rate} = \frac{\sum_{i=0}^{C} OPE_i}{C \cdot \max\{OPE_i\}}.$$

A higher filling rate is better for efficient area usage. Combining the multiple-step-allocation and the operational unit constraint (MSA+OCs) offers the best from the standpoint of "volume efficiency".

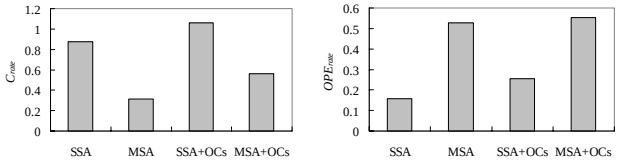

Figure 8. Context usage rate (left) and operational unit filling rates (right).

5.3. Estimated delay and throughput

Figure 9 shows the relationship between the delay estimated by the HLS and the delay obtained by static timing analysis in the place and route (P&R) tool and the delay constraint. The close agreement between the results indicates that our method works well. The saturation of the HLS curve indicates that the scheduler can no longer constrain the delay. The generated circuit is fully chained after the saturation point.

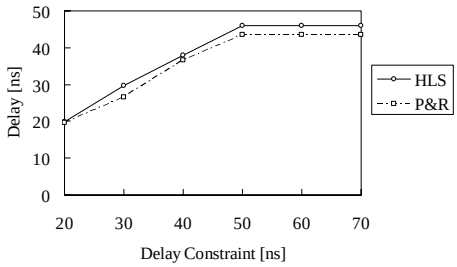

Figure 9. Delay vs. delay constraint.

The circuit throughput depends on both the delay and the number of cycles it takes to execute a test data. The cycle is determined by the number of scheduled steps and by the sequence of FSM generated by the HLS. The relationship between the delay constraint and circuit throughput is shown in Fig. 10. Thanks to the special care taken when inserting the multiplexer, the multiple-step-allocation (MSA) did not affect the throughput of the circuit. As shown in Fig.10, a delay constraint of around 30 ns offers the best throughput. Although the delay is decreased in the delay constraint of 20ns, the number of cycle is increased more.

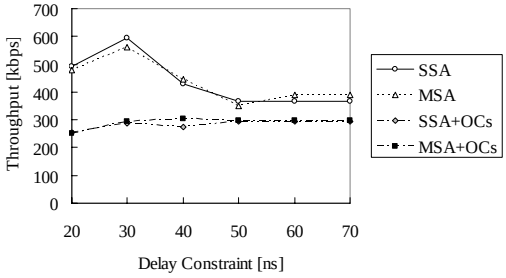

Figure 10. Throughput vs. delay constraint.

Because of the limitations on parallelizing the design with the operational unit constraints (OCs), the throughputs are lower than without them. The number of operational unit should not be constrained if the performance is critical. It can be increased to more than the peak of the units with single-step-allocation rule if number of context is critical.

The accurate delay estimation of our HLS enables the optimal throughput to be found more quickly without running the P&R tool. This is especially valuable when trying to identify the best optimization options and constraint combination in a reasonable amount of time. Since there are many combinations of the numbers to be constrained, there is an iterative optimization function in the integrated development environment (see Fig. 1) to support a designer. It automatically tries all combination of constrains, which are the number of operational unit and the delay, in given ranges. Our HLS automatically also tries many compiler options such as the multiple-step-allocation, a loop unrolling and a function inlining with the support of the iterative optimization function. Along with design productivity, this is one of the best key advantages of HLS compared to design in RTL language. Moreover, it fits well with the nature of the DRP, which offers a trade-off between the best throughput (by increasing circuit parallelization) and the minimum area (by dynamically switching the contexts).

6. Summary and future work

We have developed a C-based compiler for our dynamically reconfigurable processor (DRP), using C language in all phases of development; algorithmic-level optimization, circuit optimization considering hardware architecture, functional-level simulation, and on-chip debugging. Using the short turn-around-time tool set, the designer can take full advantage of the reconfigurable processor, for example, by starting a design before the functional specifications have been fixed or by designing using one or a few developers. By making use of multi-context and a coarse-grained data path of the DRP architecture, our high-level-synthesizer can control delay at processing element level and constrain the number of elements properly. Although the techniques introduced in this paper are primarily for our DRP, most of them can be applied to other coarse grained reconfigurable processors.

Because the compiler which converts a high-level description into logic-level coding can deal with only a limited range of parallelization, a source code optimization process is needed. To overcome the disadvantages of adopting a high-level language, more parallelizing techniques are needed to expand the synthesis search space.

References

[1] K. Bondalapati, "Parallelizing DSP nested loops on reconfigurable architectures using data context switching", Proc. 38th DAC, pp. 273–276, 2001.

[2] T. Sato, H. Watanabe, and K. Shibata, "Implementation of dynamically reconfigurable processor DAPDNA-2", VLSI Design, Automation and Test, pp.323-324, April 2005.

[3] M. Vorbach and R. Becker, "Reconfigurable processor architectures for mobile phones", Proc. Parallel and Distributed Processing Symposium, pp.22-26, 2003.

[4] M. Motomura, "A Dynamically Reconfigurable Processor Architecture", Microprocessor Forum, Oct. 2002.

[5] A. DeHon and J. Wawrzynek, "Reconfigurable Computing: What, Why, and Design Automation Requirements?", Proc. 36th DAC, pp. 610–615, Jun. 1999.

[6] J.P. Cardoso and M. Weinhardt, "Fast and Guaranteed C Compilation onto the PACT-XPP Reconfigurable Computing Platform", Proc. 10th FCCM, 2002.

[7] S. M. Trimberger, "Field-Programmable Gate Array Technology", Kluwer Academic Publisher, p.10, 1994.

[8] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor", Proc. 5th FCCM, pp. 12–21, 1997.

[9] T.J. Callahan, J.R. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler", IEEE Computer, Vol. 33, No. 4, pp. 62–69, Apr. 2000.

[10] Xilinx Forge, http://www.xilinx.com.

[11] C. Sullivan et al., "Deterministic hardware synthesis for compiling high-level descriptions to heterogeneous reconfigurable architectures", Proc. 38th HICSS, 2005.

[12] K. Wakabayashi, "C-based synthesis experiences with a behavior synthesizer, "Cyber"", Proc. DATE 1999, pp. 390–393, March 1999.

[13] K. Wakabayashi, and T. Okamoto, "C-based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective", IEEE Trans. on CAD, Vol.19, Issue 12, pp. 1507–1522, Dec. 2000.

[14] K. Wakabayashi and T. Yoshimura, "A resource sharing and control synthesis method for conditional branches", ICCAD-89, pp. 62–65, Nov. 1989.

[15] K. Wakabayashi and H. Tanaka, "Global scheduling independent of control dependencies based on condition vectors", Proc. 29th DAC, pp. 112–115, Jun.1992.

[16] M. Xu and F.J. Kurdahi, "Area and timing estimation for lookup table based FPGAs", Proc. 1996 ED&TC, pp. 151–157, Mar. 1996.

[17] M. Xu and F. J. Kurdahi, "Layout-driven high level synthesis for FPGA based architectures", Proc. 1998 DATE, pp. 446–450, 1998.

[18] N. Suzuki, et al., "Implementing and Evaluating Stream Applications on the Dynamically Reconfigurable Processor", 12th Annual IEEE Symposium on FCCM, pp. 328–329, Apr. 2004.

[19] M. Suzuki, et al., "Stream Application on the Dynamically Reconfigurable Processor", Proc. 2004 FPT, pp. 137–144, Dec. 2004.