A Hardware/Software Codesign Method for a General Purpose Reconfigurable Co-Processor

Shinji Kimura†, Mitsuteru Yukishita‡, Yasufumi Itou†, Akira Nagoya‡, Makoto Hirao†, Katumasa Watanabe†

†: Nara Institute of Science and technology ‡: NTT Communication Science Laboratory

{kimura, yasufu-i, watanabe}@is.aist-nara.ac.jp
{yuki, nagoya}@cslab.kecl.ntt.co.jp

Abstract

This paper shows a hardware/software codesign method for a computer system with a reconfigurable co-processor. The reconfigurable co-processor is constructed from FPGA's, internal cache and a control part, and is connected to the system bus of the computer system. This paper shows the architecture of the reconfigurable co-processor, a hardware/software separation method and a co-operation method via the DMA based memory sharing. We also show co-operation examples and the effectiveness of our approach for the fast execution of user processes.

1 Introduction

With recent development of hardware technology, reconfigurable hardware devices have been devised. Especially, Field Programmable Gate Arrays (FPGA's) are widely used in logic emulators, rapid hardware prototyping, etc.

Several researches have been studied on computer systems including reconfigurable hardware for speed-up of applications [1], [2], [3], [4], [5]. In these researches, the data communication line between a main processor (a central processing unit in a computer system) and a reconfigurable hardware is rather narrow, and a reconfigurable hardware devices are used for operations with little data transfer, or all data are transmitted to the hardware device and operations are executed off-line.

In the paper, we propose a novel reconfigurable co-processor architecture, a method of the co-operation between the main processor and the reconfigurable co-processor, and a hardware/software co-operation method for the computer system with the reconfigurable co-processor.

The co-processor is based on the bus of the computer system for the accessibility to the main memory and other peripherals on the co-operation. On the memory access, the co-processor uses the direct memory access and includes cache/local memory to escape the bus bottleneck.

We have implemented and tested the co-processor for SUN SBus, and have shown the effectiveness of our codesign method.

2 Bus Based Reconfigurable Co-Processor Architecture

2.1 Co-Processor Architecture

The architecture of the reconfigurable general purpose co-processor is shown in the section. The co-processor architecture has three main parts as shown in Fig. 1.

- the internal bus with 50 MHz clock,
- the reconfigurable FPGA's connected to the internal bus,
- the external bus interface which manages the connection between the internal and the external buses.

The internal bus has 32-bit data lines, 32-bit address lines, and 10-bit control lines. We have designed the co-processor for SUN SBus, and the internal bus specification is the same as the external SBus specification. The following control signals are used on SUN SBus: the request and the acknowledge signals for the bus access, the number of data bytes, and the memory access type.

Hardware modules implemented in FPGA's can access to the external bus via the internal bus, and are designed to be executed under the internal bus clock. Note that the hardware modules can access to the main memory and other peripherals connected to the external bus.

0-8186-7895-X/97 $10.00 © 1997 IEEE

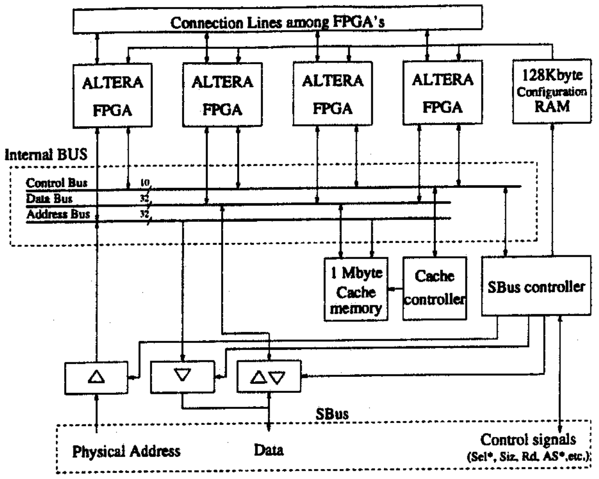

Figure 1: The structure of a general purpose co-processor for SUN SBus.

Table 1: Access time for the cache memory

Access type	# CLK	Exec. time
local read/write	2	40 *nsec*
cache hit read/write	3	60 *nsec*
cache miss read (4byte)	17	340 *nsec*
cache miss read (32byte)	36	720 *nsec*
non-cache read (read through)	15	300 *nsec*
non-cache write (write through)	3	60 *nsec*

Since the co-processor is connected to the bus, the main processor can access to the co-processor via the bus address, which is usually called memory mapped I/O. We can define control behaviors, such as the reset of the co-processor, the configuration of FPGA, etc., for specific bus addresses.

The co-processor has 1 MByte of 2 way associative cache/local memory, where we use SRAM with 17 nsec access time. When the cache is used as the local memory, the access needs 2 global clocks (with 50 MHz clock). Cache hit access needs 3 clocks, and cache non-hit access needs 20-40 clocks depending the external bus status. Access times are shown in Table 1.

2.2 Structure of Reconfigurable Part

The reconfigurable part of the co-processor is made from 4 EPF81188 FPGA's. EPF81188 is a LUT-type FPGA, includes 1008 logic cells, 180 I/O pins, 1180 flip-flops. Each logic cell in EPF81188 can implement any logic function with up to 4 inputs and 1 output, and EPF81188 can implement circuits up to 12,000 gates. EPF81188 can execute 24-bit accumulator with 87 MHz and 16-bit up/down-counter with 125 MHz.

FPGA's are connected to each other with internal connection lines. Each FPGA has 180 I/O pins. To maximize the number of connection lines, the address line of the internal bus is connected only one FPGA. Thus 32 (address or data) + 10 (control) I/O pins in each FPGA are used for the connection to the internal bus and 45 I/O pins are used for the internal connection among FPGA's. The network among FPGA's is symmetric, and we can use each FPGA in symmetric manner.

We have designed a library for basic operations specific to Altera FPGA's. The library includes an n-bit addition, an n-bit subtraction, an n-bit comparison, and multipliers up to 16-bits. The delay of each modules is estimated as follows:

module (# bits)	delay	# clocks	# LUT's
add ($n \leq 16$)	< 15 ns	1 clocks	n
add (16 ... 32)	< 30 ns	2 clocks	n
shift (use wiring)	< 15 ns	1 clocks	0
comp ($n \leq 16$)	< 15 ns	1 clocks	n
comp (16 ... 32)	< 30 ns	2 clocks	n
W-mult ($n \leq 8$)	< 30 ns	2 clocks	< 64
W-mult (8 ... 16)	< 70 ns	4 clocks	< 240

3 Codesign for the Reconfigurable Co-Processor

3.1 Overview of the Codesign

The co-processor can implement functions in C programs using the bus based architecture and the SBus DVMA (Direct Virtual Memory Access) mechanism. C functions in a user program are implemented in the reconfigurable part of the processor, and the main program on the main processor and the functions on the co-processor are co-operated.

The flow of codesign of hardware and software is as follows:

1. Parsing of C program and Control/Data flow graph generation
2. Hardware module generation for each function
3. Estimation of execution clocks and hardware resources

At first, C program is parsed and block structures are extracted, which is usually called control/data-flow graph. In the control flow-graph, each node represents a data-flow graph. Thus one state is assigned for each node of the control flow graph, and sub-state is assigned to the data flow graph based on the result of the scheduling.

Hardware/software separation is investigated based on C functions. Each C functions are estimated the size of hardware modules and the number of execution clocks. The estimation of hardware size and speed is based on the property of FPGA elements. The execution time is measured as the number of clocks executing the behavior, and the hardware size is measured as the number of LUT's of FPGA.

Functions suitable for hardware execution (fast execution time and reasonable hardware size) are implemented on the FPGA's, and the codesign compiler generates hardware description language such as VHDL or SFL [6]. The number of hardware implemented functions is depend on the total hardware size of the reconfigurable part.

For the functions implemented as hardware modules, C program is converted as follows:

1. the hardware-implemented function is replaced to a command to set the start signal for the co-processor, and
2. the program should monitor the end signal from the co-processor with adequate interval.

Note that the C program can do some operation when the co-processor is working. Thus the re-scheduling for the parallel execution is needed.

The DVMA mechanism on SBus makes it possible to access from the co-processor to the user area via the virtual address of user process. We can map the user area to the DVMA area using the kernel memory map operation. We should set some flags for the user area not to swap-out by the context switch.

At present, we use 256 KByte of user buffer for the communication between the co-processor and the main process. The limit depends on the UNIX kertnel (SUN OS 4.1.3). We should change the buffer window to sweep memory area more than 256 KByte, which is invoked via the signal from the co-processor. Several top bytes of the buffer is used for such control signals to change the buffer window, the end of the work of the co-processor, etc. It takes about 12 msec to change from one buffer to another buffers (256 KBytes each), and it takes about 80 msec to read and then write the data in the 256 KByte buffer from the co-processor.

3.2 Hardware Modules Implementing C Functions

C functions are converted to FPGA modules as follows. The function invocation is executed using the bus access from the main processor. The parameter is passed using also the bus access.

The structure of a hardware module is as follows: the main structure is the 7 state automaton corresponding to the access from the CPU (Fig. 2). At the initial state, the modules waits the access signal

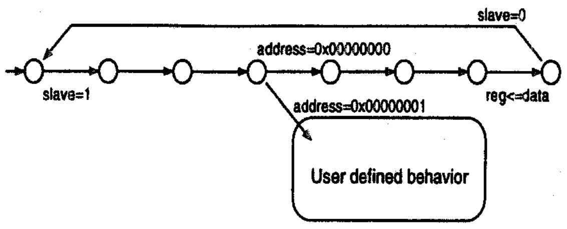

Figure 2: Basic control structure of a hardware module.

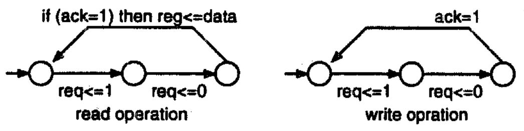

Figure 3: Basic control structure of a hardware module.

from the main processor (slave=1 in the figure), and check the address and do the defined behavior. Usually, parameters of the function is passed from the CPU and then invoke the function via some address. "address" in the figure denotes the bus address and "data" denotes the data passed via the bus. In the example, only one parameter is passed via the address "0x0000000", and the co-processor is invoked via the address "0x00000001".

A hardware module can access to the main memory using the usual request/acknowledge protocol, which can be implemented using the following automaton in Fig. 3. Note that the read/write automaton can be used as a hardware subroutine. Before calling these subroutines, the address (and the data for write) is set.

3.3 Optimization Techniques

3.3.1 hardware Independent Optimization

In the optimization of hardware modules, we tested the necessary bit length of each variable, since the size of hardware modules greatly depend on the bit-length.

The minimum v_m and maximum values v_M are associated to each variable v, and v_m and v_M is modified depending on operations on the variable. The number of bits for the variables is set to $\log(v_M - v_m)$, which may be less than the defined bit-length.

The modification of v_m and v_M is as follows:

- Constant assignment and constant addition
  x = 0; /* $x_m = 0$, $x_M = 0$ */
  x = x + 1; /* $x_m = 0$, $x_M = 1$ */
- Variable assignment
  x = y; /* $x_m = y_m$, $x_M = y_M$ */

- Assertion for the variable
  if (x < 128) /* $x_M = 129$ */

The effect of the reduction is large for loop control variables. The number of bits for flag variables are also reduced using the above method, but that may reduced automatically by logic synthesizer.

3.3.2 Hardware Specific Optimization

FPGA elements may have special hardwares for some operations, such as the addition and the comparison. Altera FPGA has the fast carry chain and the fast cascade chain, and the ripple carry adder is faster than the carry-look-ahead adder. An n-bit adder can be implemented with n LUT modules. These hardware specific properties are settled using hardware module library.

Another point is the high cost of the selector implementing with the LUT module. In the adder implementation, there is no free input variables in LUT modules, and if we set a selector before an adder, there need n extra LUT modules, which is the same amount of LUT's implementing the adder.

Thus we do not use the operation sharing for addition and subtraction. Operation scheduling is very easy since there is no sharing. Next, we count the number of LUT's for the hardware size and for the hardware delay. The delay of LUT module for Altera is 2 nsec, but the wiring delay between two LUT's is about 15 nsec. The depth should be one for control logic which is executed in 50 MHz clock.

Multiplication cost is large and 16 or less bit multipliers can be implemented. Constant multiplication is converted to shift and add operations to reduce the final hardware size. For example, if $x * 10$ is in the C-program, then that is implemented as $x*8+x*2$.

3.4 Module Allocation and Scheduling

On Altera FPGA's, the sharing of operation is not effective, thus the allocation of operational modules is rather simple. A good result can be obtained using the ASAP scheduling.

Before the allocation, operations are rescheduled and balanced on the data-flow graph. In C program, the execution of consecutive additions and subtractions can be rescheduled freely. For example,

```
x = a + b + c;
y = x - z;
w = y + x;
```

are converted to

```
w = 2 * (a + b)  + 2 * c - z;
```

Note that "2 *" is the shift operation. In the case, x and y are eliminated and new registers are inserted, since the number of inputs of an adder is two.

```
tmp1 = a + b;
tmp2 = 2 * c - z;
w = tmp1 + tmp2;
```

The computation of *tmp*1 and *tmp*2 can be executed in parallel.

In the scheduling, small operations are merged and clocked registers are inserted. The clock is 50 MHz and the delay of one LUT and wiring is 15 ns or so, thus the one hot state assignment is used.

Some operations such as 32 bit addition are executed using 2 clocks, which is implemented to control the timing of the assignment to the output register of the operation. Note that the maximum delay of the combinational logic is greater than the clock cycle, but the circuit behaves correctly. We have devised the delay estimation method for such part.

3.5 Evaluation of the Hardware Cost and Performance

In the section, we show an investigation method of the execution time of C functions. As described above, the C program is parsed and the control/data flow graph have been generated.

On the software implementation, the number of clocks to execute a simple block can be obtained as the number of operations in the disassembled compiled code. We assume that there is no pipeline hazard, and all operations are executed with one clock.

On the hardware implementation, the number of clocks to execute a simple block can be obtained as the maximum clocks of the scheduled data flow graph.

The estimation of if-statement and loop-statement is rather complex, since the condition check and the number of iterations are decided at the execution time.

For if-statement, the condition is assumed to be true with 0.5 probability, and the total clock is measured as the half of the addition of the clocks of the true block and that of the false block.

For loop statement, if the loop has the fixed iteration, then the iteration number is used. If the iteration cannot be measured, the iteration number is used as the parameter of the evaluation, and the designer decides whether the hardware implementation is useful or not.

On the memory access, the access time depends on the bus speed, the bus width and the cache hit ratio. On the estimation, the access from the co-processor is considered as 4 times slower than that from the main processor depending the present co-processor architecture.

By summing up these evaluations, we can measure the number of execution clocks for functions with software and hardware. Then we can get the speed-up ration by dividing these clocks.

4 Experimental Results

We have now implemented and tested our co-processor board for SUN SBus and the co-design compiler. In the section, experimental results are shown. We have found that bit-level parallel operations and parallel if-statement can be executed very effective on the co-processor.

4.1 Measuring Examples

We have implemented our algorithms in C and tested for two benchmarks in [9], one is the generation of ECC (Error Correcting Code) for 8 bit data and the other is GCD operation including 5 times iteration.

Data	# clk soft	# clk hard	ratio
ECC	1329	63	21.1
GCD	58	28	6.5

The real execution clocks on the co-processor is as follows:

Data	# clk soft	# clk co-proc	ratio
ECC	1329	46	28.9
GCD	58	49	1.3

The reason of the difference on the GCD execution is that the memory access from the co-processor needs fixed overhead.

4.2 Lexical Analysis

As an example of codesign systems, we have implemented a converter from a specification description of "lex" to a circuit for the co-processor. The specification description is analyzed and converted to a finite automaton, and the the finite automaton is described as a hardware description language.

A program to change 32 key words of C programs to upper case letters is used in the evaluation. When applying the program to the source text of gcc ver.2.5.8, the substitution needs 16.3 seconds on SS10/51. The lexical analysis part of the translation operation is implemented as a circuit in the co-processor and is evaluated with the same manner as shown above. The operation needs only 1.26 seconds. The real action to convert from a key word to the upper case letters needs 0.92 seconds on the main processor. Since the main processor and the co-processor can operate in parallel, total execution time is 1.26 seconds. We obtains about 13 times speed-up.

5 Conclusions

We have described a hardware/software codesign method for the general purpose reconfigurable co-processor. In the hardware module generation, the C function based method and several optimization techniques are shown. We have also shown some experiments for the execution of some functions on the co-processor.

References

[1] N. Suganuma, Y. Murata, S. Nakata, M. Tomita, and K. Hirano. Reconfigurable Machine and Its Application to Logic Diagnosis. In *Proc. of ICCAD*, pp. 538–543, 1992.

[2] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, and D Lopresti. Building and Using a Highly Parallel Programmable Logic Array. *Computer*, Vol. 24, No. 3, pp. 81–89, Jan. 1991.

[3] P. Athanas and H. Silverman. Processor Reconfiguration Through Instruction-Set Metamorphosis. *Computer*, Vol. 26, No. 3, pp. 11–18, 1993.

[4] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh. PRISM-II Compiler and Architecture. *FCCM '93*, pp. 9–16, 1993.

[5] D. Thomas, J. Adams, and H. Schmit. A Model and Methodology for Hardware-Software Codesign. *IEEE Design & Test of Computers*, Vol. 10, No. 3, pp. 6–15, 1993.

[6] High-Level Synthesis Design at NTT Systems Labs. *IEICE Trans. Info. and Systems*, Vol. E76-D No. 9, pp. 1047-1054, 1994.

[7] *Data Book.* Altera Corporation, 1993.

[8] H. Edward, Frank, and J. Lyle. *SBus Specification B.0.* SUN Microsystems, Inc, 1990.

[9] *Benckmarks for the 1991 High Level Synthesis Workshop.* 1991.