Domain-Specific High-Level Modeling and Synthesis for ATM Switch Design Using VHDL

Mike Tien-Chien Lee, Yu-Chin Hsu†, Ben Chen*, and Masahiro Fujita

Fujitsu Laboratories of America
3350 Scott Blvd., Bldg. #34
Santa Clara, CA 95054, USA

†Dept. of Computer Science
Univ. of California
Riverside, CA 92521, USA

*Fujitsu Ltd.
1015, Kamikodanaka Nakahar-Ku
Kawasaki 211, Japan

Abstract: *This paper presents our experience on domain-specific high-level modeling and synthesis for Fujitsu ATM switch design. We propose a high-level design methodology using VHDL, where ATM switch architectural features are considered during behavior modeling, and a high-level synthesis compiler, MEBS, is prototyped to synthesize the behavior model down to a gate-level implementation. Since the specific ATM switch architecture is incorporated into both modeling and synthesis phases, a high-quality design is efficiently derived. The synthesis results show that given the design constraints, the proposed high-level design methodology can produce a gate-level implementation by MEBS with about 15% area reduction in shorter design cycle when compared with manual design.*

1 Introduction

ATM (Asynchronous Transfer Mode) has been considered by telecommunication industry the ultimate solution to the networking requirement for Broadband ISDN [1]. Among the major components of ATM hardware, the core technology is the *ATM switches*, which are capable of sending ATM packets from one end to the other in a communication network when connected in a specific topology. Due to its short product life cycle and various demands from different market segments, current gate or lower level design methodology can no longer satisfy the stringent time-to-market requirement for ATM switch design. It has become necessary to exploit a higher-level methodology which can specify and synthesize the design from an abstraction level higher than logic gates. High-level synthesis [2], which takes an algorithmic specification as input and automatically generates a register-transfer (RT) level architecture, is hence considered by Fujitsu ATM switch designers a promising CAD technology to boost design quality and shorten development cycle.

There exist several successful high-level synthesis tools from system companies such as HIS [3] of IBM and CALLAS [4] of Siemens, and commercial products such as Behavioral Compiler from Synopsys [5, 6]. But these tools may not be easily adapted to Fujitsu internal ATM switch design environment.

Furthermore, in practice, most high-level synthesis tools are not able to automatically derive a high-quality ATM switch architecture if the target synthesis domains are either too general, or too specific for another design style (e.g., DSP filter). Besides, an arbitrary behavioral model without considering important architectural features, such as partitioning and pipelining, is also difficult for general-purpose high-level synthesis to obtain satisfactory results. It has been observed that although the design details of ATM switches may change with the specifications, they share similar control-dominated architectures which contain a simple datapath for buffering/transferring high-speed packets and a complex control path for managing concurrent events in the system. Therefore, it is essential to exploit such architectural features to develop a good behavior model for synthesis, and incorporate these features into high-level synthesis, which can hence focus on a more specific design space to produce high-quality architectures efficiently.

This paper reports our experience on VHDL modeling and synthesis of Fujitsu ATM switch using high-level synthesis methodology. We propose a high-level design methodology by developing a VHDL behavior modeling scheme suitable for the target ATM switch design, and prototyping a domain-specific high-level synthesis compiler to synthesize the developed behavior model. Important VHDL modeling techniques for the ATM switch, such as system partitioning for efficient synthesis, clock rate conversion for multiple clockings, process communication using appropriate protocols, and arbitrator processes to sequence concurrent signal updates, are discussed in the paper.

A VHDL high-level synthesis system, called MEBS, is proposed for domain-specific ATM switch synthesis. MEBS synthesizes the developed mixed-level VHDL model down to a gate-level netlist by performing scheduling, allocation, and RTL synthesis. It provides several features which are very useful for our ATM switch synthesis, such as design space exploration for multiple processes, module function implementation for subprograms, and high-level incremental synthesis. Experimental results show that given the design constraints, the proposed high-level design methodology is able to produce a gate-level implementation using MEBS with about 15% area reduction in shorter design cycle when compared with manual design methodology.

2 ATM Switch Architecture Specification

The system-level architecture of a 2-channel ATM switch is illustrated using the block diagram in Figure 1. Its major function can be stated as follows: a packet of words, called a *cell*, received from one of the two input highways `iHW0` and `iHW1` is stored in the cell FIFO at first, and will be switched to one of the two output highways `oHW0` and `oHW1` according to the routing information in the cell's header field. A cell consists of 27 16-bit words, where the first 3 words are the header field containing information of routing, transmission priority, congesting control, etc., while the remaining words are the payload field for data.

The switch system has two main channels for cell switching, i.e., two input highways (`iHW0`, `iHW1`) and two output highways (`oHW0`, `oHW1`), all operating at a global highway clock rate `HW_clk`. In order to reduce design cost, the internal switch core for cell processing, as shown inside the dotted square in Figure 1, operates at a slower switch clock rate `sw_clk`, which is only a fraction of `HW_clk`. Therefore, each cell arriving at the input port

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA 0-89791-833-9/96/0006..$3.50

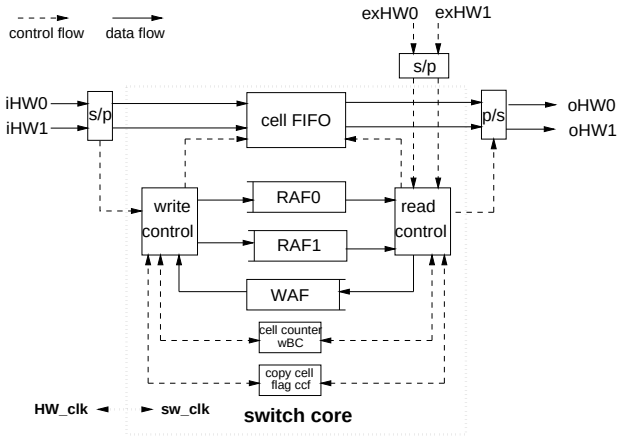

Figure 1: 2-channel ATM switch architecture.

`iHW0` or `iHW1` first takes serial-to-parallel (`S/P`) conversion to reduce to the slower processing rate `sw_clk`. This received cell is then written into a cell FIFO for buffering, and will later be switched to an appropriate output highway based on the routing information contained in the header field of the cell. When a cell is ready to be sent out, parallel-to-serial (`P/S`) conversion is performed at the output port to re-match the faster highway clock rate `HW_clk`.

When the received cell is written into the cell FIFO for buffering, its FIFO address is determined by Write Control `WC` and Write Address FIFO `WAF`. Namely, `WC` obtains from `WAF` the the next available address of the cell FIFO where a new cell can be written. Then, based on the destination of the cell, this address is kept in a Read Address FIFO `RAF0` or `RAF1`, where the switch can later find the address of the cell to be sent out. The address is kept in `RAF0` if the cell is switched to `oHW0`, or in `RAF1` if the cell is switched to `oHW1`, or in both if the cell is a *copy cell* switched for both output highways.

To send a cell from the cell FIFO to the output highway `oHW0` (or `oHW1`), Read Control `RC` is instructed by a read-out signal received from the extended highway `exHW0` (or `exHW1`) to get the top cell address stored in `RAF0` (or `RAF1`). The corresponding cell is then retrieved from the cell FIFO, and sent to `oHW0` (or `oHW1`). After this cell is read out, the used address can be recycled to `WAF`, which is used later to store other cell received at `iHW0` or `iHW1`.

The switch core also has a cell counter `wBC` to monitor the available capacity of the cell FIFO. If the switch attempts to write a cell to the cell FIFO which is full, an alarm signal is sent to notify a central switch controller, which is not included in the switch design, to handle such exception. Furthermore, to keep track if a cell written into the cell FIFO is a copy cell, a 1-bit copy cell flag `ccf` is associated with each location in the cell FIFO.

3 ATM Switch Behavioral Modeling

Developing a behavior model with consideration of target design features is essential to achieving successful high-level synthesis. This section discusses the major issues associated with modeling the ATM switch for high-level synthesis. The behavior model of the entire ATM switch design is then developed, which consists of 26 communicating processes in about 1500-line VHDL codes.

3.1 System Partitioning and Modeling

The ATM switch is a complex design which consists of a number of components operating concurrently for the highways to access and update the cell FIFO, address FIFOs, cell counter, and copy cell flag. To develop a behavioral model good for high-level synthesis, this complex design needs to be partitioned first into subsystem, or modules, to reduce complexity such that each module can be modeled and synthesized efficiently. This has been reported recently as well in a separate study by Vahid *et al.* [7] which shows the advantage of partitioning in terms of faster synthesis time and better overall design quality.

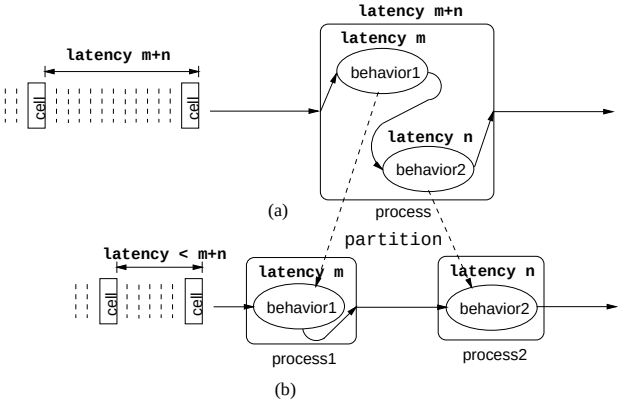

Figure 2: (a) two concurrent behaviors serialized in a single process with total latency $m+n$; (b) partitioned into two processes in a pipeline fashion with latency smaller than $m+n$ (note that process 2 can start even before process 1 finishes).

Each module in the partition can be modeled as a *process* in VHDL. In VHDL, a process executes concurrently with other processes, but only sequential statements are allowed inside a process, which can undergo scheduling and allocation by high-level synthesis. Therefore, to model the entire partitioned system, multiple communicating processes are needed with frequent interactions among them.

Two major criteria are exploited for partitioning the ATM switch system. The first criterion is based on *clocking* (`sw_clk` or `HW_clk`) and *functionality*. One common limitation in current synthesis is that only one single clock signal can be used for each process. So, clocking, as well as functionality of each component as illustrated in Figure 1, are exploited to obtain an initial partition.

The second criterion is based on *pipeline throughput requirement* of the two main channels. In the above initial partition, if modules `WC` and `RC` are each modeled by a single process, the concurrent behavior inside `WC` and `RC` has to be temporally serialized. Such serialization can impose longer delay than the maximal latency constrained by pipeline processing of the cell words in the switch core. This situation is illustrated in Figure 2. So, despite the lack of detailed circuit information for scheduling at this level of abstraction, a coarse partitioning of `WC` and `RC` into pipeline stages can be performed to derive a behavior model with higher pipeline concurrency.

3.2 Multiple Clocking

Different clocks for internal processing (by `sw_clk`) and external transmission (by `HW_clk`) draw a clocking boundary in the design, as shown by the dotted square in Figure 1. On either side of this boundary, processes are designated to model functionalities clocked by either `sw_clk` or `HW_clk`. Therefore, when a cell being processed moves across the clocking boundary, a protocol for clock rate conversion to synchronize operation rates needs to be devised and modeled. Without such mechanism, cell words could be missed or misaligned from

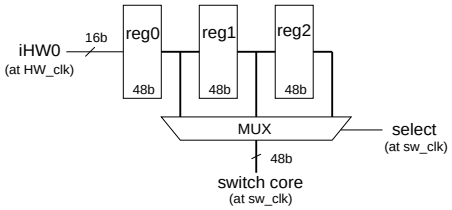

Figure 3: Buffering mechanism for clock rate conversion from `HW_clk` to `sw_clk` (= `HW_clk`/3).

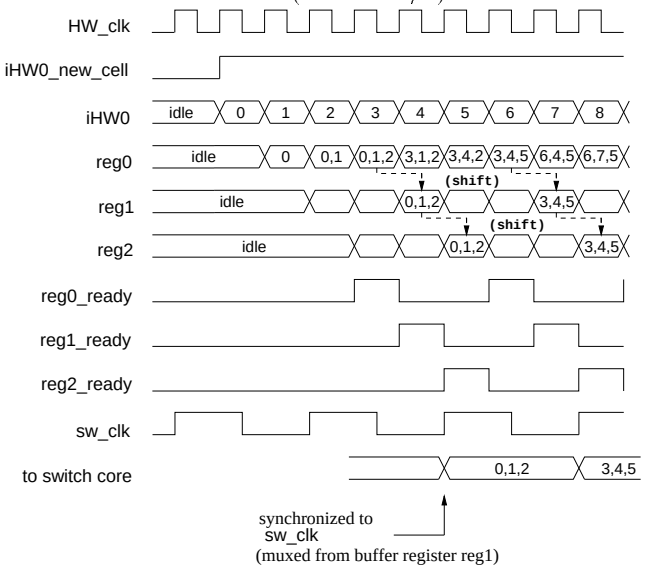

Figure 4: Timing diagram for `HW_clk`-to-`sw_clk` synchronization at `iHW0`.

correct clock cycles. The protocols of clock rate conversions from `HW_clk` to `sw_clk` by `S/P` and from `sw_clk` to `HW_clk` by `P/S` are discussed below. For illustration purpose, we assume that `sw_clk` is clocking at 1/3 the rate of `HW_clk` for the rest of the paper.

3.2.1 `HW_clk`-to-`sw_clk` Conversion by `S/P`

To convert from `HW_clk` to slower `sw_clk`, a mechanism of buffering the cell words received at the input highways is needed such that they can become synchronized to the slower clock rate when they move into the switch core. Figure 3 depicts the schematic of such buffering mechanism, where 3 48-bit buffer registers (`reg0`, `reg1`, `reg2`) are connected in a shift-register manner. Its behavior at `iHW0`, as listed below, is modeled by the three processes `iHW0_SP_reg0`, `iHW0_SP_reg1` and `iHW0_SP_reg2` clocked by `HW_clk`, and the one process `iHW0_to_FIFO` clocked by `sw_clk`.

```
iHW0_SP_reg0: process -- protocol for s/p
begin                -- conversion at reg0 of iHW0
   wait until (HW_clk'event and HW_clk='1');
   reg0_ready <= '0'; -- reset
    if iHW0_new_cell='1' then -- new cell arrives
        -- store 3 words (48 bits) in reg0
        iHW0_reg0(15 downto 0) <= iHW0;
        wait until (HW_clk'event and HW_clk='1');
        iHW0_reg0(31 downto 16) <= iHW0;
        wait until (HW_clk'event and HW_clk='1');
        iHW0_reg0(47 downto 32) <= iHW0;
        reg0_ready <= '1';
    end if;
end process;

iHW0_SP_reg1: process -- protocol for s/p
begin              -- conversion at reg1 of iHW0
   wait until (HW_clk'event and HW_clk='1');
   reg1_ready <= '0';  -- reset
    if reg0_ready='1' then
        iHW0_reg1 <= iHW0_reg0; -- shift to reg1
        reg1_ready <= '1';
    endif;
end process;

iHW0_SP_reg2: process -- protocol for s/p
begin              -- conversion at reg2 of iHW0
   wait until (HW_clk'event and HW_clk='1');
   reg2_ready <= '0'; -- reset
   if reg1_ready='1' then
        iHW0_reg2 <= iHW0_reg1; -- shift to reg2
        reg2_ready <= '1';
   endif;
end process;
```

(a) clocked by HW_clk

```
iHW0_to_FIFO: process -- sync cell words by
begin    -- sw_clk and send them to cell FIFO
    wait until (sw_clk'event and sw_clk='1');
    if reg0_ready='1' then
        cell_3words <= iHW0_reg0;
    elsif reg1_read='1' then
        cell_3words <= iHW0_reg1;
    elsif reg2_read='1' then
        cell_3words <= iHW0_reg2;
    end if;
    -- cell words get sync to sw_clk
    -- ready to send to cell FIFO
end process;
```

(b) clocked by sw_clk

Process `iHW0_SP_reg0` receives a 16-bit cell word of `iHW0` every `HW_clk` cycle when a new cell arrives. This word is packed in `reg0` with other two consecutive words from the same cell to prepare a 48-bit word for internal switch core processing. Once `reg0` is filled with three 16-bit words from `iHW0`, its content is shifted to the next register `reg1` in the next `HW_clk` cycle (by process `iHW0_SP_reg1`), and then shifted to `reg2` (by process `iHW0_SP_reg2`). Since `sw_clk` is one third frequency of `HW_clk`, `sw_clk` must have a rising edge within three consecutive `HW_clk` cycles. This rising edge enables the multiplexer in Figure 3 to select the 48-bit word from the correct buffer register. Such multiplexing is performed by process `iHW0_to_FIFO`. Clock synchronization to `sw_clk` is therefore achieved.

Since the packed 48-bit word is shifted in the three buffer registers, the three signals `reg0_ready`, `reg1_ready`, and `reg2_ready` are needed to keep track in which buffer register the packed word is located upon the rising edge of `sw_clk`. An example of timing diagram to illustrate synchronization during the clock rate conversion is shown in Figure 4, where signal `iHW0_new_cell` is high when `iHW0` is receiving a new cell. In this example, `reg1` is selected by the multiplexer to send the packed 48-bit words to the switch core, because `reg1_ready` is high on the rising edge of `sw_clk`.

3.2.2 `sw_clk`-to-`HW_clk` Conversion by `P/S`

Conversion from `sw_clk` to higher frequency `HW_clk` can be done by sequentially selecting at `HW_clk` rate each of the three 16-bit cell words by a multiplexer.

3.3 Process Communication

The communication between processes in the ATM switch behavior model is described using either *hand-shaking* protocol or *cycle-fixed* protocol:

Communication by hand-shaking occurs between two processes if the protocol schedule cannot be determined statically.

In the case of protocol for `S/P` or `P/S` conversion or for memory access, exact cycle-by-cycle behavior to be

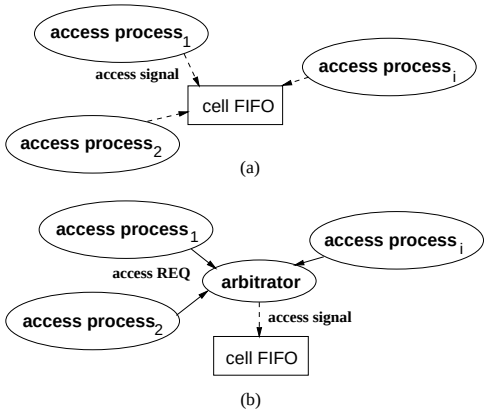

Figure 5: (a) direct concurrent accesses to the cell FIFO causing multiple drivers for the access signals; (b) concurrent access requests sequenced by the arbitrator process to resolve the multiple driver conflict for the access signal.

modeled is predefined by the protocol schedule. Such protocol schedule is called *cycle-fixed* schedule. This cycle-fixed protocol makes the communication simpler.

3.4 Concurrent Update of Global Signal

Memory modules such as the cell FIFO, `RAF0`, `RAF1`, `WAF`, `ccf`, and `wBC` are concurrently updated by global access signals, such as cell FIFO write enable, when the switch receives or transmits cells. In VHDL, however, assignments to a global signal in different processes can cause the conflict of *multiple drivers* for this signal, which needs to be resolved by a *resolution function* such as wired-and, wired-or, or a tri-state gate. Figure 5(a) depicts such conflict due to direct accesses to the cell FIFO by multiple concurrent processes.

For these global memory access signals in the ATM switch model, defining resolution functions is not trivial. So, it is necessary to devise an arbitrator process for each memory module to sequence the concurrent access requests based on some predefined priority. Figure 5(b) shows such a scheme for concurrent accesses to the cell FIFO through an arbitrator process.

3.5 Mixed-Level Model with Subprogram

Since some parts in our ATM switch model have cycle-fixed behavior, as discussed in Section 3.3, the design is best described in a mixed style of *algorithmic* level and *FSM with datapath* level.

Algorithmic description has the highest level of abstraction which specifies purely circuit behavior with no machine states assigned. The design behavior is expressed in VHDL using sequential assignments for data transfer, control constructs for conditional branch, and loop statements for iterative execution. Describing design behavior algorithmically in VHDL is very similar to writing software programs in a standard imperative programming language.

After a schedule is assigned to an algorithmic description, its cycle-by-cycle behavior is determined, where for each cycle a machine state is defined with the datapath operations executed in the cycle. So, after scheduling, the algorithmic description is translated into a finite state machine (FSM) integrated with word-level datapath operations. Such description is an extension of traditional bit-level finite state machine, and is called *FSM with datapath* (FSMD) model. FSMD is suitable to model cycle-fixed behavior with word-level operations, for which bit-level FSM description can be awkward. The VHDL statement `wait until clock'event and clock='1'` is used to define a new machine state, and is referred to as `wait until clock` statement in this paper for short.

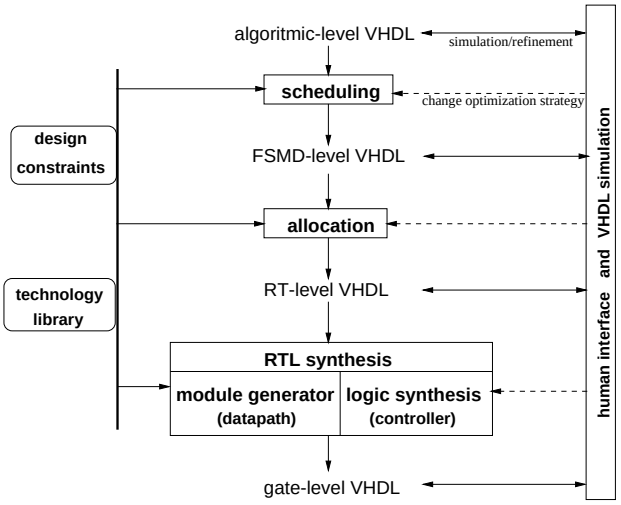

Figure 6: MEBS synthesis flow.

A subprogram is a VHDL construct which defines a sequence of operations that is frequently used in different locations of a program. Subprograms are used intensively in the ATM switch model, such as checking the cell's header field by the processes of the two input highways. So, efficient implementation of a subprogram is essential for our ATM switch synthesis, which will be discussed in Section 4.2

4 High-Level Synthesis for ATM Switch

We prototyped MEBS to synthesize the mixed-level ATM switch description down to a gate-level netlist. MEBS is a high-level synthesis system targeted towards control dominated application. Unlike circuits for DSP application which are datapath dominated, circuits for control dominated application usually contain very simple data path, and perform certain predefined sequences of operations according to protocols or external events from the environment. Most processes in the ATM switch model are control dominated that perform condition test and branch operations, simple arithmetic/logic operations, and I/O operations. Since execution delay of such operations is usually small, operations with dependencies can be chained into a single clock cycle by MEBS.

4.1 MEBS System Component

The synthesis flow of MEBS is depicted in Figure 6, which takes a mixed-level VHDL description and synthesizes it into a gate-level netlist for FPGA or standard-cell implementation. The input design description can contain multiple communicating processes, each of which is at algorithmic level, FSMD level, RT level, or gate level. The major system components of MEBS include a scheduler, an allocator, and an RTL synthesizer. The scheduler converts an algorithmic-level description into an FSMD-level description. Given resource constraints, a list-based scheduling algorithm is used to schedule the algorithmic design into a state machine. Two scheduling modes are supported: super-state mode and cycle-fixed mode. For hand-shaking communication, the operations between two `wait until clock` statements in the process are allowed to be scheduled into several states. This is provided by super-state mode for scheduling flexibility. For communication such as with memory in fixed cycles, the operations between two `wait until clock` statements are not allowed to be separated into different states. In this case, cycle-fixed mode is used. These

two scheduling modes are also recognized important and provided by Synopsys Behavior Compiler [5].

The allocater converts an FSMD-level description into an RT-level description, where the datapath with word-level operations and the control path with symbolic FSMs are separated. The allocation algorithm binds variables to registers, operations to function units, and data transfers to multiplexers and interconnections. The datapath is first constructed heuristically, and then refined by rip-up and re-binding [8] using branch-and-bound method.

The RTL sythesizer translates an RT-level description containing datapaths and control units to a netlist of gates for a specific technology library. A datapath element such as adder, multiplexer, multiplier, comparator, etc. is generated by a module generator, while the control logic is optimized by logic synthesis tools.

4.2 MEBS Synthesis Feature

Features of MEBS that were found especially useful during our ATM switch synthesis are highlighted as follows:

- *Design space exploration:*

  Compared with most other behavior synthesis approach which estimates area and performance based on RT-level implementation [9], MEBS explores the design space by synthesizing the ATM switch model down to gate level and estimates them at gate level. MEBS uses a fast logic synthesis tool incorporated with the module generator for RTL synthesis. Estimating design quality at gate level provides the user with realistic measure of the design implementation.

  Exploring the design space is an iterative and interactive process. Each iteration consists of the following three steps:

  1. Interactively make selection of synthesis parameters for design constraints, such as function units, types, and clock period.
  2. Perform scheduling, allocation, and RTL synthesis.
  3. Report timing and area. If the result is not satisfactory, go back to step 1 and modify the synthesis parameters.

  For the ATM switch design containing multiple processes, we can specify and change the synthesis parameters for one process at a time to obtain the design curve for each process. For multiple processes that are driven by the same clock, the total area is the summation of the estimated areas of the processes, while the clock cycle time is the maximum of the estimated clock periods of the processes. Such area and clock period estimation over multiple processes is calculated by combining the design space curves of the processes to derive a new design space for the overall circuit. This can be done in linear time in terms of the total design points of the processes.

- *Module function implementation:*

  For subprogram construct, the most common synthesis method is *in-line expansion*, where the subprogram body is expanded into the main program wherever it is invoked. In-line expansion, however, can suffer from program size explosion if too many copies of subprogram bodies are expanded. So, MEBS provides another synthesis method called *module function implementation*, in addition to in-line expansion.

  In module function implementation, each subprogram is synthesized as an independent module, and is treated as a function unit. It is shared by all the calls to the subprogram in the same way as an adder or a multiplier module is shared. Experimental result in Section 5 shows that for our ATM switch model, module function implementation reduces the circuit size and the synthesis time drastically when compared with in-line expansion.

- *Incremental synthesis:*

  Incremental synthesis was extensively used for the ATM switch design because the behavior model was incrementally developed. Furthermore, modifications due to bugs or specification changes were often localized in certain modules of the design. Without incremental synthesis, each local modification would require complete resynthesis, which was time-consuming and would abandon good synthesis result even though it had already satisfied design constraints.

  MEBS supports the following features for incremental synthesis:

  1. *Process-by-process synthesis:* A user can synthesize one process at a time, while leaving the other processes intact. This is useful when we concentrate on critical design parts first. The synthesis result of a process can be easily removed if it does not meet design constraints.
  2. *Source-in and source-out:* After each step of scheduling, allocation, and RTL synthesis, MEBS can output the result in synthesizable and simulatable VHDL. This enables the user to simulate the design at each level to verify the synthesis results.
  3. *Signal name preserving:* Furthermore, signal names are preserved in each step of the synthesis flow. This is also helpful for debugging purpose, because the same testbench can be easily applied to verify the correctness of the synthesis result at different stages of the design flow.

5 Experimental Result

This section reports the synthesis result of the ATM switch model by MEBS using Fujitsu CS35 0.8*um* standard–cell library.

5.1 Design Space Exploration

Three processes are used to illustrate the design space exploration results: `WAF_hptr_update`, `RAF0_hptr_update`, and `RAF2_hptr_update`, all driven by the same clock `sw_clk`. Initially, these processes were synthesized using the minimum resource constraints, i.e., 1 `addr_subtract` unit for each process. The resulting clock cycle time was about 51 ns. If the number of `add_subtract` units for each process was increased to 2, the clock cycle time was reduced to about 26 ns, which gave better performance but larger area.

Table 1 shows the experimental data of design space exploration for these three processes. The area is given in terms of the number of basic cells (or inverters) in CS35 library. The total area is 508 basic cells and the delay is 50.48 ns when the total resource was set to be 3 `add_subtract` units. If the resource was increased to 6 `add_subtract` units, the total area was increased to 620 basic cells but the delay was reduced to 25.56 ns.

Table 1: Design space exploration by MEBS for three processes (clocked by `sw_clk`) in the ATM switch design.

process	resource (`add_subtract`)	area (basic cell)	delay (ns)
`WAF_hptr_update`	1	170	50.48
	2	208	25.56
`RAF0_hptr_update`	1	169	50.48
	2	206	24.96
`RAF2_hptr_update`	1	169	50.48
	2	206	25.56

Table 2: Comparison of area, delay, and synthesis time of the ATM switch by two methods of subprogram implementation.

# of calls	method	area (basic cell)	delay (ns)	synthesis CPU time (s)
18	in-line	14161	45.11	1181.0
	module function	10797	39.21	396.5

5.2 Module Function Implementation

Both in–line expansion and module function implementation were used to synthesize the subprograms in the ATM switch description. The result for the entire switch is shown in Table 2, given the resource constraint of 6 `add_subtract` units for the three processes discussed in Section 5.1. There are totally 18 subprogram calls in the design description. The experiment shows that by implementing subprograms as module functions, the resulting design's area, delay, and synthesis time are better than that by using in–line expansion.

5.3 Synthesis Result

For CS35 technology library, the switch clock (`sw_clk`) cycle time constraint was set to 42 ns, and the highway clock (`HW_clk`) cycle time constraint was therefore set to 14 ns. The resource constraint for the three processes in Section 5.1 was set to 6 `add_subtract` units. Given these design constraints, MEBS was able to synthesize the complete ATM switch behavior model down to a gate–level netlist.

Table 3 shows the number of lines in the VHDL description synthesized by MEBS at each level of abstraction. It indicates that it is much simpler to describe the design at the initial mixed levels than at RT level or gate level. Therefore, it will be much easier to maintain and modify the design at the initial mixed levels.

Table 4 shows the statistics of the synthesized result. It lists the numbers of I/O ports, nets, library cells of the final implementation. It also gives the total area and delay. The total area is the summation of the areas of latches, flip–flips, and combinational logic.

It is difficult to accurately compare our synthesis result with the manual design, because some extra functions, such as communication with the central switch controller, were added in the manual design, which are not described in the switch architecture specification. Besides, these functions are not easy to be separated from the remaining circuitry of the manual design. However, since the implementation of these extra functions is considered a small part of the entire switch, we estimated conservatively that the size of our synthesis result is about 85% of the manual design in terms of the gate–level basic cells. Furthermore, our high–level modeling and synthesis approach took about 4 man–weeks, while the manual design took several man–months.

Table 3: Number of VHDL lines in the ATM switch model at each level of abstraction.

level	#line
initial mixed levels	1500
FSMD level	1800
RT level	3581
gate level	12869

Table 4: Statistics of the synthesized ATM design using CS35 technology library (bc: basic cell).

#port	118
#net	6143
#cell	5946
latch area	432 bc
flip–flop area	4942 bc
combinational logic area	5423 bc
total	10797 bc
delay	39.21 ns

6 Conclusion

This paper presented our experience on domain-specific high-level modeling and synthesis for Fujitsu ATM switch design. We proposed a high-level design methodology which considers the target ATM switch architectural features during behavior modeling, and applies a prototyped control-dominated high-level synthesis compiler, MEBS, to generate the gate-level implementation from the behavior model. Since the specific ATM switch architecture was incorporated into both modeling and synthesis phases, a high-quality design was efficiently derived. The synthesis results showed that given the design constraints, the proposed high-level design methodology was able to produce a gate-level implementation by MEBS with about 15% area reduction in shorter design cycle when compared with manual design.

Acknowledgment: The authors would like to thank Masami Yamazaki of Fujitsu Ltd. for interesting discussion on ATM Switch design, and Fur-shing Tsai, Ta-Yung Liu, and Shi-Zheng Lin at UC Riverside for technical support of MEBS. The management assistance by Fumiyasu Hirose and Mitsuru Kuga of Fujitsu Ltd. is highly appreciated as well.

References

[1] M. D. Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN.* Ellis Horwood Limited, 1993.

[2] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis: Introduction to Chip and System Design.* Kluwer Academic Publishers, 1992.

[3] R. A. Bergamaschi and A. Kuehlmann. "A system for production use of high-level synthesis". In *IEEE Trans. VLSI Systems*, pages 233–243, Sept. 1993.

[4] J. Biesenack, *et al.*, "The Siemens high-level synthesis system CALLAS". In *IEEE Trans. VLSI Systems*, pages 244–253, Sept. 1993.

[5] D. Knapp, T. Ly, D. MacMillen, and R. Miller. "Behavioral synthesis methodology for HDL-based specification and validation". In *Proc. DAC*, pages 286–291, 1995. Francisco, June 1995.

[6] T. Ly, D. Knapp, R. Miller, and D. MacMillen. "Scheduling using behavioral templates". In *Proc. DAC*, pages 101–106, 1995.

[7] F. Vahid and D. Gajski. "Clustering for improved system-level functional partitioning". In *Proc. Int. Symp. System Synthesis*, pages 28–33, 1995.

[8] F.-S. Tsai and Y.-C. Hsu. "STAR: a system for hardware allocation in data path synthesis". In *IEEE Trans. CAD*, pages 1053–1064, Sep. 1992.

[9] C. Ramachandran, *et al.*, "Accurate layout area and delay modeling for system level design". In *Proc. ICCAD*, pages 355–361, 1992.