6. МОДЕЛИРОВАНИЕ И СИНТЕЗ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

УДК 681.3

METHODS AND TOOLS FOR HIGH AND SYSTEM LEVEL SYNTHESIS

Anatoly Prihozhy and Redouane Merdjani

State University of Informatics and Radioelectronics

1. Introduction

Effective high-level synthesis systems include ALERT, AMICAL, CATHED-RAL, CMUDA, DAA, ELLA, FACET, HAL, HIS, MAHA, MIMOLA, PSAL2, York-town Silicon Compiler, and others (Camposano,1989, Courtois,1994, Gajski,1992, 1994, Goossens,1989, Jerraya,1993, Mcfarland,1990, Mermet,1993, Vercest, 1990). They develop synthesis methodology consisting of the following tasks:

- compiling a behavioral description presented in a hardware description language to an intermediate format
- generating the control (CFG) and data (DFG) flow graphs
- scheduling the description operators and statements
- allocation of the functional, storage, and interconnection units
- binding the behavior constructions to the units
- generating the data path (DP) and finite state machine (FSM).

This paper presents new methods and tools for high- and system-level synthesis that explore behavioral description transformation, synthesis techniques based on orthogonality analysis, and net-based synthesis techniques.

2. Synthesis methodology

The methodology is based on the following three main principles:

- transforming the source behavioral model to a special model, allowing efficient synthesis of high quality RTL-structures
- development and use of new analyzing, scheduling, allocation, binding, data path generation, and finite state machine generation techniques that explore the special model advantages
- extending traditional high-level synthesis methodology in order to automatically design and optimize asynchronous circuits and systems.

3. Behavioral model transformation

The special behavioral model is described through using a subset of VHDL statements. The *wait*, *signal assignment*, *variable assignment*, *loop*, *exit*, and *next* statements may execute unconditionally and conditionally. The *loop* statement has no the *iteration scheme*. The behavioral description CFG constructed of these statements has only one *segmented path*. Each segment may be processed separately. This allows the development of efficient lifetime analysis, scheduling, allocation, and binding techniques. The behavioral model for GCD is shown in Figure 1. A source VHDL behavioral description is equivalently transformed to the special model by applying transformation rules. These modify the behavior CDFG to speed up the design process and to improve the design parameters. The rules transform a *loop* statement with the iteration scheme to a loop without the scheme, reorder independent and dependent statements, insert a statement into *if*- and *loop*-statements, extract computations from *if*-statement, split *if*-statement into separate parts, transform *if*-statement to a logical expression and variable assignment state-ment, merge *exit*-statements, unroll *loop*-statement without the iteration scheme.

4. Scheduling for the special model

The novel extended scheduling tech-niques use orthogonality, compatibility, precedence, and proximity relations const-ructed on the sets of signals, variables, operators, and statements. The segment tree is a hierarchical structure of the special behavioral model CFG. The tree root is the process statement. The other non-terminal nodes are loop statements. The terminal nodes are sequential statements of the model. The loop or process body statements constitute a segment. The FSM states are introduced during top down traversal of the segment tree.

5. Allocation and binding for the special model

Allocation and binding methods developed for the special behavioral model use the model advantages. The single-path-based allocation and binding flow is as follows. First, the special behavioral model DFG is generated. Using the CFG single path, the variable lifetimes are computed and the variables compatibility is determined. The compatibility analysis is performed accounting the orthogonality analysis and scheduling results. Using the variables, operators, and statements compatibility, the functional, storage, and interconnection units are allocated. Each variable, operator, and statement is binded to a unit in such a way as to minimize the DP and FSM cost.

```
entity GCD is
  port(CLOCK, RESET, START: in BIT;
    XI,YI: in BIT_VECTOR(15 downto 0);
    READY: out BIT;
    RES: out BIT_VECTOR(15 downto 0));
end GCD;
architecture BEHAVIOR of GCD is
  attribute CYCLE_TIME of BEHAVIOR:architecture is 60NS;
  attribute TOTAL_AREA of BEHAVIOR:architecture is 500GT;
  attribute FUNCTION_UNITS of BEHAVIOR:architecture is "ALU16-1";
  attribute EXECUTION_TIME of BEHAVIOR:architecture is 2US;
  attribute CRITERION of BEHAVIOR:architecture is "minT";
begin
  P1:process
    variable X,Y:BIT_VECTOR(15 downto 0);
    variable V1,V2,V3,V4:BOOLEAN;
    attribute PROBABILITY of V1:variable is 1;
    attribute PROBABILITY of V2:variable is 0.05;
    attribute PROBABILITY of V3,V4: variable is 0.475;
  begin                                  -- Statement  Segment  Path
    loop                                    --1          1
    wait until CLOCK'Event and CLOCK='1';
      V1:= not START='1';                   --2          2
      if V1 then READY<='0'; end if;        --3          2
      if V1 then X:=XI; end if;             --4          2
      if V1 then Y:=YI; end if;             --5          2
      exit when V1;                         --6          2
    end loop;
      loop                                  --7          1
    wait until CLOCK'Event and CLOCK='1';
      V2:=X=Y;                              --8          3
      V3:=X<Y;                              --9          3
      V4:=X>Y;                              --10         3
      if V3 then Y:=Y-X; end if;            --11         3
      if V4 then X:=X-Y; end if;            --12         3
      if V2 then READY<='1'; end if;        --13         3
      if V2 then RES<=X; end if;            --14         3
      exit when V2;                         --15         3
    end loop;
  end process;
end BEHAVIOR;
```

Figure 1: GCD special behavioral model

6. Net-based synthesis

There are two main problems in synthesis of circuits and systems composed of variable execution time components, how to

- perform the scheduling, allocation, and binding tasks to optimize the design
- build the components, synthesize the control, and construct a system.

Both the problems can be solved within high-level synthesis net-based methodology (Prihozhy,1996). The key concept of net-based synthesis is a net schedule that desc-ribes mixed sequential/concurrent execution of the statements. The set D_M of the concurrent statement pairs defines the net schedule of maximum concurrency. For a subset D of set D_M we search for a net schedule of less concurrency. Two optimization tasks are possible in order to

- minimize the net schedule execution time with constraints on the cost
- minimize the net schedule cost with constraints on the execution time.

The net schedule can be a source for synthesizing sequential schedules. An ordinary sequential schedule is generated by ASAP and ALAP techniques if the statement execution time equals the clock cycle time. A sequential schedule with chaining is generated by the list scheduling and other techniques with constraints on the cost or on the number of functional units. A sequential schedule with multicycling is generated if the functional unit execution time is greater than the clock cycle time. If functional units are functionally pipelined, statements have to be splitted into parts, one for each stage of the pipeline.

Synthesis of asynchronous circuits and systems is based on net scheduling algorithms, net allocation algorithms for functional, storage, and interconnection units, net binding algorithms, methods of constructing asynchronous circuit and system components, methods of synthesis of asynchronous circuits and systems composed of these components.

7. Results

The described models, methods, techniques, and algorithms are realized within the AHILES high-level synthesis system (Figure 2) (Prihozhy,1996). Results generated for five benchmarks (Courtois,1994, Mermet,1993) are presented in Tables 1 and 2. All the RTL-structures are synthesized on a PC 486/50. The VHDL compiler throughput is 100 to 280 lines per second. The overall synthesis time is 5 to 14 sec. Generated RTL-structure parameters appear in Table 1. The internal form size is 1.4 times greater than the VHDL-text size for behavioral descriptions and 0.86 times less for structural descriptions. AHILES introduced few FSM states for all the designs. This is due to the preliminary transformation of the behavioral descriptions, special behavioral model, and novel scheduling, allocation, and binding techniques. The average execution time of generated net schedules is 18% less than the execution time of optimal sequential schedules.

Table 1 Design parameters

Parameter	*Benchmarks*				
	Bubble	*Gcd*	*Gcdf*	*Kalman*	*Pid*
Behavior VHDL text (lines)	119	50	60	220	180
Behavior VHDL text (bytes)	3009	2089	2844	7966	9978
Behavior internal form (bytes)	10148	7160	7512	19478	13680
Statements	79	19	29	176	122
Objects	46	15	19	122	75
CFG and DFG (bytes)	5171	1409	1987	12340	8152
FSM states	20	2	5	16	23
FSM transitions	31	4	9	29	33
ALUs	0	1	1	1	1
Functional units width (bits)	0	16	32	17	32
Registers	7	2	2	18	13
Register width (bits)	104	32	64	138	389

RAMs	1	0	0	3	0
ROMs	0	0	0	3	1
Collectors	0	0	0	5	9
Multiplexers	4	4	4	14	8
Multiplexer width (bits)	68	64	128	155	227
Multiplexer inputs	13	8	8	36	33
DP internal form (bytes)	6177	3031	3170	17075	12425
FSM internal form (bytes)	4522	1132	1532	9927	8197
Structure internal form (bytes)	10699	4163	4702	27002	20622
Structure VHDL text (lines)	416	164	184	1000	724
Structure VHDL text (bytes)	12383	4647	5241	31550	22938

9. References

1. Jerraya, A.A., Park, I., O'Brien, K. (1993) Amical: An Interactive High-Level Synthesis Environment, in *Proc. EDAC'93*, IEEE Computer Society Press, Los Alamitos, Calif.
2. Mermet, J. ed. (1993) Fundamentals and Standards in Hardware Description Languages. Kluwer Academic Publishers, the Netherlands.
3. Prihozhy, A. (1996) Net Scheduling in High-Level Synthesis. *IEEE Design & Test of Computers*, **13**, Spring, 26-35.
4. Prihozhy, A. (1996) Use of VHDL-Based Design Methodology and AHILES System for Education in Belarus, in *Proc. Europ. Workshop on Microelectr. Education* (ed. G. Kamarinos, N. Guillemot., and B. Courtois), World Scientific, Singapore, 217-20.