Control and Cybernetics

vol. **22** (1993) No. 3

A system for behavioral level synthesis of PLD–based circuits

by

Stanisław Deniziak and Krzysztof Sapiecha

Kielce University of Technology,
Department of Computer Science
Al. Tysiąclecia Państwa Polskiego 7,
25–314 Kielce
Poland

PLD–based circuits have been designed with the help of specific software tools. However, input formats of such tools (CUPL's input format, for example[1]) describe a circuit at a very low logic level. Moreover, they require an early selection of the target PLD devices which is particularly inconvenient when more complex circuits are considered. Therefore, designing of large circuits with the help of these tools is tiring and time consuming.

In the paper another concept is investigated. Behavioral description of a circuit is formulated using procedural CHDL called UPLAND. Then, UPLAND source description is automatically translated into its corresponding target input format (CUPL's input format) where target PLD devices are optional. The paper introduces UPLAND and outlines the principles of CUPLAND compiler work.

Keywords: PLD–based systems, behavioral level synthesis, design automation, ASICs.

1. Introduction

In recent years the volume of ICs implemented as ASICs has been constantly growing. One of technologies applied here is the semi–custom one using PLD devices as standard components. It has proved its low costs and simplicity. Widely available, erasable and cheap PLDs are excellent for rapid development of prototypes, short series production and training — Muroga (1982), Majewski, Luba, Jasiński, Zbiechrzowski (1992), Bolton (1990).

PLDs must be programmed for each specific application (function). Software tools supporting PLD programming play a key role in the whole digital system design procedure.

[1] As an example P–CAD's CUPL was choosen for its wide popularity. However, input format of any such tool might be considered, as well.

P–CAD's CUPL is one of the most popular tools for translating functional description of a circuit into its corresponding PLD structure, PCAD (1986). However, CUPL's input format describes a circuit at the very low logic level. Moreover, it requires an early selection of the target PLD devices which is particularly inconvenient when more complex circuits are considered. Therefore, designing of large circuits with the help of CUPL is tiring and time consuming.

In this paper a behavioral level description compiler called CUPLAND is introduced, Deniziak, Sapiecha (1992). It can compile a behavioral level description of a system (circuit) into its corresponding CUPL input format where target PLD devices are optional. Flip–flops and asynchronous outputs are reported which enables the choice of an optimum PLD set.

Thereby, a system designer describes designed systems at the behavioral level (RTL–level), much higher logic level then CUPL does. A starting point of the designing procedure is shifted to the upper level of abstraction. Then the designing procedure is much easier and faster.

Section 2 describes the main features of UPLAND "by example". Section 3 presents principles of CUPLAND work. The paper ends with conclusions.

2. Describing digital system behavior with UPLAND

UPLAND, Sapiecha K., Deniziak S., Sapiecha J. (1991), is a procedural CHDL which refers to the behavioral level of the system (circuit) description, Hartenstein (1987). It stems from a very popular and efficient technique of digital systems (circuits) designing which is based on a concept of a control graph and a microoperation, Chu (1972).

As a procedure any UPLAND description consists of two parts: a head and a body. The former declares all variables used in the description (IN:inputs, OUT:outputs, BUS:bidirectional lines, OWN:auxiliary variables (if necessary), CLOCK:main system clock and SUBCLK:subclocks (if any)) of the system (circuit). Synchronous and asynchronous assignments (microoperations) are distinguished using a specific declaration of SEQ:sequential variables. The latter describes an algorithm which is performed by a circuit. Moreover, the description may more or less adequately reflect the circuit structure.

The UPLAND description of the algorithm consists of labeled microinstructions. Each microinstruction consists of an execution part and a control part, both containing synchronous, asynchronous, conditional and unconditional microoperations.

The execution part refers to an operation unit of the system and comprises all operational microoperations which can be executed at the same clock cycle. The control part refers to the control unit of the system and points out to a microinstruction to be executed in the next clock cycle. It begins with 'NEXT' keyword and may be conditional.

UPLAND enables the description of main functional blocks like registers, counters, multiplexers, adders etc. using a single assignment statement (single

microoperation). Moreover, it contains standard microoperations which make it possible to describe easily some more complicated functions like carry generation for adders, shifting, rotations etc.. Single–bit variables and vector (multi–bit) ones are available.

Main features of UPLAND are the following:

- procedural system description,
- time factor,
- labeled microinstructions describing main states of the system,
- distingushed execution and control parts of each microinstruction,
- both synchronous and asynchronous assignment microoperations,
- numbering of pins,
- multiclock timing,
- standard microoperations,
- input, output, bus, internal(own), external (when modules are distinguished) variables.

As an example let us consider the description of the Intel 8214 interrupt controller. This chip has 8 inputs of interrupt request (IR0,...,IR7) with fixed priority. When some interrupts are requested and the inputs INTE and ETLG of the controller have been activated then the interrupt with highest priority is selected provided that it is not masked.

If this happens the output INT is activated for one CK clock cycle and the address of the selected interrupt appears on A2,...,A0 outputs. An interrupt request is masked when the number loaded into the BR register equals or is higher then the address of an interrupt, and masking is enabled (BR3=0).

States of the B2,...,B0 and SGS inputs are loaded into the BR register during the rising edge of ECS clock. ECS also clears the interrupt disable register which is switched on when INT is active. The input ELR enables the A2,...,A0 outputs (active low). ENLG output is used for disabling interrupts in other 8214 chips of the system (if more than 8 interrupts are serviced by 8214 controllers).

The UPLAND description of the circuit begins with the following declarations:

BBOX: *I8214;*
IN: *ir(7:0), b(2:0), sgs, inte, ecs, clk, etlg, elr;*
OUT: *a(2:0), int, enlg;*
OWN: *br(2:0), br3, intdis, irqno(2:0), irq, nomask, intreq;*
CLOCK: *clk=R;*
SUBCLK: *ecs;*
SEQ: *(intdis,br,br3) (ecs=R),irqno,irq;*

Auxiliary variables: *irqno, irq, nomask* and *intreq* are introduced for the description of intermediate variable functions. Registers: *intdis, br* and *br3* are synchronized with the rising edge of the *ecs* clock, while latches *irqno* and *irq* are loaded during low level of *intdis.*

The remaining part of the description of the 8214 controller consists of two

following microinstructions:

```
0:intdis:=0,  br:=b,  br3:=sgs,
COND (intdis):
/0/: COND(ir(7),ir(6),ir(5),ir(4),ir(3),ir(2),ir(1),ir(0)):
 /0,x,x,x,x,x,x,x/:   irqno <= 7,   irq<=1
 /1,0,x,x,x,x,x,x/:   irqno <= 6,   irq<=1
 /1,1,0,x,x,x,x,x/:   irqno <= 5,   irq<=1
 /1,1,1,0,x,x,x,x/:   irqno <= 4,   irq<=1
 /1,1,1,1,0,x,x,x/:   irqno <= 3,   irq<=1
 /1,1,1,1,1,0,x,x/:   irqno <= 2,   irq<=1
 /1,1,1,1,1,1,0,x/:   irqno <= 1,   irq<=1
 /1,1,1,1,1,1,1,0/:   irqno <= 0,   irq<=1

   ELSE irq <=0
   END
END,
nomask <= (irqno>br) | br3,
intreq <= irq & etlg & inte & !intdis & nomask,
enlg <= !irq & etlg & br3,
COND(elr,etlg,irq) :
/0,1,1/: a <= irqno
ELSE a <= Z
END,
int <= Z
NEXT COND(intreq):
    /0/: 0
    /1/: 1
    END;

1: intdis <= 1, br:=b,  br3:=sgs,
 int <= 0, enlg <= 0,
 COND(elr,etlg):
  /0,1/: a <= irqno
 ELSE a <= Z
 END,
 NEXT 0;
END.
```

where ":=" and "<=" respectively denote synchronous and asynchronous assignments (microoperations), and Z denotes high impedance.

First microinstruction (labeled 0:) describes the behavior of the controller while waiting for any request of nonmasked interrupt. When this happens the controller leaves the state #0 and reaches the state #1. The second microoperation (labeled 1:) describes the behavior of the controller in this new state. INT output is active during only one CK clock cycle.

It is readily seen that descriptions of large digital circuits (digital systems) given in UPLAND are much shorter and much more understandable than the ones obtained using CUPL input format. Even a priority decoder (described above in UPLAND by one conditional microinstruction) or a comparator (described in UPLAND by one assignment microoperation) require CUPL input formats comparable in lengths with the complete description of the controller.

For example, CUPL input format of the decoder might be described as follows:

```
IRQNO[0].ap = !INTDIS & !STATE &
 (!IR[7] #
  !IR[5] & IR[6] & IR[7] #
  !IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[0].ar = !INTDIS & !STATE &
 (!IR[6] & IR[7] #
  !IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[0] & IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[0].d=IRQNO[0];
IRQNO[1].ap = !INTDIS & !STATE &
 (!IR[7] #
  !IR[6] & IR[7] #
  !IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[1].ar = !INTDIS & !STATE &
 (!IR[5] & IR[6] & IR[7] #
  !IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[0] & IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[1].d=IRQN0[1];
IRQNO[2].ap = !INTDIS & !STATE &
 (!IR[7] #
  !IR[6] & IR[7] #
  !IR[5] & IR[6] & IR[7] #
  !IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[2].ar = !INTDIS & !STATE &
 (IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7] #
  !IR[0] & IR[1] & IR[2] & IR[3] & IR[4] & IR[5] & IR[6] & IR[7]);
IRQNO[2].d=IRQNO[2];
```

CUPLAND compiles an UPLAND source code into CUPL input format (object code). In case of the decoder it automatically generates equations equivalent

to the above ones, starting from the description given in UPLAND.

For the controller the CUPL input format which would have been written (manually) by a designer would consist of 230 lines comparing with 40 lines of its structured UPLAND description given above.

3. UPLAND to CUPL input format translation rules

Each vector operation in a source code is compiled into a sequence of single-bit statements in the object code.

Object code generation is performed in five steps:
- generation of statements for combinational variables,
- generation of statements for sequential variables,
- generation of statements for clocks,
- generation of statements for bidirectional lines,
- machine state generation.

For each combinational variable sequence of logic the following expressions are generated:

APPEND var = *exp* & *cond* & *state:state_i*

where: *var* is a name of variable;
exp defines an expression corresponding to the right side of the microoperation:
cond is an expression generated for conditional microoperations;
state represents state variable;
state_i is a constant which defines the state of activity of the involved microoperation.

For example, the source code

```
...
4: COND(a,b,c):
  /0,x,1/: y <= d | e
  ...
```

is translated into the following object code:

APPEND *y=(d # e) & (!a & c) & state:state_4;*

For each sequential variable one SEQUENCE statement is generated. It deals with all synchronous microoperations referring to the variable. The SEQUENCE statement includes a pair of NEXT statements for each of the involved microoperations. The first NEXT statement corresponds to the state 0 of the variable and the second one corresponds to the state 1 of it. The skeleton of the SEQUENCE statement is as follows:

```
SEQUENCE var {
PRESENT 'b'0
```

```
  IF exp & cond & state:state_i NEXT 'b'1
  ...
  DEFAULT NEXT 'b'0
  PRESENT 'b'1
   IF !exp & cond & state:state _i NEXT 'b'0
   ...
   DEFAULT NEXT 'b'1
}
```

For each sequential variable a sequence of logic expressions for all asynchronous microoperations is generated. Hence:

```
APPEND var.AR = !exp & cond & state:state_i;
APPEND var.AP =  exp & cond & state:state_i;
```

For example, the source code

```
...
4:COND (a,b,c):
 /0,x,1/: q := d | e
 /0,1,0/: q <= f
...
```

is translated into the following object code:

```
SEQUENCE q {
    PRESENT 'b'0
       IF ( d # e ) & (!a & c) & state:state_4 NEXT 'b'1
      DEFAULT NEXT 'b'0
    PRESENT 'b'1
       IF !( d # e) & (!a & c) & state:state_4 NEXT 'b'0
      DEFAULT NEXT 'b'1
    }
APPEND q.AR = !f & (!a & b & !c) & state:state_4;
APPEND q.AP =  f & (!a & b & !c) & state:state_4;
```

For each sequential variable synchronized with nonstandard clock signals a set of equations controlling programmable clock inputs is generated.

Let us consider the following description:

```
...
CLOCK: CK=F;
SUBCLK: CK1,CK2;
SEQ: Q1,(Q2) (CK1=R),(Q3) (CK2=F);
...
```

Timing signals of the *Q1* and *Q2* registers are nonstandard. Hence, a PLD chip with programmable clocks must be applied and the following statements should be generated:

```
Q1.CK = !CK;
```

Q2.CK = *CK1;*
Q3.CK = *!CK2;*

For each bidirectional line and a tri–state output a set of equations controlling an output enable line (OE) is generated.

For example, the source code:

```
...
OUT: Y2;
BUS: Y1;
...
0: I <= Y1, Y2 <= Z,
...
1: COND(A,B):
   /0,1/: Y1 <= C, Y2 <= Z
...
2: Y1 <= D, Y2 <= A,
...
```

is translated into the following object code:

Y1.OE = *!A & B & state:state_1 # state:state_2;*
Y2.OE = *!(state:state_0 # !A & B & state:state_1);*

For devices using more than one state (more than one microinstruction in UPLAND) a state machine (a control unit) is to be generated. For this purpose an n–bit *STATE* variable is created that holds $2^n \leq <$ number of states $< 2^{n+1}$. To generate a SEQUENCE statement describing the state machine the following rules are applied:

- unconditional NEXT *i;* is translated into a NEXT *state_i;* statement;
- conditional NEXT *i;* is translated into a sequence of IF *cond* NEXT *state_i;* statements;
- ELSE *i;* is translated into a DEFAULT NEXT *state_i;* statement.

State encoding for a finite state machine (FSM) is performed using a simple optimization procedure. This procedure is described in Deniziak, Sapiecha (1993).

4. Conclusions

The application of UPLAND causes that designing of large circuits requires much less effort than when CUPL is used directly. However, shifting of a starting point of a digital circuit design procedure to the upper level of abstraction may sometimes result in a nearly optimal PLD–based structure.

Non–optimality of the CUPLAND's work may be caused by two reasons: CUPLAND in the current version does not minimize boolean functions and it does not take into account all specific structural (schematic) features of all PLD devices.

For a boolean function F CUPL enables only the description of the sets F^1

and F^0. If the function is partly specified (set F^x is not empty) then minimization of the function should be done before the translation of its description into the CUPL input format. This may be easily achieved by enriching CUPLAND with an appriopriate minimization algorithm (ESPRESSO, for example).

Internal XOR gates could be taken as an example of a specific structural feature of a PLD device which would not be optimally used by CUPLAND. Instead of an optimal expression $F = F_i$ **xor** F_j an expression $F = F$ **xor** 0 might be generated.

In the paper an example is given which illustrates CUPLAND efficiency. Usually, an UPLAND source code length is ten times smaller than its corresponding CUPL input code length, not mentioning their readability.

The selection of PLD devices is shifted to the end of the design procedure which makes minimization of the structure possible.

Finally, the source description of the design is much more understandable. Its correctness can be easily checked by simulation.

References

BOLTON M. (1990) Digital Systems Design with Programmable Logic, Addison–Wesley Publishing Company.

CHU Y. (1972) Computer Organization and Microprogramming, Prentice–Hall.

DENIZIAK S., SAPIECHA K. (1992) CUPLAND–Behavioral Level Description Compiler for Designing of PLD–based Circuits, *Proc. International Conference of Microelectronics*, Sept. 1992, 205–212.

DENIZIAK S., SAPIECHA K. (1993) State Encoding Optimization Procedure for Synthesis of PLD–based Circuits, *Technical Report No.* **1/93,** Department of Computer Science, Kielce University of Technology.

HARTENSTEIN R.W. (ED.) (1987) Hardware Description Languages, Elsevier Science Publishers B.V. North–Holland.

Intel Component Data Catalog, INTEL Corp., 1980.

MAJEWSKI W., ŁUBA T., JASIŃSKI K., ZBIECHRZOWSKI B. (1992) Programowalne moduły logiczne w syntezie układów cyfrowych, WKiŁ, Warszawa.

MUROGA S. (1982) VLSI SYSTEM DESIGN, John Wiley & Sons, Inc.

PCAD (1986) CUPL User's Manual, Personal CAD Systems, Inc., June 1986.

SAPIECHA K., DENIZIAK S., SAPIECHA J. (1991) MESSMATE–Modular Economical System for designing, verifying And TEsting of digital systems, ResComp, 1991.