数理解析研究所講究録
第1186巻 2001年 177-188

Verification of logic circuits using Mizar and its application to an adder circuit on a radix-2^kSD number.

長野県情報技術試験場 清水英孝(Hidetaka SHIMIZU)

信州大学工学部 中村八束(Yatsuka NAKAMURA)

信州大学工学部 藤沢義範(Yoshinori FUJISAWA)

信州大学工学部 不破 泰(Yasushi FUWA)

1. Introduction

To answer the request of higher performance from electric equipment, logic circuit is becoming more complicated and more large-scaled. This makes it more difficult to verify correctness of a designed circuit.

Heretofore, to verify a logical circuit, output for every possible state and input should be confirmed by simulation. However, when the scale of a circuit increases, states of circuit increases exponentially, accordingly (Example, when a circuit has 100 return wires, it has 2^{100} states.). So it is impossible or very difficult to complete such a simulation for large-scaled circuits.

As a new way to verify correctness of logical circuit, we express logical circuit with mathematical description (called mathematical model). The correctness of the circuits is assured when correctness of the

circuit's mathematical model is verified by a proof checker system.

A proof checker system has been used to verify correctness of proof in mathematics. It can be applied to verifying correctness of a designed circuit with the method we suggest.

In this paper, after introducing the proof checker system Mizar which is used to verify correctness of proof (Section 2), we give some definitions as a preparation for mathematics descriptions of a logical circuit (Section 3). Then, we explain how a logical circuit is described by these definitions, and how its correctness is verified by the system (Section 4). At last, as an example, we apply the method in designing the adder circuit on a radix-2^kSD number.

2. Proof checker system Mizar

As an attempt to reconstruct mathematical vernacular, the Mizar project started in 1973. [1]

And, it has become the most important activity in the project to develop the database of mathematics since 1989. Now, more than 2,000 definitions and 20,000 theorems are included in the increasing database.

As a characteristic of Mizar, useful verified proof is accepted by Mizar's library. Using Mizar, besides mathematical proof, a mathematical model can be verified too.

Just as a large-scale circuit can be designed as a combination of

smaller circuits which function has been verified, the correctness of a model can be showed when the model is a combination of smaller models which has been accepted by the library.

3. Preparation for mathematical description of a logical circuit

We give a relation between basic concepts of logical circuits and mathematics as follows:

(1) We think input and output signals as sets. The logic of a signal line has 2 states, 0 and 1. 0 is defined as the empty set ϕ, and 1 is defined as a non-empty set.

It is described as follows at Mizar:

```
definition let a be set;
  redefine attr a is empty;
  antonym $a;
end;
```

(2) We consider the expression of every possible states formed by input and output signals. It is described like this:

```
$s0 iff $AND2 (NOT1 q2,NOT1 q1)
```

(3) We define a circuit as a Boolean function of sets defined above.

For example, NOT circuit writes follows:

```
func NOT1 a -> set equals
     φ  if $a
     otherwise {φ  : not contradiction};
  end;
```

4. A new method of logic circuit's verification using Mizar system

Here, we introduce how the new method works with a simple example.

Consider a 3bit up counter circuit. The correctness of the circuit can be guaranteed with Mizar at the following steps.

(1) Describing the input and output as sets.

(2) Defining every possible state of input and output as following with the set of step1. (Fig.1)

```
$s0=$AND3(NOT1 q3, NOT1 q2, NOT1 q1);
$s1=$AND3(NOT1 q3, NOT1 q2,      q1);
$s2,$s3,$s4,$s5,$s6,$s7 is similar to $s0,$s1.
```

(3) Expressing the behavior of 3bit up counter circuit with Boolean expressions as follows: (Fig.2)

```
$nq1=$AND2(NOT1 q1,R)
$nq2=$AND2(XOR2(q1,q2),R))
$nq3=$AND2(OR2(AND2(q3,NOT1 q1),AND2(q1, XOR2(q2,q3))),R)
```

Here, **$q1,$q2,$q3,R** are circuit inputs and **$nq1,$nq2,$nq3** are outputs.

(4) Verifying the correctness of circuit by confirming its Boolean expressions' tautology using Mizar system.

```
($ns1 iff $AND2(s0,R)) & ($ns2 iff $AND2(s1,R)) & ($ns3 iff $AND2(s2,R))&
($ns4 iff $AND2(s3,R)) & ($ns5 iff $AND2(s4,R)) & ($ns6 iff $AND2(s5,R))&
($ns7 iff $AND2(s6,R)) & ($ns0 iff $OR2(s7,NOT1 R));
```

Here, **$s0,…,$s7** means current states and **$ns0,…, $ns7** means next states of current states.

($ns1 iff $AND2(s0,R)) means the next state will be **$ns1**, if and only if the current state is **$s0** and **R** is "1".

Behavior of a 3bit up counter circuit

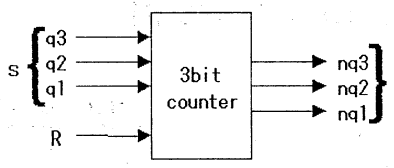
Have 4 inputs(R,q3,q2,q1) and 3outputs(nq3,nq2,nq1)

States change as follows:

000→001→010→011→100→101→110→111→000→

return to the initial state(000) by the reset input

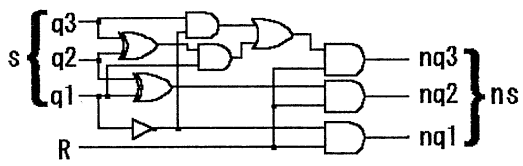
Signal definitions



Input signal definitions
O O O→ $s0=$AND3(NOT1 q3, NOT1 q2,NOT1 q1)
Output signal definitions
O O 1→ $ns1=$AND3(NOT1 nq3, NOT1 nq2, nq1)

Fig.1 Definitions of 3bit up counter.

Definitions and proof to correctness of a 3bit up counter



Definitions of a 3bit up counter

```
($nq1 iff $AND2(NOT1 q1,R))&
($nq2 iff $AND2(XOR2(q1,q2),R))&
($nq3 iff $AND2(OR2(AND2(q3, NOT1 q1),
                AND2(q1, XOR2(q2,q3))),R))
```

Needs to proof these definitions

```
($ns1 iff $AND2(s0,R))&
($ns2 iff $AND2(s1,R))&
($ns3 iff $AND2(s2,R))&
($ns4 iff $AND2(s3,R))&
($ns5 iff $AND2(s4,R))&
($ns6 iff $AND2(s5,R))&
($ns7 iff $AND2(s6,R))&
($ns0 iff $OR2(s7,NOT1 R));
```

Fig. 2 Definitions (cont.) and proof of correctness of 3bit up counter.

5. An application to a radix-2^kSD number coded adder circuit

Here, we apply the new method to designing an adder circuit on a radix-2^kSD number.

In a radix-2^kSD (signed-digit) coded adder circuit, calculations can be finished in a constant time no matter whether there is a ripple carry.

Here, we will verify the correctness of such a circuit of case k=2.

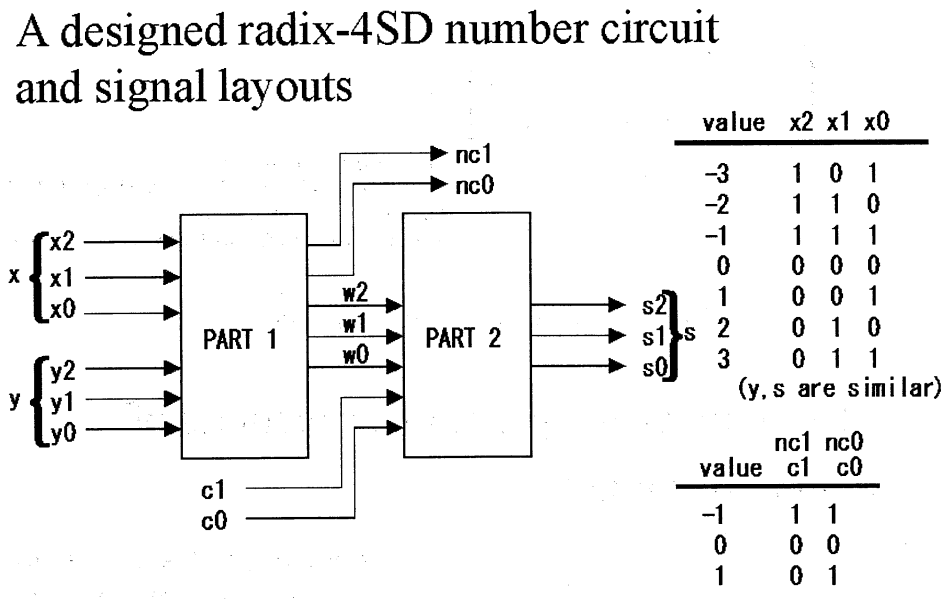


Fig.3 Signal layout of radix-4SD number coded adder circuit

Outputs of PART1 which used radix-4SD number

Value		w2	w1	w0	nc1	nc0	other expression
-6	-4 + -2	1	1	0	1	1	
-5	-4 + -1	1	1	1	1	1	
-4	-4 + 0	0	0	0	1	1	
-3	-4 + 1	0	0	1	1	1	0 + -3
-2	0 + -2	1	1	0	0	0	-4 + 2
-1	0 + -1	1	1	1	0	0	-4 + 1
0	0 + 0	0	0	0	0	0	
1	0 + 1	0	0	1	0	0	4 + -3
2	0 + 2	0	1	0	0	0	4 + -2
3	4 + -1	1	1	1	0	1	0 + 3
4	4 + 0	0	0	0	0	1	
5	4 + 1	0	0	1	0	1	
6	4 + 2	0	1	0	0	1	

Several kinds numerical value expression is possible so that figure increasing number can take out a mark in SD number.

In this study, we represent numerical value by expression surrounded by a square.

Fig.4 Definitions of radix-4SD number coded adder circuit

The correctness of a 4-SD number adder circuit is verified with the 4 steps described in former section.

First, input and output status can be defined as follows:

INPUT STATE:

```
($xm3 iff $AND3(      x2,NOT1 x1,      x0))&
($xm2 iff $AND3(      x2,      x1,NOT1 x0))&
($xm1 iff $AND3(      x2,      x1,      x0))&
($xz  iff $AND3(NOT1 x2,NOT1 x1,NOT1 x0))&
($xp1 iff $AND3(NOT1 x2,NOT1 x1,      x0))&
($xp2 iff $AND3(NOT1 x2,      x1,NOT1 x0))&
($xp3 iff $AND3(NOT1 x2,      x1,      x0))
```

Here, **$x0,$x1,$x2** express the three inputs of PART1 (Fig.3), **$xm3,$xm2,$xm1,$xz,$xp1,$xp2,$xp3** are all possible input states. The expression **$xm3 iff $AND3(x2,NOT1 x1,x0)** means that an input state is called **$xm3** if and only if inputs x2="1", x1="0", x0="1".

$ym3,$ym2,$ym1,$yz,$yp1,$yp2,$yp3 can be defined similarly.

OUTPUT STATE:

Output states can be defined in same way as follows.

```
($nz  iff $AND5(NOT1 nc1,NOT1 nc0,NOT1 nw2,NOT1 nw1,NOT1 nw0))&
($np1 iff $AND5(NOT1 nc1,NOT1 nc0,NOT1 nw2,NOT1 nw1,      nw0))&
($np2 iff $AND5(NOT1 nc1,NOT1 nc0,NOT1 nw2,      nw1,NOT1 nw0))&
($np3 iff $AND5(NOT1 nc1,      nc0,      nw2,      nw1,      nw0))&
($np4 iff $AND5(NOT1 nc1,      nc0,NOT1 nw2,NOT1 nw1,NOT1 nw0))&
($np5 iff $AND5(NOT1 nc1,      nc0,NOT1 nw2,NOT1 nw1,      nw0))&
($np6 iff $AND5(NOT1 nc1,      nc0,NOT1 nw2,      nw1,NOT1 nw0))&
```

Next, the behavior of PART1 can be described as a Boolean function as follows.

```
($nc0 iff $OR8(AND4(NOT1 x2,x1,NOT1 y2,y1),
      AND3(NOT1 x2,NOT1 y2,OR2(AND2(x1,x0),AND2(y1,y0))),
      AND3(NOT1 x2,NOT1 y2,OR2(AND2(x1,y0),AND2(x0,y1))),
      AND3(NOT1 x1,NOT1 y1,OR2(AND4(x2,x0,NOT1 y2,NOT1 y0),
      AND4(NOT1 x2,NOT1 x0,y2,y0))),
      AND5(x2,x1,y2,y1,NAND2(x0,y0)),AND5(x2,x1,y2,NOT1 y1,y0),
      AND5(x2,NOT1 x1,x0,y2,y1),AND6(x2,NOT1 x1,x0,y2,NOT1 y1,y0)))
```

$nc1,$nw2,$nw1,$nw0 can be described in same way.

Then, the relation between input status and output status is built.

The following expression is showed tautology by Mizar system. So the correctness of PART1 circuit is verified. (Fig.5)

```
($nm6 iff $AND2(xm3,ym3)) &
($nm5 iff $OR2(AND2(xm3,ym2),AND2(xm2,ym3))) &
($nm4 iff $OR3(AND2(xm3,ym1),AND2(xm2,ym2),AND2(xm1,ym3))) &
($nm3 iff $OR4(AND2(xm3,yz),AND2(xm2,ym1),AND2(xm1,ym2),
            AND2(xz,ym3))) &
($nm2 iff $OR5(AND2(xm3,yp1),AND2(xm2,yz),AND2(xm1,ym1),
            AND2(xz,ym2),AND2(xp1,ym3))) &
($nm1 iff $OR6(AND2(xm3,yp2),AND2(xm2,yp1),AND2(xm1,yz),
            AND2(xz,ym1),AND2(xp1,ym2),AND2(xp2,ym3))) &
```

```
($nz iff $OR7(AND2(xm3,yp3),AND2(xm2,yp2),AND2(xm1,yp1),
          AND2(xz,yz),AND2(xp1,ym1),AND2(xp2,ym2),AND2(xp3,ym3))) &
($np1 iff $OR6(AND2(xm2,yp3),AND2(xm1,yp2),AND2(xz,yp1),
          AND2(xp1,yz),AND2(xp2,ym1),AND2(xp3,ym2))) &
($np2 iff $OR5(AND2(xm1,yp3),AND2(xz,yp2),AND2(xp1,yp1),
          AND2(xp2,yz),AND2(xp3,ym1))) &
($np3 iff $OR4(AND2(xz,yp3),AND2(xp1,yp2),AND2(xp2,yp1),
          AND2(xp3,yz))) &
($np4 iff $OR3(AND2(xp1,yp3),AND2(xp2,yp2),AND2(xp3,yp1))) &
($np5 iff $OR2(AND2(xp2,yp3),AND2(xp3,yp2))) &
($np6 iff $AND2(xp3,yp3))
```

Here, for example, the expression **($nm5 iff $OR2(AND2(xm3,ym2), AND2(xm2,ym3)))** means a state is called **$nm5**(-5) if and only if the input state (**$xm3**(x=-3) and **$ym2**(y=-2)) or (**$xm2**(x=-2) and **$ym3**(y=-3)). Other expressions are similar.

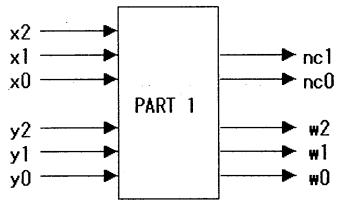
The correctness of PART2 can be verified in same way. (Fig.6)

Thus, the behavior of the whole circuit can be expressed by the following expressions.

```
($xm2 iff $AND3(      x2,      x1,NOT1 x0))&
($xm1 iff $AND3(      x2,      x1,      x0))&
($xz  iff $AND3(NOT1 x2,NOT1 x1,NOT1 x0))&
($xp1 iff $AND3(NOT1 x2,NOT1 x1,      x0))&
($xp2 iff $AND3(NOT1 x2,      x1,NOT1 x0))&
($cm  iff $AND2(      c1,      c0))&
($cz  iff $AND2(NOT1 c1,NOT1 c0))&
($cp  iff $AND2(NOT1 c1,      c0))&
($nm3 iff $AND3(      ns2,NOT1 ns1,      ns0))&
($nm2 iff $AND3(      ns2,      ns1,NOT1 ns0))&
($nm1 iff $AND3(      ns2,      ns1,      ns0))&
($nz  iff $AND3(NOT1 ns2,NOT1 ns1,NOT1 ns0))&
($np1 iff $AND3(NOT1 ns2,NOT1 ns1,      ns0))&
```

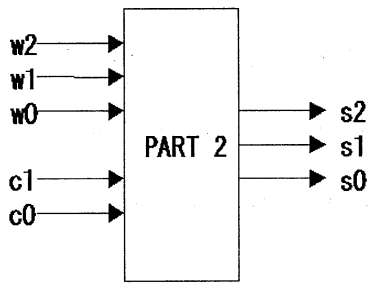
```
($np2 iff $AND3(NOT1 ns2,      ns1,NOT1 ns0))&
($np3 iff $AND3(NOT1 ns2,      ns1,      ns0))&
($ns0 iff $OR4(AND4(NOT1 x2,NOT1 x1,NOT1 x0,c0), AND3(x1,NOT1 x0,c0),
    AND5(NOT1 x2,NOT1 x1,x0,NOT1 c1,NOT1 c0),
    AND5(x2,x1,x0,NOT1 c1,NOT1 c0)))&
($ns1 iff $OR5(AND5(NOT1 x2,NOT1 x1,NOT1 x0,c1,c0),
    AND5(NOT1 x2,NOT1 x1,x0,NOT1 c1,c0),AND3(x1,NOT1 x0,NOT1 c1),
    AND5(x2,x1,x0,NOT1 c1,NOT1 c0),AND5(x2,x1,x0,c1,c0)))&
($ns2 iff $OR7(AND2(c1,NOT1 c0),AND4(NOT1 x2,NOT1 x1,NOT1 x0,c1),
    AND3(NOT1 x2,x1,x0),AND3(x2,x1,NOT1 x0),AND2(x2,NOT1 x1),
    AND3(x2,NOT1 c1,NOT1 c0),AND2(x2,c1)))

($nm3 iff $AND2(xm2,cm)) &
($nm2 iff $OR2(AND2(xm2,cz),AND2(xm1,cm))) &
($nm1 iff $OR3(AND2(xm2,cp),AND2(xm1,cz),AND2(xz,cm))) &
($nz iff $OR3(AND2(xm1,cp),AND2(xz,cz),AND2(xp1,cm))) &
($np1 iff $OR3(AND2(xz,cp),AND2(xp1,cz),AND2(xp2,cm))) &
($np2 iff $OR2(AND2(xp1,cp),AND2(xp2,cz)))&
($np3 iff $AND2(xp2,cp))
```

The tautology can be showed because part 1 and part 2 have been verified. So the correctness of radix-4SD adder circuit is verified.



nc1 = x2 * ~x1 * ~x0 + y2 * ~y1 * ~y0
+ ~x2 * ~x1 * ~x0 * y2 * ~y1 + x2 * ~x1 * ~y2 * ~y1 * ~y0
+ x2 * y2 * (~x1 + ~y1) + x2 * y2 * (x1 * ~x0 + y1 * ~y0)
nc0 = ~x2 * ~y2 * (x1 * x0 + y1 * y0)
+ ~x2 * ~y2 * (x1 * y0 + x0 * y1) + ~x2 * x1 * ~y2 * y1
+ ~x1 * ~y1 * (x2 * x0 * ~y2 * ~y0 + ~x2 * ~x0 * y2 * y0)
+ x2 * x1 * y2 * y1 * (x0 NAND y0)
+ x2 * x1 * y2 * ~y1 * y0 + x2 * ~x1 * x0 * y2 * y1
+ x2 * ~x1 * x0 * y2 * ~y1 * y0
w2 = ~x2 * ~y2 * (x1 XOR y1) * (x0 XOR y0)
+ x2 * x1 * ~y2 * ~y1 * (x0 NAND y0)
+ ~x2 * ~x1 * y2 * y1 * (x0 NAND y0)
+ x2 * ~x1 * y1 * ~y0 + x1 * ~x0 * y2 * ~y1
+ ~x1 * x0 * ~y1 * y0 * (x2 XOR y2)
+ x2 * x0 * y2 * y0 * (x1 XNOR y1)
w1 = (x1 XNOR y1) * (x0 * y0) + (x1 XOR y1) * (x0 NAND y0)
w0 = x0 XOR y0

Fig.5 The circuit verified of correctness by Mizar system (PART1)



s2 = c1 * ~c0 + ~w2 * ~w1 * ~w0 * c1
+ ~w2 * w1 * w0 + w2 * w1 * ~w0
+ x2 * ~x1 + x2 * ~c1 * ~c0 + x2 * c1

s1 = ~w2 * ~w1 * ~w0 * c1 * c0
+ ~w2 * ~w1 * w0 * ~c1 * c0
+ w1 * ~w0 * ~c1
+ w2 * w1 * w0 * c1 * c0
+ x2 * x1 * x0 * ~c1 * ~c0

s0 = ~w2 * ~w1 * ~w0 * c0 + x1 * ~x0 * c0
+ ~x2 * ~x1 * x0 * ~c1 * ~c0
+ x2 * x1 * x0 * ~c1 * ~c0

Fig.6 The circuit verified of correctness by Mizar system (PART2)

6. Conclusion

We showed it is possible to verify correctness of logical circuit's mathematical model using proof checker Mizar.

This can be considered as a new approach to verifying correctness of logical circuit.

The circuit we proved has been accept by the library of Mizar, and it can be used to prove larger circuits.

When the library is substantially in future, verification of logical circuits in practice can be expected. Verification of cryptogram circuit can be realized in same way.

References

[1] Mizar Project: http://mizar.org/project/

[2] Nakamura: "Logic Gates and Logical Equivalence of Adders", Volume 11,Journal of Formalized Mathematics 1999

[3]Yang,Wasaki,Fuwa,Nakamura: "Correctness of Binary Counter Circuits", Volume 11,Journal of Formalized Mathematics 1999