X-value Equivalence Checking

B07901075 李彥儒, B07901103 陳孟宏

Abstract

In order to check if the two given circuits are compatible equivalent, we try to test all the possible input patterns. And check the corresponding output bit patterns. To deal with the case of X-Value. We use a method of encoding all fanin bits into a string of two bits (“00”, “01”, “10”, but not “11”). Where the strings describe 0, 1, X respectively. We then build the new truth table for our definitions, transforming the original circuit into cnf form. Finally, performing SAT to obtain our results.

1.Introduction

We define ‘0’ = “00”, ‘1’ = “01”, ‘X’ = “10”, and “10” is don’t care term. Then from the definition of X-value and that of each gates (including :_DC GATE, MUX GATE), we construct truth table and simply the logic expression.

Since the input pattern doesn’t contain X-Value and that we have transformed X-Value at all FanIn and Fanout into Boolean expression, we simply apply the modules given by abc-master to perform SAT. We demonstrate the partial codes in the following.

2.File Reading

2.1 Input File Parsing

The test cases given are containing gate names (see Figure 1) and constant 0, const 1 at wire which disable the file reading function of abc-master. We thus come up with a program help to ignore the gate name when reading the input file.

Our idea is: upon reading the gate name, the program ignores all text before reading ‘ (‘. Then write the remaining text into a new file for abc-master to read.

Consequently, we create two files, “ReadGate.c” and “ReadGate.h” in “exteda” to do the work. (Figure 1.2)

```
buf \U$1abaj94 ( \O[10] , \693_Z[10] );
buf \U$1abaj95 ( \O[9] , \695_Z[9] );
buf \U$1abaj96 ( \O[8] , \697_Z[8] );
buf \U$1abaj97 ( \O[7] , \699_Z[7] );
buf \U$1abaj98 ( \O[6] , \701_Z[6] );
buf \U$1abaj99 ( \O[5] , \703_Z[5] );
buf \U$1abaj100 ( \O[4] , \705_Z[4] );
buf \U$1abaj101 ( \O[3] , \707_Z[3] );
buf \U$1abaj102 ( \O[2] , \709_Z[2] );
buf \U$1abaj103 ( \O[1] , \711_Z[1] );
buf \U$1abaj104 ( \O[0] , \713_Z[0] );
```

Figure 1.1

```
char* stringslice( char* str, size_t len )
{
  size_t i = 0, count = 0;
  int now = 0;
  char* aft_slice = NULL;
  for(i = 0; i < len; i++){
    if (now == 2)
    {
      aft_slice[0] = str[0];
      aft_slice[1] = str[1];
      aft_slice[2] = str[2];
      if (str[i] == '(') { aft_slice[count+3] = str[i]; ++count; }
      if (i == len-1) { return aft_slice; }
    }
    if (now == 3)
    {
      aft_slice[0] = str[0];
      aft_slice[1] = str[1];
      aft_slice[2] = str[2];
      aft_slice[3] = str[3];
      if (str[i] == '(') { aft_slice[count+4] = str[i]; ++count; }
      if (i == len-1) { return aft_slice; }
    }
    if (now == 4)
    {
      aft_slice[0] = str[0];
      aft_slice[1] = str[1];
      aft_slice[2] = str[2];
      aft_slice[3] = str[3];
      aft_slice[4] = str[4];
      if (str[i] == '(') { aft_slice[count+5] = str[i]; ++count; }
      if (i == len-1) { return aft_slice; }
    }
    // Gate name length = 2
    if (str[0] == 'o' && str[1] == 'r' && str[2] == ' ') { now = 2; }
    // Gate name length = 3
    if ((str[0] == 'b' && str[1] == 'u' && str[2] == 'f' && str[3] == ' ')
      ||(str[0] == 'n' && str[1] == 'o' && str[2] == 't' && str[3] == ' ')
      ||(str[0] == 'n' && str[1] == 'o' && str[2] == 'r' && str[3] == ' ')
      ||(str[0] == 'a' && str[1] == 'n' && str[2] == 'd' && str[3] == ' ')
      ||(str[0] == '_' && str[1] == 'D' && str[2] == 'C' && str[3] == ' ')
      ||(str[0] == 'x' && str[1] == 'o' && str[2] == 'r' && str[3] == ' ')) { now = 3; }
    // Gate name length = 4
    if ((str[0] == 'n' && str[1] == 'a' && str[2] == 'n' && str[3] == 'd' && str[4] == ' ')
      ||(str[0] == 'x' && str[1] == 'n' && str[2] == 'o' && str[3] == 'r' && str[4] == ' ')) { now = 4; }
    else { return str; }
  }
}
```

Figure 1.2

3.X-Value definition

3.1 Define X-Value

To deal with X-Values in the circuits. We define the bits as mentioned in Introduction (Also refer to figure 2)

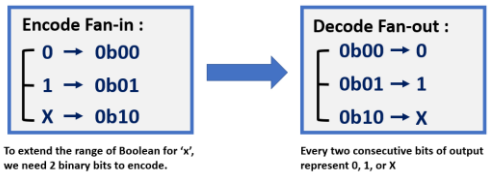


Figure 2

In “abc/abcAig.c”, we define Const1 and ConstX. Notice that we don’t have to define Const0, since Const0 = Abc_ObjNot(pConst1).

We then modify the codes in “src/base/abc/abcAig.c” to create the x-value node . And define Abc_AigConst in ”src/base/abci/abcVerify.c”. Finally in ”src/base/abc/abc.h” we define value-x.

```
// create the x-value node
assert( pNtkAig->vObjs->nSize == 0 );
pMan->pConstX = Abc_NtkCreateObj( pNtkAig, ABC_OBJ_NODE );
pMan->pConstX->Type = ABC_OBJ_CONSTX;
pMan->pConstX->fPhase = 1;
pNtkAig->nObjCounts[ABC_OBJ_NODE]--;
```

src/base/abc/abcAig.c

```
Abc_Obj_t * Abc_AigConstX( Abc_Ntk_t * pNtk )
{
    assert( Abc_NtkIsStrash(pNtk) );
    return ((Abc_Aig_t *)pNtk->pManFunc)->pConstX;
}
```

src/base/abci/abcVerify.c

```
Value0 = ((int)(ABC_PTRINT_T)Abc_ObjFanin0(pNode)->pCopy) ^ (int)Abc_ObjFaninC0(pNode);
Value1 = ((int)(ABC_PTRINT_T)Abc_ObjFanin1(pNode)->pCopy) ^ (int)Abc_ObjFaninC1(pNode);
ValueX = ((int)(ABC_PTRINT_T)Abc_ObjFaninX(pNode)->pCopy) ^ (int)Abc_ObjFaninCX(pNode);
pNode->pCopy = (Abc_Obj_t *)(ABC_PTRINT_T)(Value0 & Value1);
```

src/base/abc/abc.h

4. Gate Define

4.1define _DC GATE

From the given truth table in the problem. We add the _DC GATE into the gate library, and define the logic expression accordingly.

First, we build an object type to define constX in “abc.h”. Then build a node for cocnstX (Figure 3) in “abcAig.c”, where we also define pConst and AbcConstX.

Finally, we put _DC GATE into the library where abc-master defines its gates. (figure 4) And build the lookup table based on the problem. (figure5)

```
Abc_Obj_t * Abc_NtkCreateNodeConstX( Abc_Ntk_t * pNtk )
{
    Abc_Obj_t * pNode;
    assert( Abc_NtkIsLogic(pNtk) || Abc_NtkIsNetlist(pNtk) );
    pNode = Abc_NtkCreateNode( pNtk );
    if ( Abc_NtkHasSop(pNtk) || Abc_NtkHasBlifMv(pNtk) )
        pNode->pData = Abc_SopRegister( (Mem_Flex_t *)pNtk->pManFunc, " 10\n" );
#ifdef ABC_USE_CUDD
    else if ( Abc_NtkHasBdd(pNtk) )
        pNode->pData = Cudd_ReadOne((DdManager *)pNtk->pManFunc), Cudd_Ref( (DdNode *)pNode->pData );
#endif
    else if ( Abc_NtkHasAig(pNtk) )
        pNode->pData = Hop_ManConstX((Hop_Man_t *)pNtk->pManFunc);
    else if ( Abc_NtkHasMapping(pNtk) )
        pNode->pData = Mio_LibraryReadConstX((Mio_Library_t *)Abc_FrameReadLibGen());
    else if ( !Abc_NtkHasBlackbox(pNtk) )
        assert( 0 );
    return pNode;
}
```

Figure 3

```
// add logic function
if ( GateType == VER_GATE_AND || GateType == VER_GATE_NAND )
    pNode->pData = Hop_CreateAnd( (Hop_Man_t *)pNtk->pManFunc, Abc_ObjFaninNum(pNode) );
else if ( GateType == VER_GATE_OR || GateType == VER_GATE_NOR )
    pNode->pData = Hop_CreateOr( (Hop_Man_t *)pNtk->pManFunc, Abc_ObjFaninNum(pNode) );
else if ( GateType == VER_GATE_XOR || GateType == VER_GATE_XNOR )
    pNode->pData = Hop_CreateExor( (Hop_Man_t *)pNtk->pManFunc, Abc_ObjFaninNum(pNode) );
else if ( GateType == VER_GATE_BUF || GateType == VER_GATE_NOT )
    pNode->pData = Hop_CreateAnd( (Hop_Man_t *)pNtk->pManFunc, Abc_ObjFaninNum(pNode) );
else if ( GateType == VER_GATE_DC )
    pNode->pData = Hop_CreateOr( (Hop_Man_t *)pNtk->pManFunc, Abc_ObjFaninNum(pNode) );
if ( GateType == VER_GATE_NAND || GateType == VER_GATE_NOR || GateType == VER_GATE_XNOR || GateType == VER_GATE_NOT )
    pNode->pData = Hop_Not( (Hop_Obj_t *)pNode->pData );
return 1;
```

Figure 4

```
char * Abc_SopCreateX( Mem_Flex_t * pMan )
{
    return Abc_SopRegister(pMan, "0000 00\n0100 01\n0010 10\n0100 10\n0101 10\n0110 10\n1000 10\n1001 10\n1010 10\n");
}
```

Figure 5

4.2 define other gates

(1) MUX

Building MUX is similar to the procedure of building _DC GATE. But we have to discuss the case of S = 0, S = 1, S = X separately.

(2) native gates

Add X-Value to the truth table. (Figure 6)Derive the additional conditions, and update the lookup table. (Figure 7) Reconnect the new defined gate to the circuit. (Figure 8)

AND

	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

OR

	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

XOR

	0	1	X
0	0	1	X
1	1	0	X
X	X	X	X

XNOR

	0	1	X
0	1	0	X
1	0	1	X
X	X	X	X

NOR

	0	1	X
0	1	0	X
1	0	0	0
X	X	0	X

NAND

	0	1	X
0	1	1	1
1	1	0	X
X	1	X	X

DC

	0	1	X
0	1	0	X
1	0	1	X
X	X	X	X

BUF

IN	OUT
0	0
1	1
X	X

NOT

IN	OUT
0	1
1	0
X	X

MUX (s=0)

	0	1	X
0	0	1	X
1	0	1	X
X	0	1	X

MUX (s=1)

	0	1	X
0	0	0	0
1	1	1	1
X	X	X	X

MUX(s=X)

	0	1	X
0	0	X	X
1	X	1	X
X	X	X	X

Figure 6

```
Abc_Obj_t * Abc_AigXor( Abc_Aig_t * pMan, Abc_Obj_t * p0, Abc_Obj_t * p1 )
{
    Abc_Obj_t * pXor, * pConst1, * pConstX;
    unsigned Key;
    assert( Abc_ObjRegular(p0)->pNtk->pManFunc == pMan );
    assert( Abc_ObjRegular(p1)->pNtk->pManFunc == pMan );
    // check for trivial cases
    pConst1 = Abc_AigConst1(pMan->pNtkAig);
    pConstX = Abc_AigConstX(pMan->pNtkAig);
    // both in0 and in1 = 0 or 1
    if ((p0 == pConst1 || p0 == Abc_ObjNot(pConst1)) && (p1 == pConst1 || p1 ==
    {
        return Abc_AigOr( pMan, Abc_AigAnd(pMan, p0, Abc_ObjNot(p1)),
                                Abc_AigAnd(pMan, p1, Abc_ObjNot(p0)) );
    }
    // At least one input is x-value
    else
    {
        // One x-value will make all outputs be x-value
        if ((p0 != Abc_ObjNot(pConst1)) && (p0 != pConst1)) { return pConstX; }
        if ((p1 != Abc_ObjNot(pConst1)) && (p1 != pConst1)) { return pConstX; }
    }
```

Figure 6

```
{
    int nFans0 = Abc_ObjFanoutNum( Abc_ObjRegular(p0) );
    int nFans1 = Abc_ObjFanoutNum( Abc_ObjRegular(p1) );
    if ( nFans0 == 0 || nFans1 == 0 )
        return NULL;
}

// order the arguments
if ( Abc_ObjRegular(p0)->Id > Abc_ObjRegular(p1)->Id )
    pXor = p0, p0 = p1, p1 = pXor;
// get the hash key for these two nodes
Key = Abc_HashKey2( p0, p1, pMan->nBins );
// find the matching node in the table
Abc_AigBinForEachEntry( pMan->pBins[Key], pXor )
    if ( p0 == Abc_ObjChild0(pXor) && p1 == Abc_ObjChild1(pXor) )
    {
        assert( Abc_ObjFanoutNum(Abc_ObjRegular(p0)) && Abc_ObjFanoutNum(p1) );
        return pXor;
    }
return NULL;
```

Figure 7

5.cnf/SAT

5.1 cnf transformation

Upon defining _DC GATE, MUX, and X-Value, we use the modules in abc-master directly to transform the circuits into cnf form.

5.2 SAT

We then perform SAT on the cnf expression to see if the two given circuits are compatible equivalent.

6.Comparing Results

6.1 Output Message

After conducting SAT, we can determine whether the two circuits are compatible equivalent. We then print “EQ” or “NEQ” depending on the result.

6.2 trace back

Unfortunately, for the time being, we are unable to trace the input pattern that leads to the result of “NEQ”. Yet, we believe that the work can be done by inspection of “inputpattern”. We will keep working on it.

7. Experimental Results

7.1 a simple case

For debugging convenience, we use the example test case given by CAD contest to test our work. (figure8) The theoretical results is given by table1. From the table, the expected result is “NEQ”, and the counter examples are “0X” under the inputs “01” and “11”.

To test this case, we first write the given circuits into Verilog files, and read them into abc. Yet, due to some segmentation fault, we couldn’t do the work properly.

Furthermore, some input wires are

fed with constant input, which we could not deal with. We are still trying to parse such case.

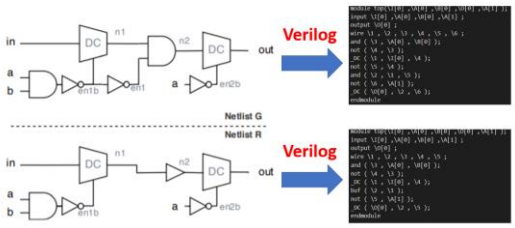


Figure8

In	a	b	G	R
0	0	0	X	X
0	0	1	X	X
0	1	0	0	X
0	1	1	0	0
1	0	0	X	X
1	0	1	X	X
1	1	0	0	X
1	1	1	1	1

Table1

8.Conclusions and Future Work

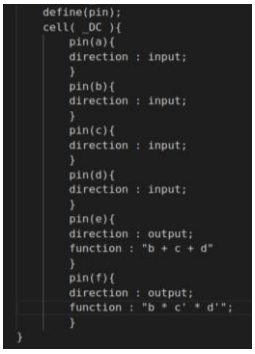
8.1 alternative encoding

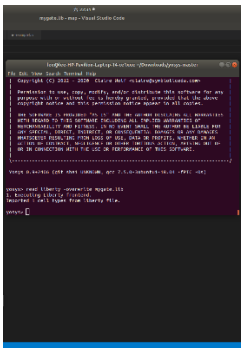
For the time being, we only tried to encode X-Value as “10”. But because of the existence of don’t care terms. Other encoding such as “11” is possible. Such encoding may make the circuit transformation easier. We can perform simplification using K-MAP to find our best choice.

8.2 tracing back input pattern

We will try to figure out how abc data structure store the input patterns. Then when a “NEQ” case occurs, we can trace back the input pattern immediately.

After final presentation, we also tried to use “Yosys” to read the test Verilog file as well as transform to blif file. Also, we re-define _DC gate and X-value in a liberty file. Yet, because the lack of time, we regretfully could not finish our work. We still handled the parsing error.





9. Job Division

陳孟宏:

(1) File Reading & Parsing
(2) Define DC GATE (coding)
(3) Modify Truth Table (coding)
(4) Test Case
(5) PPT

李彥儒:

(1) Define x-value
(2) Define DC GATE (derivation)
(3) Modify Truth Table (derivation)
(4) Equivalence Checking
(5) Report

10.Reference

(1) https://reurl.cc/D92EME
(2) https://reurl.cc/0oqR8b
(3) https://reurl.cc/X68yY7
(4) https://reurl.cc/R4VmV9
(5) https://reurl.cc/Qdqxq9
(6) https://reurl.cc/oLEg5j
(7) https://reurl.cc/NjKr4p