Sidik Nurcahyo. Int. Journal of Engineering Research and Application *www.ijera.com*
ISSN : 2248-9622, Vol. 7, Issue 7, (Part -5) July 2017, pp.01-05

RESEARCH ARTICLE OPEN ACCESS

General Algorithm for Testing the Combinational Logic Gates inside Digital Integrated Circuits

*Sidik Nurcahyo
Electrical Engineering, State Polytechnic of Malang, Indonesia 65145
Corresponding Author: Sidik Nurcahyo

ABSTRACT
This article describes general algorithm used to build a tester for combinational logic gate(s) inside a digital Integrated Circuit (IC). The challenges include how to handle different type of gate, variant number of input, and cascaded gates. Proposed solution is using common function for all types of digital IC by interpreting defined data abstraction for combinational logic gate(s) inside digital IC. This data abstraction is written in simple array of byte to present pin numbers and gates. The solution has been verified by simulation using ISIS Proteus and in real condition where the algorithm is implemented on AVR microcontroller ATmega32. The result show that the algorithm works successfully to test all types of combinational logic gates inside TTL IC and CMOS IC as well.
Keywords: algorithm, AVR microcontroller, combinational logic gate, digital IC, Proteus, test

Date of Submission: 13-07-2017 Date of acceptance: 15-07-2017

I. INTRODUCTION

In electronic world, digital Integrated Circuits (IC) needs to be tested properly before used in application. For the digital IC that only contains combinational logic gate(s), the testing is done simply by applying some logic states to the IC inputs and then evaluating the IC output based on the IC truth table. Someone has made fix or non-programmable IC tester [1]. Using programmable chip like microcontroller, this job can be handle easily as long as the microcontroller has been programmed with a firmware having correct testing algorithm. While testing, the microcontroller sends some logic states to IC inputs and then evaluates IC output. If the output matches with the value noted by IC truth table then microcontroller notifies the IC is good. Otherwise, the microcontroller says it is bad. Someone has proposed an algorithm for this job but it seems not as practical as described in this paper. He/She implemented algorithm for IC tester in the form of a function for every type of digital IC [2, 3]. While in this paper, testing algorithm is only implemented in one or two common function(s) for all types of combinational logic ICs, instead. Even though this design requires data abstraction for each IC being tested, it is more simple and extensible than using dedicated function per digital IC.

II. TYPES OF DIGITAL IC

Digital IC can be classified into two types, combinational and sequential. The first type does not require timing or clock, while the second does. The combinational digital IC only contains combination of logical gates, e.g. AND, OR, NAND, etc. While the sequential digital IC may contain memory and timing circuit to build special functions like flip-flop, timer/counter, shift-register, etc. This paper discusses development of general algorithm to test the combinational digital IC (TTL or CMOS). As mentioned earlier, this algorithm requires data abstraction for each digital IC. To formulate data abstraction accurately, it is necessary to explore logic diagrams of all combinational digital ICs. The following is several diagrams for selected combinational digital ICs. First is 74LS00 (TTL type) and 74HC00 (CMOS type). Both ICs contain four NAND gates where each gate has two inputs. These are packaged in 14 pins DIP, as shown in Figure 1.

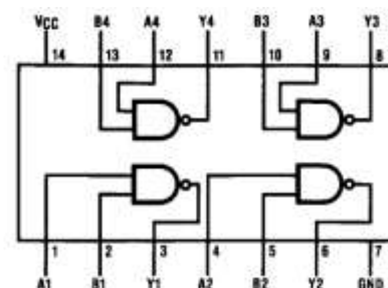

Figure 1. Diagram for 74LS00 or 74HC00 [4]

Similar to this IC is 74LS03 (TTL type) and 74HC03 (CMOS type) but its gate output is open collector and open drain, respectively. There is smaller version for 74HC00, i.e. 74HC1G00. It contains only one NAND gate with two inputs, as shown in Figure 2.

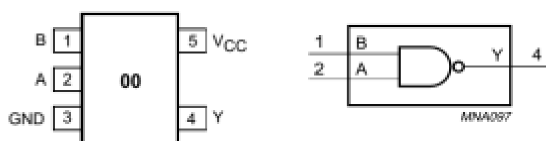

Figure 2. Diagram for 74HC1G00 [5]

Other important diagram of combinational digital IC is found in 74LS54 as shown in Figure 3.

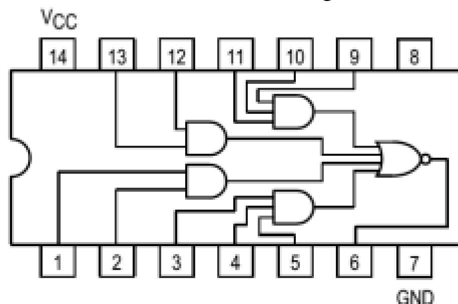

Figure 3. Diagram for 74LS54 [6]

This IC has only one output at pin 6 but rather complex as it has two stages of gate. First stage contains AND gates while the second stage contains a NOR gate. Gate outputs from first stage become gate inputs of the second stage. Similar to this configuration also can be found in 74HC51 but with two outputs. The last diagram is 74HC125 as shown in Figure 4.

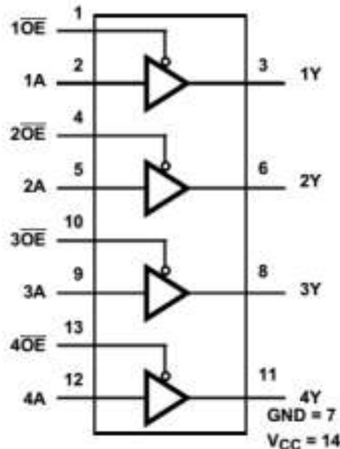

Figure 4. Diagram for 74HC125 [7]

This is a quad three state buffer where each buffer has an active low enable pin as depicted in the figure. Similar to this IC is 74HC126 but enable pin is active high, instead.

III. DATA ABSTRACTION

Exploration to selected logic diagrams results in three important points. First, IC package size varies from several pins to 14 pins. The Vcc pin is always at top right of the package while the Gnd pin is always at bottom left of the package. Second, number of gate and number of input per gate inside IC are not same between IC to IC. Third, an IC can contain some kind of gates and those gates may be cascaded.

Based on this result, it is proposed a format for data abstraction required to develop better testing algorithm, as shown in Figure 5.

```
{ic_id}
<pin_num>,
[<pin>,[EN | EN_NOT,]
<pin1>,<pin2>[,<pinx>],<gate1>,[NOT],<pin3 | X>,
<pin4>,<pin5>[,<piny>],<gate2>,[NOT],<pin6 | X>,
...
<gateN>,[NOT,][ OC,]<pin5>,
...
END
```

Figure 5. Data abstraction format

Using this format, data abstraction for some selected ICs can be written as shown in Table 1.

Table 1. Data Abstraction for some ICs

IC number	Data Abstraction
74LS00 or 74HC00	‘7’,’4’,’-’,’0’,’0’,0,14, 1,2,AND,NOT,3, 4,5,NAND,6, 9,10,NAND,8, 12,13, NAND,13,END
74LS03	‘7’,’4’,’L’,’S’,’0’,’0’,0, 14, 1,2,AND,NOT,3, 4,5,NAND,6, 9,10,NAND,8, 12,13, NAND,OC,13,END
74HC1G00	‘7’,’4’,’H’,’C’,’1’,’G’,’0’,’0’,0, 6 1,2,NAND,4,END
74HC51	‘7’,’4’,’H’,’C’,’5’,’1’,0,14, 1,13,AND,X, 9,10,AND,X, NOR,8, 2,3,AND,X, 4,5,X, NOR,6,END
74LS54	‘7’,’4’,’L’,’S’,’5’,’4’,0,14, 1,2,AND,X, 3,4,5,AND,X, 9,10,11,AND,X, 12,13,X, NOR,6,END
74HC125	‘7’,’4’,’H’,’C’,’1’,’2’,’5’,0,14, 1,NEN,2,BUF,3, 4,NEN,5,BUF,6, 10,NEN,9,BUF,8, 13,NEN,12,BUF,11,END

IC_id is a null terminated string, but written per character, to identify IC part number. This string is useful to identify IC part number when all data abstractions for various ICs are companied into single array. Using such array, it is possible to auto detect IC part number. The auto detection can be done by applying data abstraction one by one to IC being tested. If a particular testing is succeeded with a particular data abstraction then the string in current data abstraction is the part number for that IC.

A common way to start testing IC is by plugging the IC into ZIF socket of the tester. This ZIF socket makes loading and unloading IC easier. As can be seen in Table 1, that IC pin number is written as it is depicted in its diagram (not as socket

pin number). This is done to simplify the creation of data abstraction because it is not necessary to convert IC pin number to socket pin number manually. In turn, the pin conversion or pin mapping must be handled by software. If the IC tester is using 28 pins ZIF socket and the IC placement is bottom aligned, then pins mapping can be illustrated as in Figure 6.

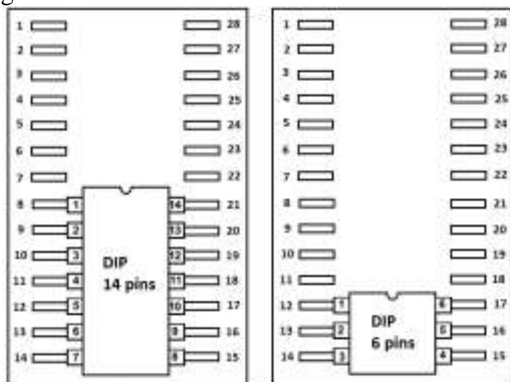

Figure 6. Pin mapping between IC and socket (28pin, bottom aligned)

As can be seen that IC Gnd is mapped to socket pin 14 and the IC Vcc is mapped to socket pin 21 (for DIP 14) or at socket pin 17 (for DIP 6). To convert IC pin number into socket pin number or microcontroller port/pin, the converter software needs to do two steps. First, read IC pin number from data abstraction then convert it to socket pin number using:

$$s = i + (28-n)/2 \quad (1)$$

Where: i = IC pin (read from data abstraction)
n =Pin_num (read from data abstraction)
s =socket pin (microcontroller pin)

Second, the software converts socket pin number to microcontroller port/pin by accessing array of structure (shown in Figure 7) at index s-1.

```
typedef struct{
   volatile unsigned char *port;
   unsigned char pin;
}pins_t;

pin_t socket[] = {
   {&PORTA,0},//socket pin 1
   ...
   {&PORTA,6},//socket pin 7
   {&PORTA,7},//socket pin 8
   {&PORTC,7},//socket pin 9
   {&PORTC,6},//socket pin 10
   {&PORTC,5},//socket pin 11
   {&PORTC,4},//socket pin 12
   ...
   {&PORTC,2},//socket pin 14
   {&PORTD,5},//socket pin 15
   ...
   {&PORTD,0},//socket pin 20
   {&PORTB,7},//socket pin 21
   ...
   {&PORTB,0},//socket pin 28
};
```

Figure 7. Array of microcontroller port/pin

Referring to "Fig. 6", "Fig. 7" and applying (1), IC pin number 1 in DIP 14 will result in socket pin number 8 (s=1+(28-14)/2=8) or microcontroller pins[7]={&PORTA, 7}. Similarly, IC pin number 1 in DIP 6 will result in socket pin number 12 (s=1+(28-6)/2=12) or microcontroller pins[11]={ &PORTC, 4}.

Tokens used in data abstraction and their meaning are listed in Table 2.

Table 2. Token used in data abstraction

Token	Remark
AND	and gate
OR	or gate
XOR	exclusive or gate
NOT	not gate
NAND	and gate
NOR	nor gate
NXOR	not exclusive or gate
BUF	buffer gate
NBUF	buffer not
EN	pin before EN is high enable pin,
NEN	pin before NEN is low enable pin
X	temporary output/input
OC	open collector
OD	open drain
END	end of data abstraction
ENDS	end of all data abstractions

Token is a byte constant, but it must be unique and greater than number of socket pin. This rule is to avoid token clashing with pin number or other token. If the number of socket pin is 28 then value for AND is 29, OR is 30, and so forth.

IV. BUILDING ALGORITHM

Data abstraction can be considered as byte collection ended with END token. Successive bytes that followed by gate and pin is called a chain. Testing an IC using defined data abstraction is done for every chain found in the data abstraction. Algorithm to test an IC using defined data abstraction can be designed as shown in Figure 8.

```
test an IC:
   for each byte in data_abstraction
      if byte is pin
         if next byte in [EN, END_NOT]
            none
         else if prev byte is gate
            data_max = 1<<input_num - 1
            for data in[0 to data_max]
               test chain with data
            reset input_num
         else
            increment input_num
```

Figure 8. Pseudo-code to test an IC

Each byte in selected data abstraction needs to be sweep and parsed up to END token. If the byte is a pin and previous byte is a gate then a chain is found. Testing a chain with particular data can be

done by calling other routine named test chain with data (see “Fig. 9”). The data is a value ranging from zero to two power number of input minus one. For instant, if number of input is three then data will be value ranging from 0 to 2^3-1 = 7 or in binary 000 to 111. Number of input is counted while parsing the data abstraction. Algorithm to test a chain with particular data is shown in Figure 9.

```
test chain with data:
 for each byte in chain
  if next byte is EN or EN_NOT
    activate gate enabler at prev byte
  else if byte is pin
    if previous byte is gate
      copy prev_val into val
    else if prev byte is not gate
      put data&mask to socket_pin(byte)
      shift left mask 1 bit
  else if byte is X
    copy prev_val to val
  else if byte is gate
    if prev byte is gate
      invert prev_val then copy to val
    else if prev byte is X
      get all val in X, operate as gate
      if byte in [NAND, NOR, NXOR]
        complement result
      save to val
  read output pin
  compare with val[i]
```

Figure 9. Pseudo-code to test chain with data

This routine declares array of byte named val[] to keep logic states sent to IC input pins and the result of particular gate(s). Length of this array equals to length of the chain. If test chain routine is called then all bytes in provided chain is parsed one by one until end of chain. When EN or NEN is found then an appropriate logic state will be sent to appropriate microcontroller pin to enable gate. When byte is a pin and previous byte is not a gate then save V to val[i] and set appropriate microcontroller pin with logic L. Value V and L depend on the result of AND logic operation between data and mask. If the result is not zero then V=0xff and L=1, otherwise V=0x00 and L=0. Mask is bit masking to know bit state in byte value of data. For instant, if number of input is three then mask in binary will be 001, 010 and 100. Again, appropriate microcontroller pin is calculated using (1) and accessing array in “Fig. 7” as described in previous subsection. Index i in array val[] is pointer to current position of chain and the position is incremented every reading a byte from that chain. When byte is a pin and previous byte is a gate or current byte is X then copy val[i-1] to val[i]. When byte is a gate and previous byte is a gate too then copy the complement of val[i-1] to val[i]. When byte is a gate and previous byte is X then subroutine will consider all val[] with index same as index of X in this chain, as input current gate. Test chain routine then calculate all these val[](s) based on current gate and save the result into val[i]. Finally test chain routine read physical gate output and compares it with calculated value. If both values are matched then subroutine indicates gate is good, otherwise gate is bad.

V. VERIFICATION AND RESULT

Simulation and real experiment has been made to verify proposed algorithm. The algorithm is implemented in C language and compiled with GNU AVR-GCC. Then resulting firmware is run on 8-bit AVR microcontroller named Atmega32 with internal clock source 8MHz. A single digit common anode seven segment display is provided to show the testing result. Testing is started by pressing a button after placing IC on the ZIF socket. If the IC being tested is 74HC51 or 74LS51 and the IC is in good condition then the display should show patterns ‘7’, ’4’, ’-‘, ’5’, ’1’, successively. On the other hand, if the IC being tested is in bad condition or its data abstraction not found, then display should shows patters ‘E’, ‘r’, ‘r’, successively. Situation when simulating the testing of 74LS54 using ISIS Proteus is shown in Figure 10.

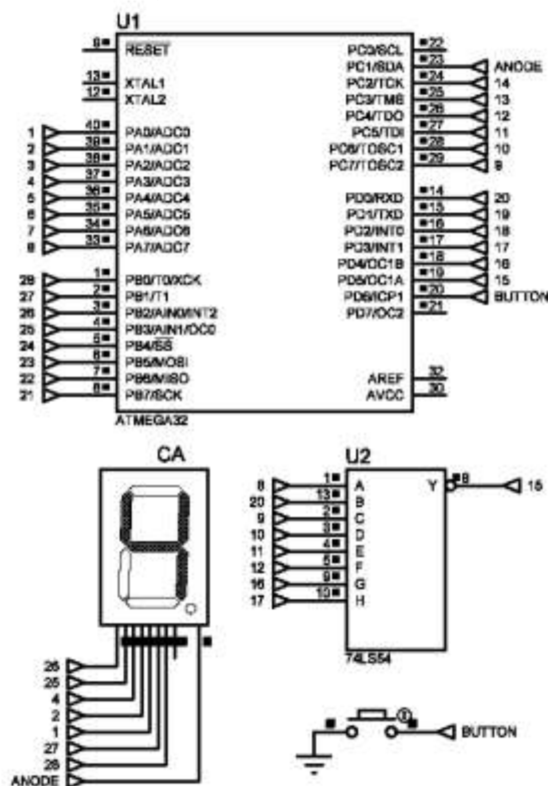

Figure 10. Simulation for 74LS54

Microcontroller pins labeled with 26, 25, 4, 2, 1, 27 and 28 are shared between ZIF socket and seven segment display. While testing an IC, these pins are fully used for ZIF socket. If testing is completed then these pins are used to drive the seven segment display. Situation when real testing of 74HC54 is shown in Figure 11.

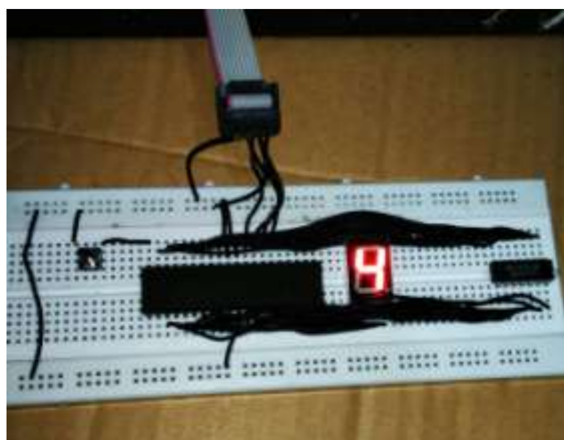

Figure 11. Real testing for 74LS54

Electronic connection for this real test is same as in simulation, except for IC Vcc and IC Gnd is controlled directly by microcontroller. In simulation, they are not controlled by microcontroller as they are hidden and automatically connected to Vcc and Gnd when simulation starts.

The real test is not using ZIF socket but project board. Plugging/Unplugging IC on project board is as easy as on ZIF socket but still requires a minus crew driver to help unplugging the IC. This is not a problem as it is just prototyping, not final product.

Testing result is summarized as in Table 2.

Table 2. Testing report for some ICs

IC part number	Number of IC under test	Remark*	
		good	bad
74LS00	10	10	0
74HC00	10	10	0
74LS03	10	8	2
74HC1G00	10	10	0
74HC51	10	9	1
74HC54	10	10	0
74HC125	10	7	3

*each IC is tested 3 times and shows the same result

This result verifies that proposed algorithm has worked as expected.

VI. CONCLUSION

Proposed algorithm has been developed and implemented successfully. The main advantage is that it does not require special function for each type of IC but only common function for all type of ICs with special data abstraction per IC. This design is more extensible than the one that requires dedicated function per IC. Data abstraction is suggested to be saved in updatable memory, like external EEPROM, or SD card, to facilitate enhancement of IC data abstraction. This algorithm does not support other type IC like comparator, encoder/decoder, and sequential IC. It seems more suitable if these ICs are handled using dedicated function per group of IC. So, comparator ICs has one common function for comparator, encoder/decoder has one common function, and so on.

REFERENCES

[1] Donald Farness Hanson, A Digital Integrated Circuit Tester, undergraduate manuscript, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1975.
[2] Yousif Taha Yousif Elamin, Abdelrasoul Jabar Alzubaidi, Microcontroller Based Integrated Circuit Tester, Journal of Engineering Research and Applications, 5(2), 2015, 118-122.
[3] Bhaskar Jyoti Borah, Rajib Biswas, Digital IC Tester with Embedded Truth Table, Research Gage: Electronics For You, https://www.resear chgate.net/publication/289534300, Jan 2016, 122-126.
[4] Fairchild Semiconductor™, MM74HC00 Quad 2-Input NAND Gate, www.fairchildsemi.com, 1999.
[5] Philips Semiconductor, Product specification of 74HC1G00/74HCT1G00: 2-input NAND gate, http://www.semiconductors.philips.com, 2002.
[6] Motorola, Fast and LS TTL Data: SN54/74LS54 3-2-2-3-Input AND-OR-INVERT Gate, 2003.
[7] Texas Instruments, High Speed CMOS Logic Quad Buffer Three-State: 54HC125, 74HC125, 54HCT125, 74HCT125, 2003.

International Journal of Engineering Research and Applications (IJERA) is **UGC approved** Journal with Sl. No. 4525, Journal no. 47088. Indexed in Cross Ref, Index Copernicus (ICV 80.82), NASA, Ads, Researcher Id Thomson Reuters, DOAJ.

*Sidik Nurcahyo. " General Algorithm for Testing the Combinational Logic Gates inside Digital Integrated Circuits." International Journal of Engineering Research and Applications (IJERA) 7.7 (2017): 01-05