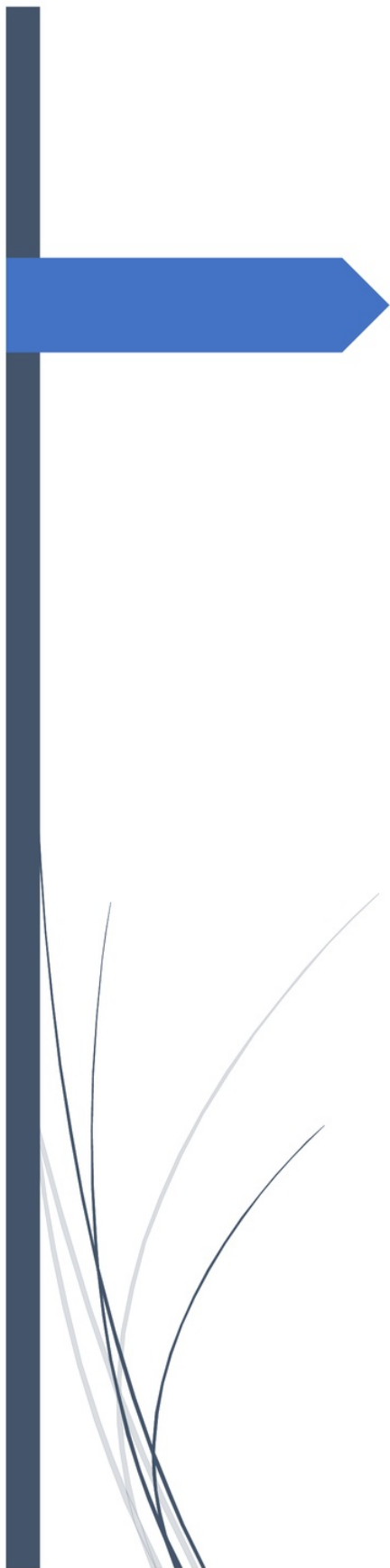


Ass1 final

by Michael Lewis

FILE	5293_MICHAEL_LEWIS_ASS1_FINAL_155126_838199843.DOCX (9.17M)		
TIME SUBMITTED	14-FEB-2018 08:46PM (UTC+0000)	WORD COUNT	5845
SUBMISSION ID	83191985	CHARACTER COUNT	38094



Assignment 1

PSDE6018 – Electronic production
and test

Michael Lewis
7/2/18

Contents

a. Table of figures	2
1. Introduction	4
2. Research.....	5
Surface mount technology.....	5
Two-layer PCB design.....	6
Soldering techniques for SMT	7
Soldering techniques for TH and SMT	8
Automatic x-ray inspection (AXI) and Automatic optical inspection (AOI).....	9
3. Design of assignment requirements and constraints	12
4. Analysis of NI Multisim results.....	17
5. Analysis of results using hardware descriptive language FPGA technology	24
6. Analysis of PCB design	36
7. Practical PCB build and testing of prototype on prototyping board	42
8. Conclusions and recommendations.....	46
9. References	47
10. Appendices.....	49
6bit hybrid wrapper.vhd	49
Hybrid.vhd.....	51

a. Table of figures

Figure 1- SMT pick and place	5
Figure 2- Two-layer PCB.....	6
Figure 3- Reflow temp profile	7
Figure 4- Wave soldering temp profile	8
Figure 5- Where PCBs should be checked.....	10
Figure 6- Automatic inspection comparison	11
Figure 7- Rule 90	12
Figure 8- Rule 150	13
Figure 9- CAR rule 90	14
Figure 10- CAR rule 150	14
Figure 11- 4-bit hybrid	15
Figure 12- 4-bit hybrid car truth table	15
Figure 13- Multisim 4-bit hybrid	16
Figure 14- 6-bit hybrid MS	17
Figure 15- 6-bit hybrid CAR max count table	18
Figure 16 - Multisim Car/Boolean comparison	19
Figure 17 - CAR/Boolean analyser output.....	20
Figure 18- MS fault 1.....	20
Figure 19- MS fault 1 waveform	21
Figure 20- MS fault 2.....	21
Figure 21- MS fault 2 waveform	22
Figure 22- MS fault 3.....	22
Figure 23- MS fault 3 waveform	23
Figure 24- MS faults summary table.....	23
Figure 25- CAR 90.....	24
Figure 26- CAR 150.....	24
Figure 27- 4-bit hybrid VHDL.....	25
Figure 28- VHDL 6-bit hybrid max count waveform	26
Figure 29- VHDL 6-bit hybrid count table	27
Figure 30- VHDL hybrid 6 bit	28
Figure 31- 6-bit hybrid VHDL.....	29
Figure 32- 6-bit hybrid VHDL.....	29
Figure 33- VHDL 4 input and gate	30

Figure 34-VHDL 2 input or gate.....	30
Figure 35- VHDL inverter (NOT gate)	30
Figure 36- VHDL 2 input XOR gate	31
Figure 37- VHDL clock divided to 5KHz	31
Figure 38-FPGA fault 1	32
Figure 39- FPGA fault 2	33
Figure 40- FPGA fault 3	34
Figure 41- FPGA test results.....	35
Figure 42- Eagle logic	36
Figure 43- Eagle routing.....	37
Figure 44 - eagle .cmp.....	37
Figure 45 - eagle .drd	38
Figure 46- eagle .plc.....	38
Figure 47- eagle .pls	39
Figure 48- eagle .sol	39
Figure 49- eagle .stc	40
Figure 50- eagle combined files	41
Figure 51- PCB.....	42
Figure 52- FPGA PCB test	42
Figure 53- FPGA PCB fault.....	43
Figure 54 - 74ALS21	44
Figure 55- 74AC11086.....	45
Figure 56- SN74LS32N.....	45
Figure 57- 7404	45

1. Introduction

In this assignment, various modern methods of producing printed circuit boards (PCB's). The modern technology of the components will also be investigated, along with the newest techniques used to mount the components to the board.

The way that the PCB's are automatically inspected will be investigated, such as; Automatic optical inspection (AOI) and Automatic X-ray Inspection (AXI).

Various examples of pseudo random pattern generators will be investigated, simulated and built practically. Very-high speed hardware descriptive language (VHDL) will be used to program the pseudo random pattern generator onto an FPGA chip, which will be tested for functionality.

All circuits (simulation, practical, PCB, VHDL) will then be constructed on a custom PCB. All versions of the circuit will be compared with each other to ensure they all function the same way. Some of the manufacturing techniques researched will be used to construct the PCB version of the circuit, such as; PCB design, through-hole components, surface mount components, reflow techniques etc.

A test procedure will be created to fully test the circuits for functionally. Faults will be simulated on the physical build and compared with working circuits (FPGA, for example), then simple logic should be able to show the fault when compared with a working version of the circuit.

Good introduction which explains the task ahead.

2. Research

Surface mount technology

SMT components are functionally identical to their passive through hole (PTH) equivalent, with the benefit of being magnitudes smaller in size than PTH. SMT is most commonly processed by machines, due to their small size and even smaller terminals. Due to the size and weight difference, much higher density of components can fit onto a PCB using SMT. Figure 1 shows a pick and place robot automatically placing components onto a PCB.



Figure 1- SMT pick and place

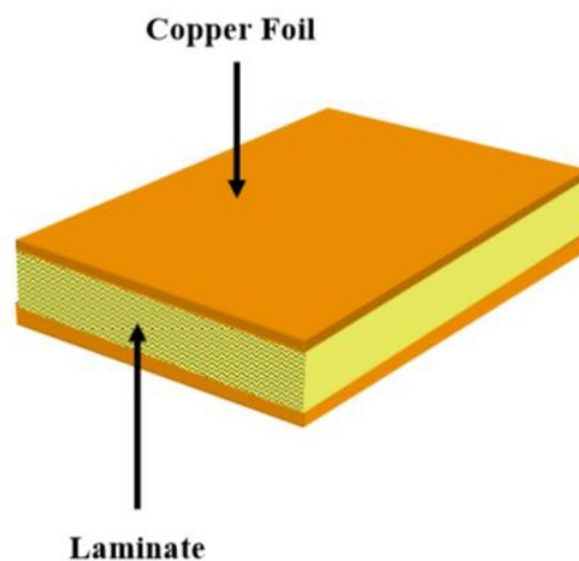
(Surface Mount Technologies., 2018)

Two-layer PCB design

The two layered PCB allows designs to be more compact and efficient by utilising both sides of the board with different tracks. An electronics service company states that the development of multi-layered PCB's allows "the designers to produce highly complex and compact circuits and further development of blind and buried via hole technology has pushed these limits even further" (Clarydon Electronic Services Limited , 2018). The same company offers a layer count of up to 32 on their PCB's!

Figure 2 shows how a simple 2-sided board is constructed. A rigid insulator is sandwiched in between two thin sheets of copper, creating the base that the circuit can be 'printed' onto.

One method of 'printing' a circuit onto a PCB is by using light sensitive film, ultraviolet light, and a negative image of the tracks that are being etched. The negative (where tracks are 'hollow') is laid on top of the copper and then exposed to UV light. The UV light hardens the film which leaves only film where the tracks are supposed to be. The board then goes through a chemical process that removes any copper which is not coated in film. This leaves the copper tracks on the board along with the hardened film which must be removed from the copper tracks. (Clarydon Electronic Services Limited , 2018)



(Advanced Circuits Inc, 2009)

Figure 2- Two-layer PCB

Soldering techniques for SMT

The most common soldering technique for SMT components is reflow soldering:

The circuit board needs to be raised up to a present temperature very slowly. The heat in the PCB must be even to avoid warping. The heating is usually only 2°C/second per second. (process, 2015)

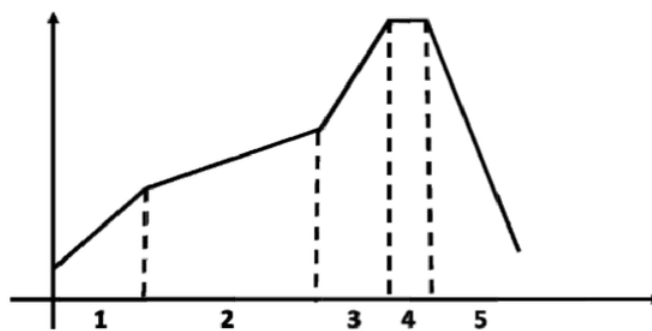
The SMT components are then placed, usually by means of a robot or machine.

A soak process then begins which ensures the board and components are up to the correct temperature.

The oven then rises to the temperature which solder becomes a liquid. Once the solder becomes a liquid it flows connecting the components and the pads they were placed on. Usually this stage only lasts about a minute to avoid overheating which causes dry/brittle joints. (Bergenthal, 1997)

The final stage is cooling. The entire board must be slowly cooled by no more than 3°C/second.

The stages explained above are shown in the temperature profile in Figure 3; Each number above the graph represents the different stages of the reflow process.



Profile Feature	Requirement
1. Ramp up	1-4 °C/second
2. Soak	150 °C~200 °C 60-180 seconds
3. Ramp up to peak	1-4°C/second
4. Peak soak *	245~260 °C 10 seconds max
5. Ramp-down Rate	1~6 °C/second max.

Figure 3- Reflow temp profile

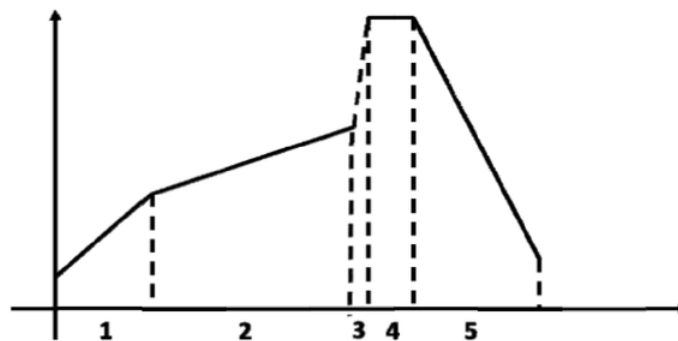
(ALPHA & OMEGA SEMICONDUCTOR, INC. , n.d.)

Soldering techniques for TH and SMT

The most common type of soldering technique for TH and SMT components is wave soldering: Machines can be bought which can automatically place the PTH components, with a wave soldering machine to solder the legs onto the track. However, PTH components are still being placed and soldered by hand by modern manufacturers (Kasdon Electronics, 2018). A combination of machine and man-made can be combined with the PTH components due to their relatively large size, unlike the microscopic SMT components which would be much more challenging to place and solder by hand.

The simplified process is:

- SMT components are glued on PCB with pick and place robot
- PTH components are placed using a pick and place robot, or by hand
- PCB goes through machines that; flux, preheat and apply solder
- A corresponding thermal profile is set (works in similar way to reflow)



Profile Feature	Recommended Condition
1. Preheat - Ramp up rate	1-7 °C/second
2. Soak - Temperature: - Time:	110°C ramp to 140°C 60-120 seconds
3. Ramp up - Ramp up rate	~150 °C/sec
4. Peak - Peak package body temperature - Time	245 °C to 260 °C 10 seconds max. For dual wave also
5. Ramp down: - Ramp down rate:	1-7 °C/second

Figure 4- Wave soldering temp profile

(ALPHA & OMEGA SEMICONDUCTOR, INC. , n.d.)

The two graphs appear to be very similar when comparing the temperature profiles for wave and reflow soldering. The big difference is the temperature setpoints and allowed rate-of-change of the temperature. The wave soldering temperature profile shows very steep gradient at stage 3, compared with reflow soldering. The reason that the profiles are different is due to the different processes that occur in each of the soldering methods.

But fundamentally, the temperature profile just tells the machine what temperature it wants, and how quickly it is 'allowed' to rise or fall to the setpoint temperature.

Automatic x-ray inspection (AXI) and Automatic optical inspection (AOI)

As the name suggests, AXI is the process of using x-ray images to get a detailed image of how the components on a board are connected and orientated. The benefit of using this type of technology is that it is not invasive, and a PCB that has had an AXI can still be used afterwards.

A manufacturer of this technology states another uses of their product: "IC inspection using 3D X-Ray imaging; also used for identification of counterfeit components." (Retronix Ltd., 2018) Which is not always obvious with the naked eye as counterfeiters can copy an actual manufacturer logo and etch it into the front of the fake chip.

Below is a list of some of the features a different manufacture offers on their latest machines:

PCB faults that AXI/AOI machine can detect:

Insufficient solder	Excess solder	Missing solder
Solder bridging/short circuit	Tombstone effect	Lifted lead
Solder defect	Solderability	Contamination
Missing component	Polarity error	Displaced component
Rotation	Component damaged	Incorrect component
Face down component	Component on its side	Excess components
Form defects	Twisted pin	Damaged pin
THT filling degree	Voids in the solder joint	BGA head in pillow

(Viscom AG, 2018)

Although this section is titled AXI, the modern machines that companies are producing are capable of both AXI and AOI.

AOI is a similar process to AXI but uses visible light instead of X-rays to inspect PCB's. The manufacture of these product explains how they achieve this: "Using advanced optics, algorithms and data processing capabilities to detect, and categorize defects" (Orbotech, 2018).

The diagram in Figure 5 shows the locations on the production line where the PCBs should be checked. Checking stations are placed after each major manufacturing process to enhance production quality.

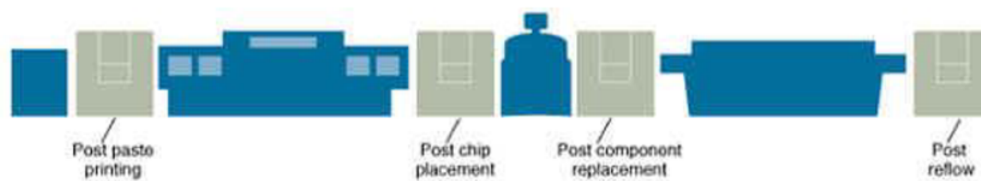


Figure 5- Where PCBs should be checked

(AspenCore, 2001)

1. Post paste printing – Solder defects can occur when the paste is being applied (smearing, bridging, skewing, insufficient paste etc.) It's important to monitor after this early process as this will affect most of the leading processes. 2D processes can find these problems with the 3D machines benefiting from being able to calculate volume of paste.
2. Once the chips are placed, the equipment can check for missing components, skewed components and can also check solder paste on each pad.
3. This process is identical to number 2 and checks for the same things previously mentioned.
4. The post reflow is the final inspection which checks: All components orientation and placement, polarity defects, missing components and checks to ensure the solder joints are not weak or sub-par (bridges, dry joints, insufficient solder, lifted leads).

A table of comparison has been put together on Excel using the referenced sources. The column marked "ICT" or 'in circuit testing' is the process of testing the PCB circuit once its built, hence the name in circuit testing. Points of contact or test points are built into the PCB to allow this test to be performed. A human could test these circuits using test equipment, or machines are used with a 'bed-of-nails' to allow the machine to make contact with the test points. The circuit tests are then performed automatically as the machine now has electrical access to the circuit its testing.

Defect type	AOI	AXI	ICT
Soldering defects			
Open circuits	Y	Y	Y
Solder bridges	Y	Y	Y
Solder shorts	Y	Y	Y
Insufficient solder	Y	Y	N
Solder void	N	Y	N
Excess solder	Y	Y	N
Solder quality	N	Y	N
Component defects			
Lifted lead	Y	Y	Y
Missing component	Y	Y	Y
Misaligned or misplaced component	Y	Y	Y
Incorrect component value	N	N	Y
Faulty component	N	N	Y
BGA defects			
BGA shorts	N	Y	Y
BGA open circuit connections	N	Y	Y

Figure 6- Automatic inspection comparison

3. Design of assignment requirements and constraints

One way to generate pseudorandom patterns is a linear feedback shift register. The initial value loaded into the LFSR is called the seed value and the register is said to be MAXIMAL LENGTH if $2^n - 1$ numbers are generated before repetition. But these circuits show significant correlation between successive patterns, as might be expected because of shifting, with the outputs also being jumbled.

A 6-bit hybrid cellular automata register (CAR) can overcome these drawbacks, one must be designed to give a pseudorandom pattern of numbers. The CAR topologies have various sets of rules which dictate how the circuit is connected. Rules 90 and 150 will have to be used to create a circuit that outputs numbers, has a maximum length (0-63) with no repeats. The logical component used in the CAR circuits is the XOR. The output is obtained by 'XOR'ing the flip-flop outputs either side. Rule 90 is shown on Figure 7 and rule 150 is shown in Figure 8. The names for each rule correspond to the binary value of the outputs of the XOR gates.

If either of these rules are used on their own, the maximum length (maximum count) will not be achieved. To achieve max length, rules 90 and 150 must be used together in the following order: 90, 150, 90, 150.....etc. A rule 90 is achieved by XORing each neighbours state, with the 150 being achieved by XORing its neighbours state and its own output.

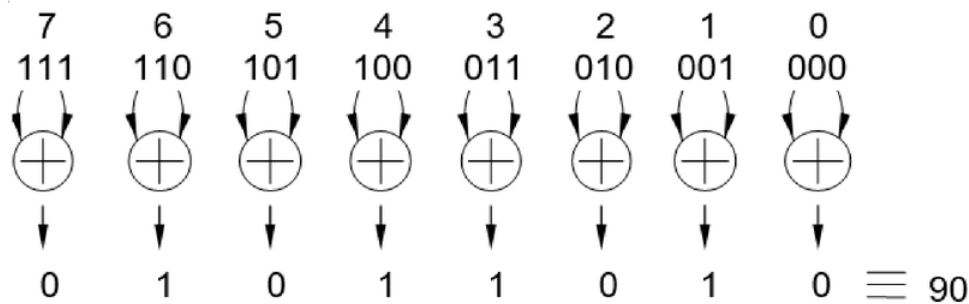


Figure 7- Rule 90

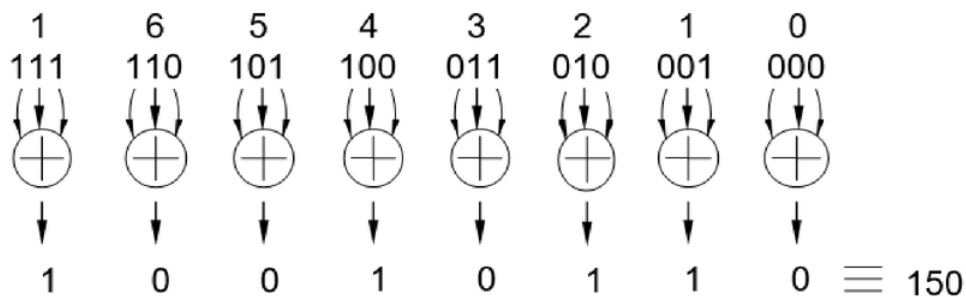


Figure 8- Rule 150

Simple CAR circuits have been designed on Multisim to prove that the maximum length cannot be achieved with these rules on their own. Rule 90 is shown in Figure 9. The circuit gives a count of 1,2,3,8,4,10 before repeating itself. Obviously not a maximum count circuit. The same circuit was setup to test rule 150, shown in Figure 10. The circuit gave an output of 1, 3, 4, 14, 5, 13 before repeating itself, another non-maximum length circuit.

The 4 bit hybrid car circuit is shown in Figure 11. Theory says that this circuit should produce a maximum length. The circuit is configured as: Rule 90, Rule 150, Rule 90, Rule 150. The table in Figure 12 shows the output of the hybrid circuit. The table proves that the combination of rule 90 and 150 in the correct order will give a maximum length count.

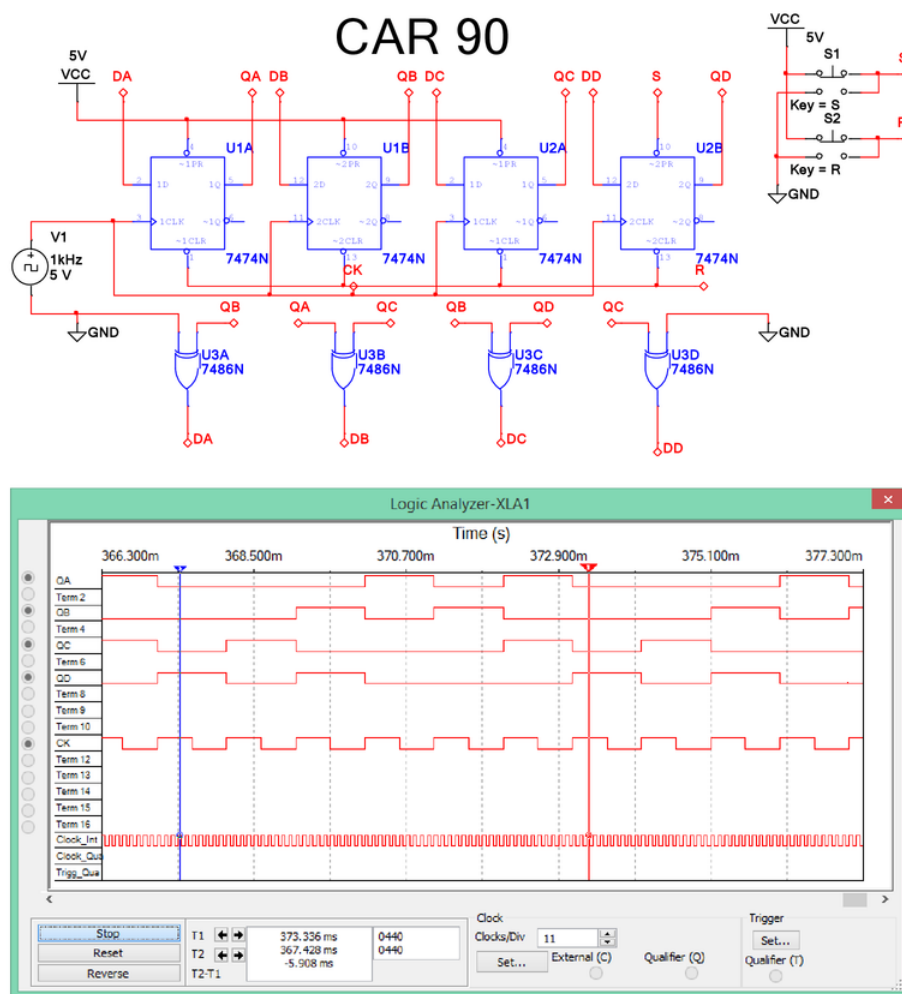


Figure 9- CAR rule 90

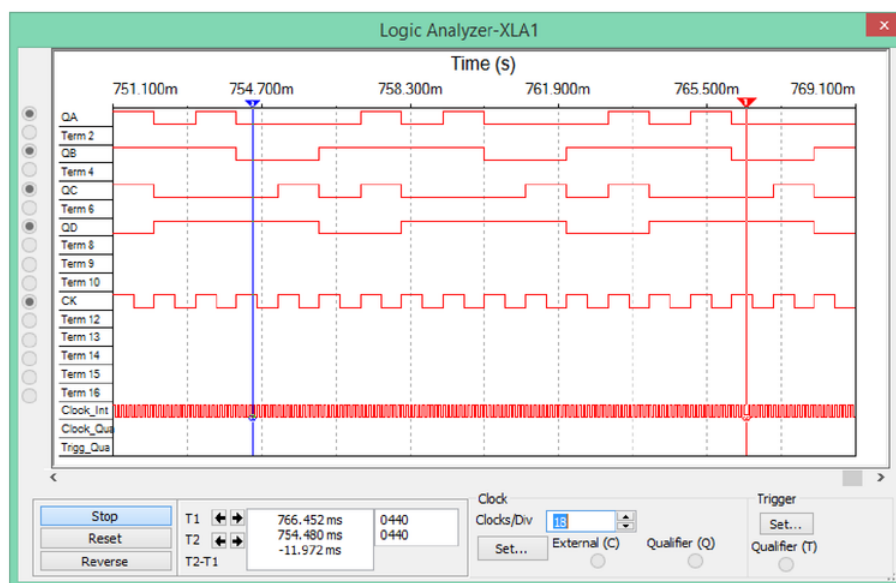
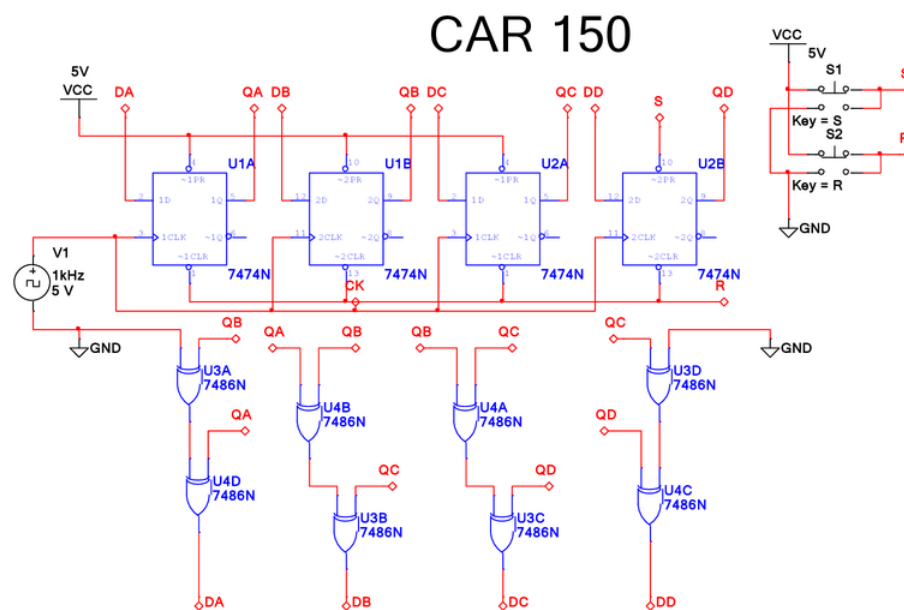


Figure 10- CAR rule 150

The 4 bit hybrid car circuit is shown in Figure 11. Theory says that this circuit should produce a maximum length. The circuit is configured as: Rule 90, Rule 150, Rule 90, Rule 150. The table in Figure 12 shows the output of the hybrid circuit. The table proves that the combination of rule 90 and 150 in the correct order will give a maximum length count. The two end points are null boundary conditions and connected to ground.

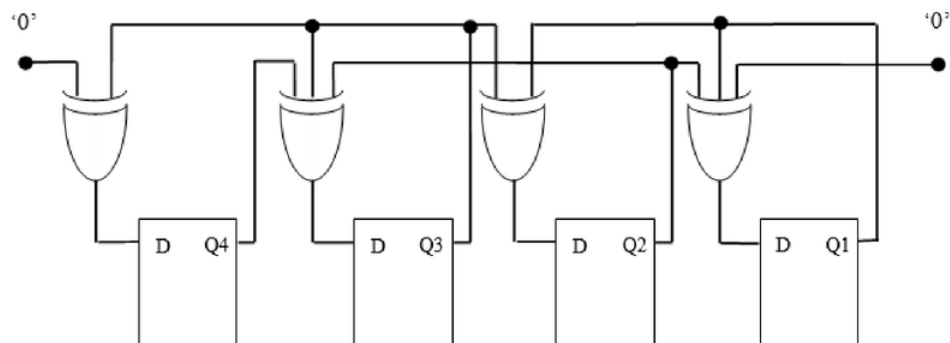


Figure 11- 4-bit hybrid

Q4	Q3	Q2	Q1	Decimal Value
0	0	0	1	1
0	0	1	1	3
0	1	1	0	6
1	0	1	1	11
0	0	1	0	2
0	1	0	1	5
1	1	0	1	13
1	0	0	1	9
0	1	1	1	7
1	0	0	0	8
0	1	0	0	4
1	1	1	0	14
1	1	1	1	15
1	1	0	0	12
1	0	1	0	10
0	0	0	1	1

Figure 12- 4-bit hybrid car truth table

The CAR prototypes are tied in with theory very well.

A 4-bit hybrid circuit was then designed on Multisim to prove the theory in simulation, see Figure 13. The circuit produced the following count: 1, 3, 6, 11, 2, 5, 13, 9, 7, 8, 4, 14, 15, 12, 10 before repeating itself. As expected, the hybrid circuit produces a maximum count. To meet the initial specifications for the assignment, the circuit below will be modified to be a 6-bit hybrid that produces a maximum count.

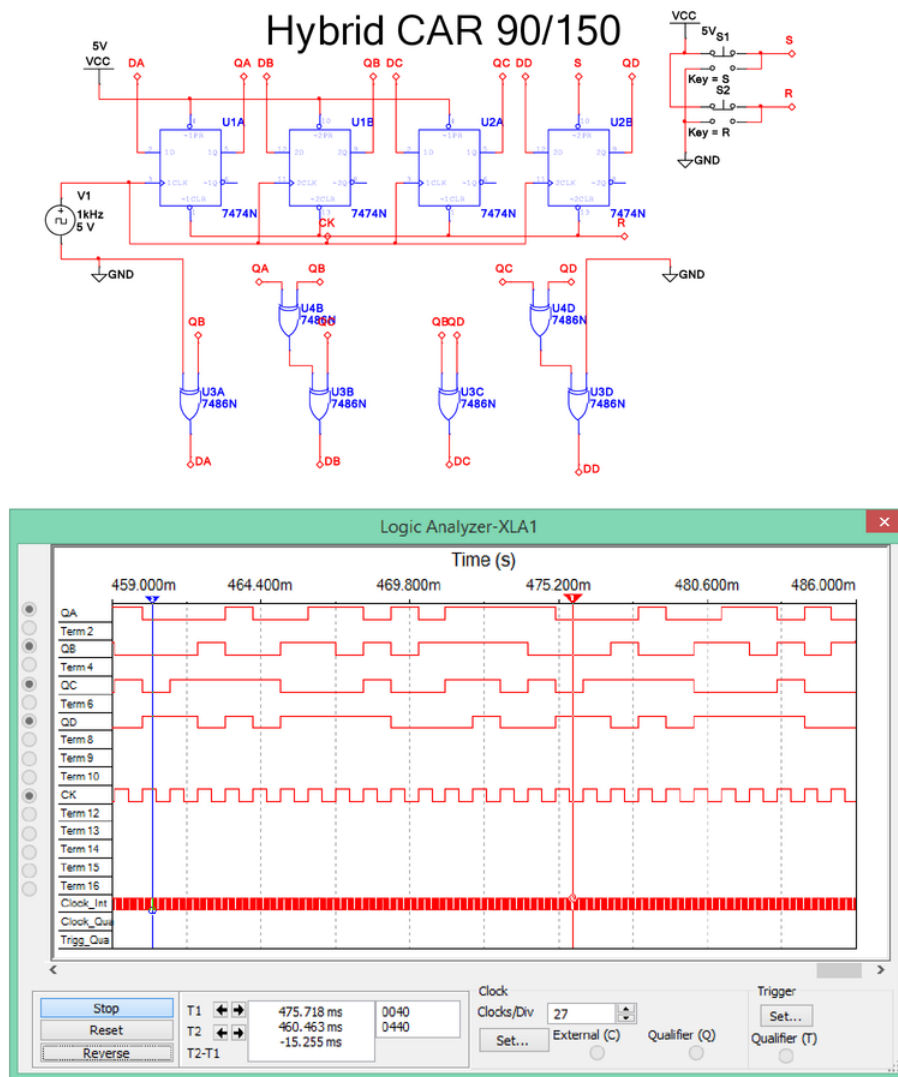


Figure 13-Multisim 4-bit hybrid

Good
analysis of 4
bit CAR to
prove the
maximum
count.

4. Analysis of NI Multisim results

The 4-bit hybrid circuit was modified to include 2 more bits. To adhere to the specification, the count must be max length, which can be achieved using a mix of rule 90 and 150. A screenshot of the circuit and the logic output is shown in Figure 14. The circuit was developed by adding another rule 90 and rule 150. The easiest way to differentiate the two on the schematic is: Rule 90 has 2 inputs and the rule 150 has 3. The logical output shows the count of the circuit, the circled part of the graph shows one count cycle, with the arrow pointing at 1. The logic analyser output has been studied and the results put into a table for scrutiny. The results gathered proves that as theory suggested, a maximum count circuit can be built using a combination of rule 90 and rule 150. The table clearly shows that there are no repeats until the full count has been achieved.

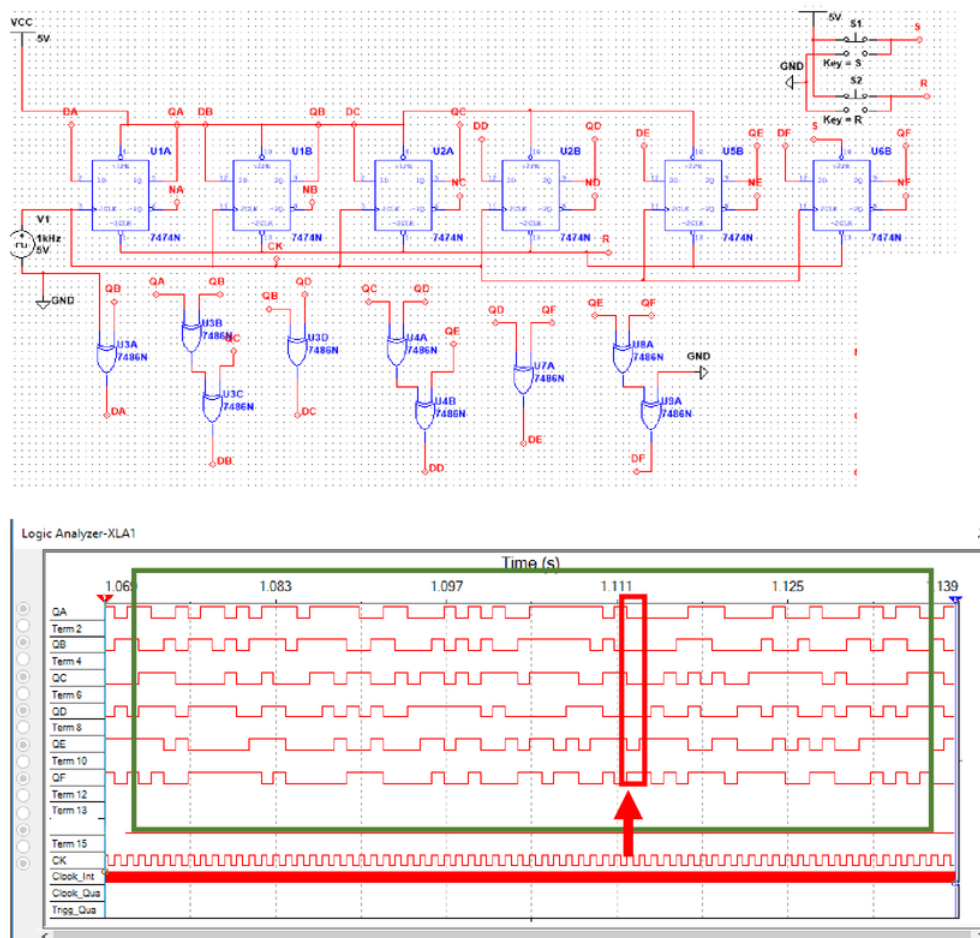


Figure 14- 6-bit hybrid MS

1	00000001		19	00010011
3	00000011		62	00111110
6	00000110		55	00110111
11	00001011		32	00100000
18	00010010		16	00010000
61	00111101		56	00111000
49	00110001		60	00111100
43	00101011		50	00110010
2	00000010		45	00101101
5	00000101		9	00001001
13	00001101		23	00010111
25	00011001		48	00110000
47	00101111		40	00101000
12	00001100		4	00000100
26	00011010		14	00001110
41	00101001		31	00011111
7	00000111		36	00100100
8	00001000		30	00011110
20	00010100		39	00100111
54	00110110		24	00011000
35	00100011		44	00101100
22	00010110		10	00001010
51	00110011		17	00010001
45	00101101		59	00111011
15	00001111		58	00111010
28	00011100		57	00111001
34	00100010		63	00111111
21	00010101		52	00110100
53	00110101		38	00100110
37	00100101		27	00011011
29	00011101		42	00101010
33	00100001		1	00000001

Figure 15- 6-bit hybrid CAR max count table

$$NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F$$

Good and

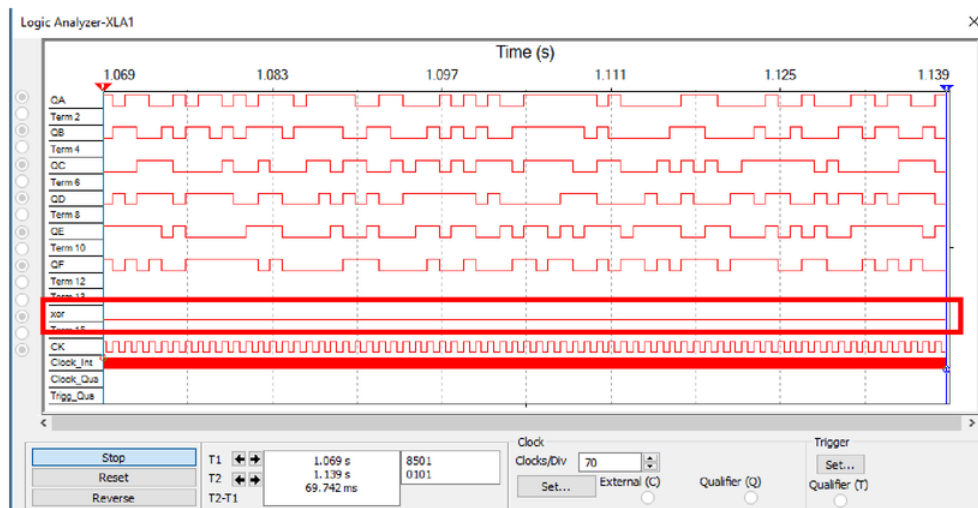


Figure 17 - CAR/Boolean analyser output

Now that the circuit has proved to be correct, it can now be used to compare against other potentially faulty circuits. The first fault to be tested is grounding the 'F' pin. The following Boolean expression defines a working circuit:

$$NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F$$

If a different Boolean expression is compared with the working circuit, a fault should be detected.

Figure 13 shows the circuit with the fault pin highlighted.

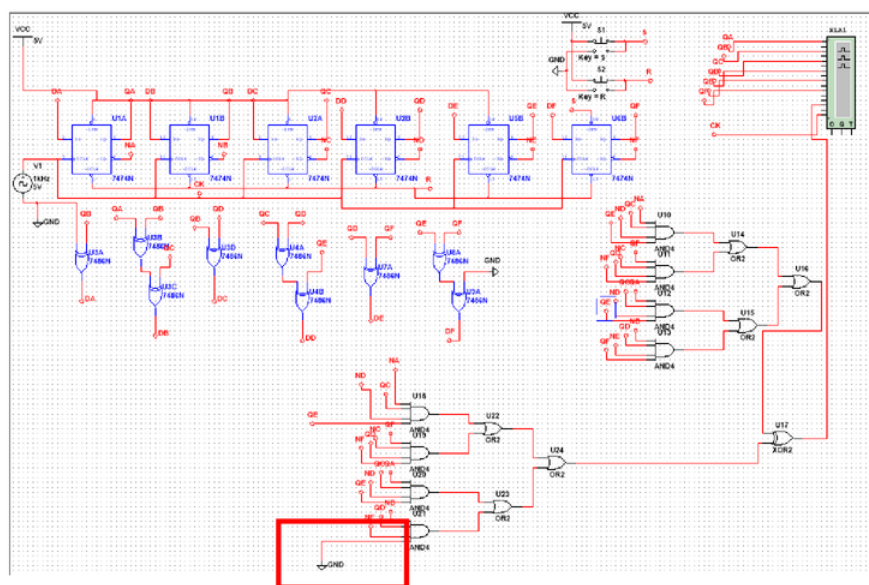


Figure 18- MS fault 1

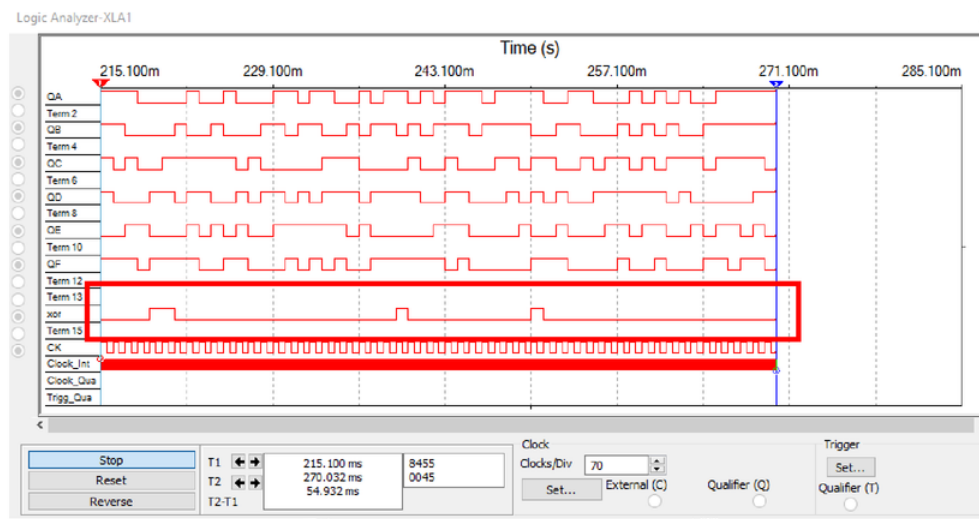


Figure 19- MS fault 1 waveform

The waveform in Figure 19 clearly shows the XOR gate giving an output at a few points in the waveform. The original circuit successfully detected the fault.

The second fault that is being tested is the A pin grounded. The test has the exact same format as the previous test. See Figure 20 for the circuit diagram.

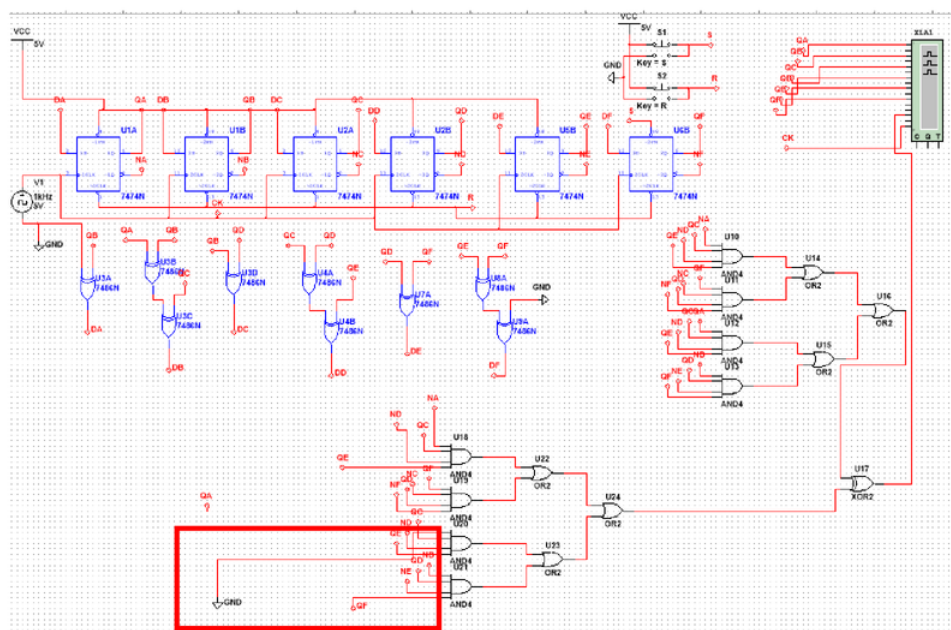


Figure 20- MS fault 2

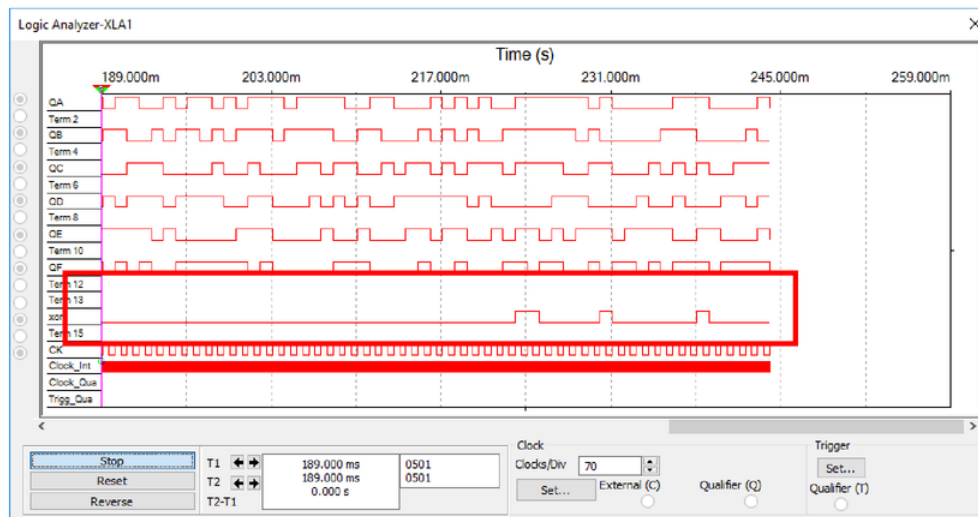


Figure 21- MS fault 2 waveform

Figure 21 clearly shows the faults being detected on the second circuit, shown by the XOR gate outputting.

The final test to be carried out is Vcc on Qe, as shown in Figure 22 .

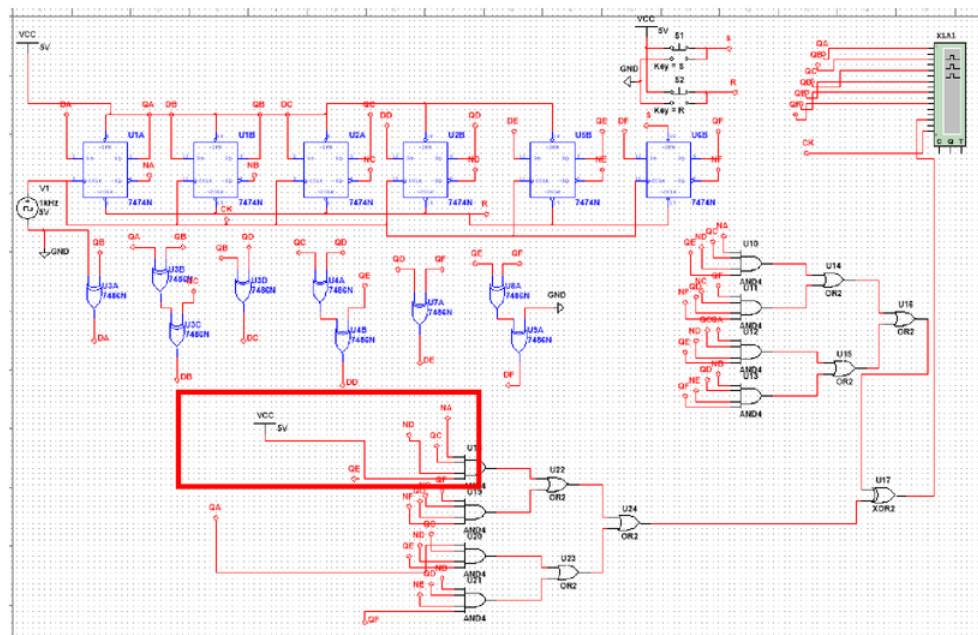


Figure 22- MS fault 3

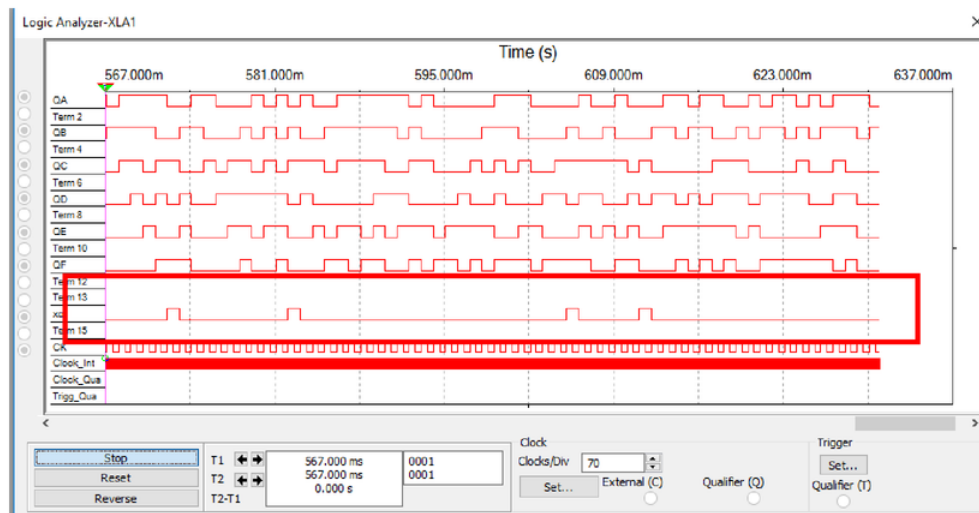


Figure 23- MS fault 3 waveform

The third and final Multisim circuit to be tested had its faults successfully detected, shown again by the XOR output. The table in Figure 24 shows a summary of the test results:

Boolean o/p from 6-bit CAR	Boolean required from good cct	Sim fault result from test cct	Fault detected y/n
NA.NB.NC.D.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.'0'	Y
A.B.C.ND.E.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.E + B.NC.D.NF + '0'.C.ND.E + NB.D.NE.F	Y
NA.NB.C.ND.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.'1' + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	Y

Figure 24-MS faults summary table

Test and analysis is very good and shows the fault condition results clearly.

5. Analysis of results using hardware descriptive language FPGA technology

A 4 bit CAR 90 circuit and a 4 bit CAR 150 circuit was setup separately using VHDL on Vivado to prove that these circuit won't provide a maximum count. The output waveforms for the CAR 90 and 150 are shown in Figure 25 and Figure 26. The expectations were correct as theory suggested, the 2 configurations do not produce a maximum count on their own.

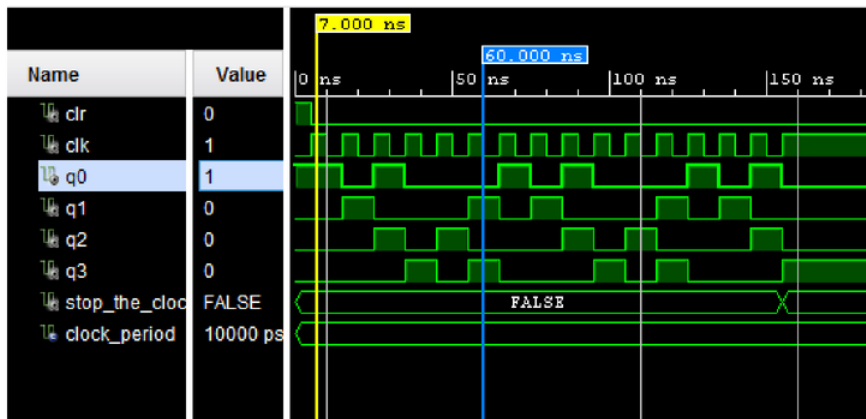


Figure 25- CAR 90

CAR 90 count: 1, 2, 5, 8, 4, 10, 1 – No max count

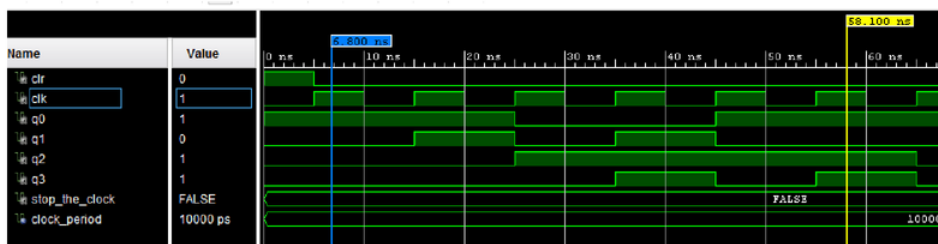


Figure 26- CAR 150

CAR 150 count: 1, 3, 4, 14, 5, 13, 1 – No max count

The two 4bit car circuits were then developed into a 4-bit hybrid to prove that the maximum count was achieved. The key piece of code that uses the rules in sequence is shown below:

```

D(3) <= '0' xor qs(2);
D(2) <= qs(3) xor qs(2) xor qs(1);
D(1) <= qs(2) xor qs(0);
D(0) <= qs(1) xor qs(0) xor '0';

```

When simulated, the VHDL program produced the results in Figure 27.

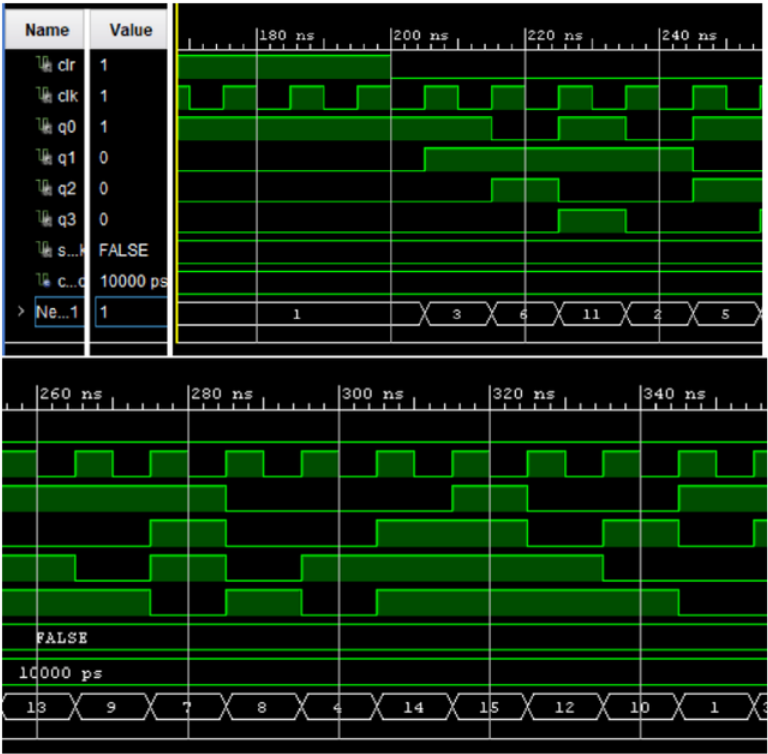


Figure 27- 4-bit hybrid VHDL

CAR90/150 count: 1, 3, 6, 11, 2, 5, 13, 9, 7, 8, 4, 14, 15, 12, 10, 1 – Maximum count achieved

analysis 4 bit hybrid proves the max count theory.

1	00000001		19	00010011
3	00000011		62	00111110
6	00000110		55	00110111
11	00001011		32	00100000
18	00010010		16	00010000
61	00111101		56	00111000
49	00110001		60	00111100
43	00101011		50	00110010
2	00000010		45	00101101
5	00000101		9	00001001
13	00001101		23	00010111
25	00011001		48	00110000
47	00101111		40	00101000
12	00001100		4	00000100
26	00011010		14	00001110
41	00101001		31	00011111
7	00000111		36	00100100
8	00001000		30	00011110
20	00010100		39	00100111
54	00110110		24	00011000
35	00100011		44	00101100
22	00010110		10	00001010
51	00110011		17	00010001
45	00101101		59	00111011
15	00001111		58	00111010
28	00011100		57	00111001
34	00100010		63	00111111
21	00010101		52	00110100
53	00110101		38	00100110
37	00100101		27	00011011
29	00011101		42	00101010
33	00100001		1	00000001

Figure 29- VHDL 6-bit hybrid count table

The 6 bit hybrid circuit worked as expected and was made into an IP in the Vivadoo software. The base components of the circuits were built and saved as IP files in Vivado. This allowed individual IP components to be imported into Vivado and connected to other IP's to create a complete circuit. The benefit of using this method is that it is very clear and logical what's happening in the circuit. As the internal components for each IP are hidden (but can easily be opened), this allows a big picture look at what is happening overall.

One of the most important IP's in the circuit is the 6-bit hybrid linear feedback shift register. This IP contains the crucial algorithms that enable the circuit to produce pseudo random numbers. Highlighted in Figure 30 is Wolframs algorithms: CAR rule 90 and CAR rule 150. There is one instance of a rule per bit, the first rule used is 90 and then alternates 150, 90, 150, 90, 150. This IP along with other IP's were imported and connected into a functional schematic. This schematic can then be used to program the FPGA. See Figure 31 and Figure 32 for the full diagram.

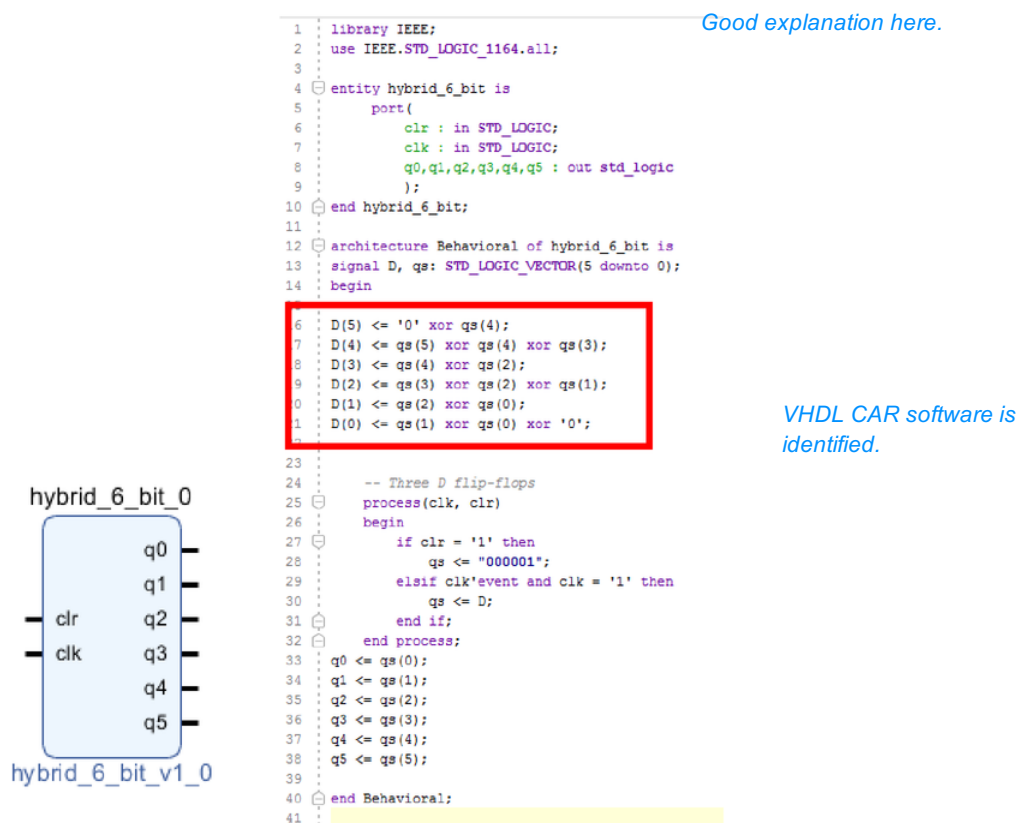


Figure 30- VHDL hybrid 6 bit

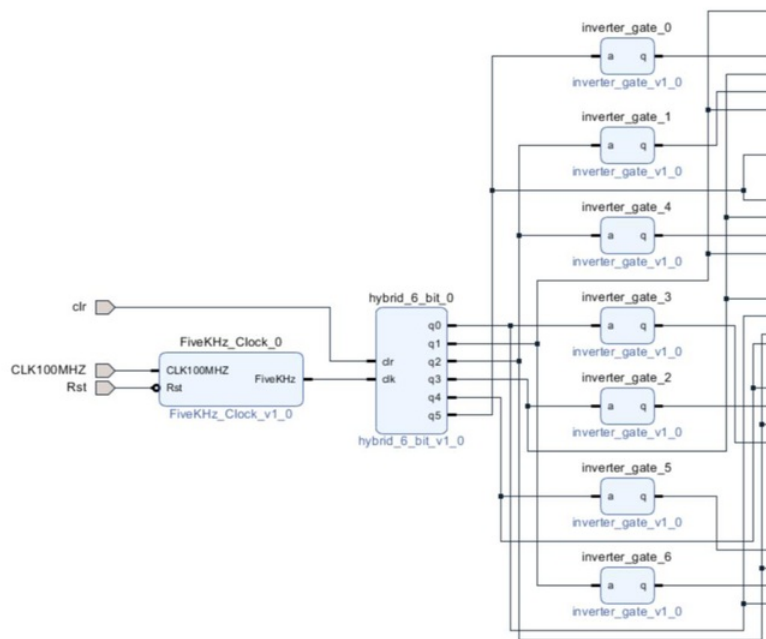


Figure 31- 6-bit hybrid VHDL

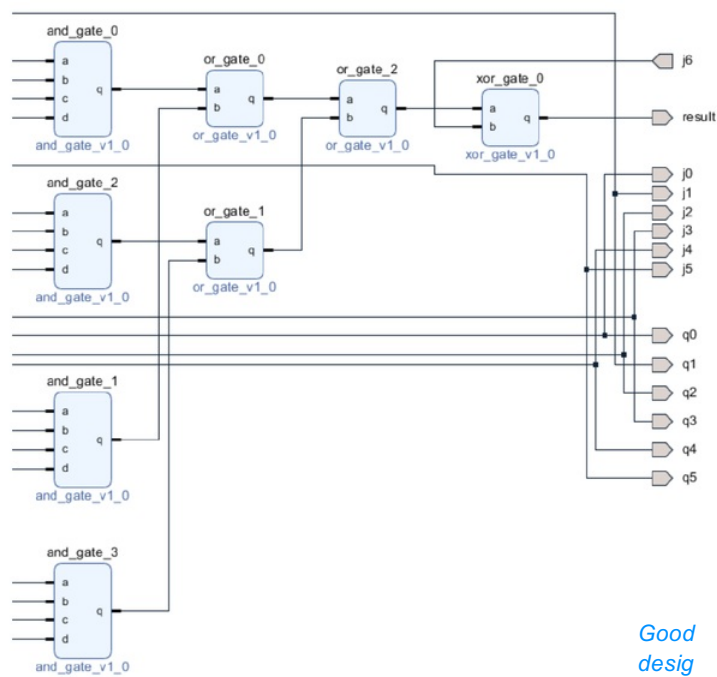
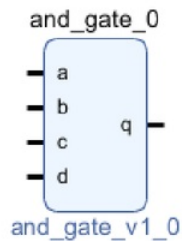


Figure 32- 6-bit hybrid VHDL

Good design which includes the all important

Each individual IP used in the schematic, along with the VHDL code are listed below:

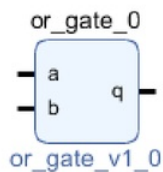


```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6  entity and_gate is
7  -- Port ( );
8  Port ( a : in std_logic;
9         b : in std_logic;
10        c : in std_logic;
11        d : in std_logic;
12        q : out std_logic);
13 end and_gate;
14
15 architecture Behavioral of and_gate is
16
17 begin
18   q <= a and b and c and d;
19
20 end Behavioral;
21

```

Figure 33-VHDL 4 input and gate

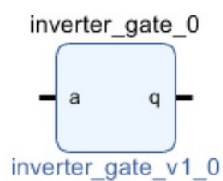


```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6  entity or_gate is
7  -- Port ( );
8  Port ( a : in std_logic;
9         b : in std_logic;
10        q : out std_logic);
11 end or_gate;
12
13 architecture Behavioral of or_gate is
14
15 begin
16   q <= a or b;
17
18 end Behavioral;
19

```

Figure 34-VHDL 2 input or gate

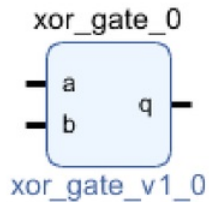


```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6  entity inverter_gate is
7  -- Port ( );
8  Port ( a : in std_logic;
9        q : out std_logic);
10 end inverter_gate;
11
12 architecture Behavioral of inverter_gate is
13
14 begin
15   q <= not a;
16
17 end Behavioral;
18

```

Figure 35- VHDL inverter (NOT gate)

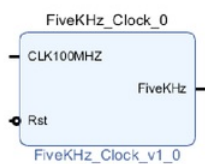


```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6  entity xor_gate is
7  -- Port ( );
8  Port ( a : in std_logic;
9        b : in std_logic;
10       q : out std_logic);
11 end xor_gate;
12
13 architecture Behavioral of xor_gate is
14
15 begin
16   q <= a xor b;
17
18 end Behavioral;
19
20

```

Figure 36- VHDL 2 input XOR gate



```

4  use IEEE.std_logic_unsigned.all;
5
6  entity FiveKHz_Clock is
7
8  Port ( CLK100MHZ : in    STD_LOGIC;
9        Rst       : in    std_logic;
10       FiveKHz    : out   STD_LOGIC
11
12       );
13
14 end FiveKHz_Clock;
15
16 architecture Behavioral of FiveKHz_Clock is
17
18   signal cnt1      : integer := 0;
19   signal temp1     : std_logic := '0';
20
21
22
23 begin
24
25
26   process(CLK100MHZ,Rst)
27   begin
28     if Rst = '1' then
29       temp1 <= '0';
30       cnt1 <= 0;
31     elsif rising_edge (CLK100MHZ) then
32       if(cnt1= 50000000) then
33         temp1 <= not temp1;
34         cnt1 <= 0;
35       else
36         cnt1 <= cnt1 +1 ;
37       end if;
38     end if;
39
40   end process;
41   FiveKHz <= temp1;
42
43 end Behavioral;
44
45

```

Figure 37- VHDL clock divided to 5KHz

The FPGA board was programmed using the full design. The same faults that were tested on Multisim are being tested using the FPGA board. The full 6-bit hybrid VHDL program was downloaded to the FPGA and compared with a logic circuit built on breadboard. As with previous tests; A 0 out for the XOR gate means the circuits match, a 1 output say that the circuit has detected a fault. The following Boolean expression describes a correct circuit:

$$NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F$$

The first fault to be tested is the 'F' pin tied to ground. As the connections to the breadboard are impossible to see from the images, a diagram in Figure 38 shows the pin F set to 0, which would usually be a 1 for the good circuit. The fault was successfully detected as shown by LED 15 lighting up.

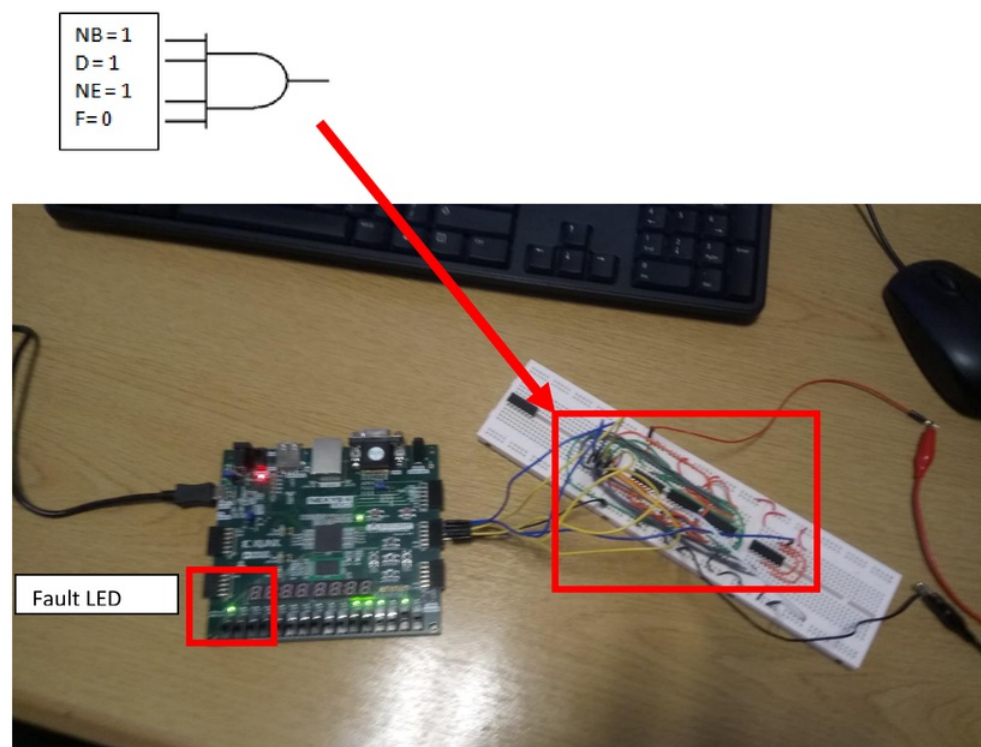


Figure 38-FPGA fault 1

The actual Boolean expression for the incorrect circuit is:

$$NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.'0'$$

The second fault to test is the 'A' pin tied to ground. A diagram of the fault and a picture of the output is shown in Figure 39.

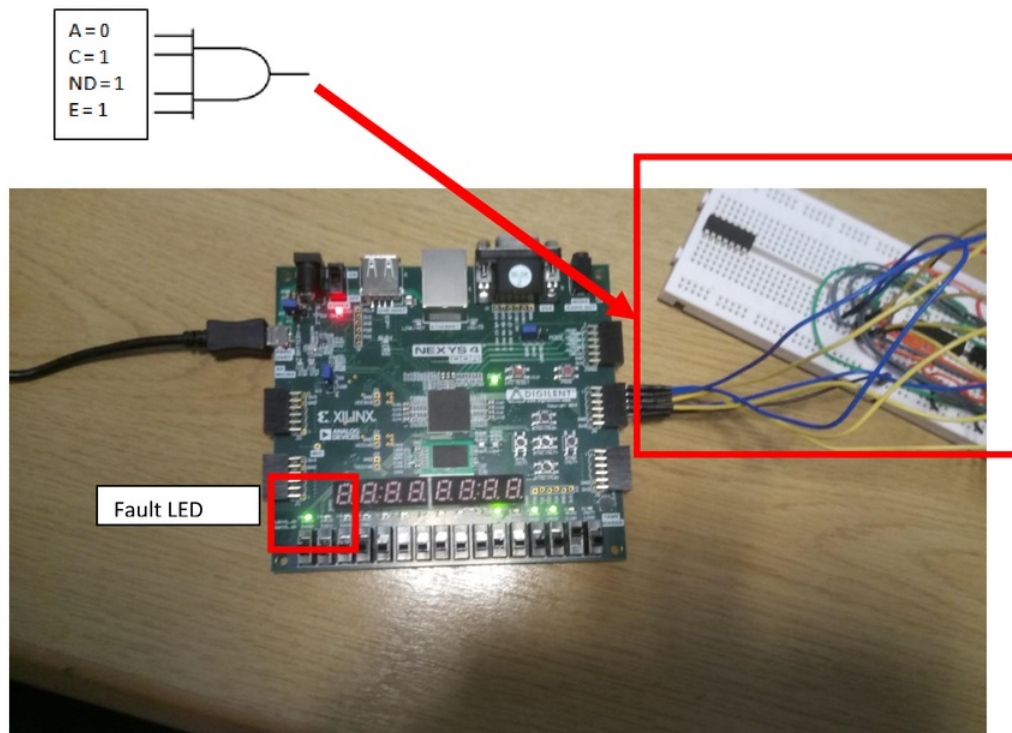


Figure 39- FPGA fault 2

The Boolean expression for this incorrect circuit is:

$$NA.C.ND.E + B.NC.D.NF + '0'.C.ND.E + NB.D.NE.F$$

The third test is tying the 'E' pin to Vcc. The diagram of the fault and results are shown in Figure 40.

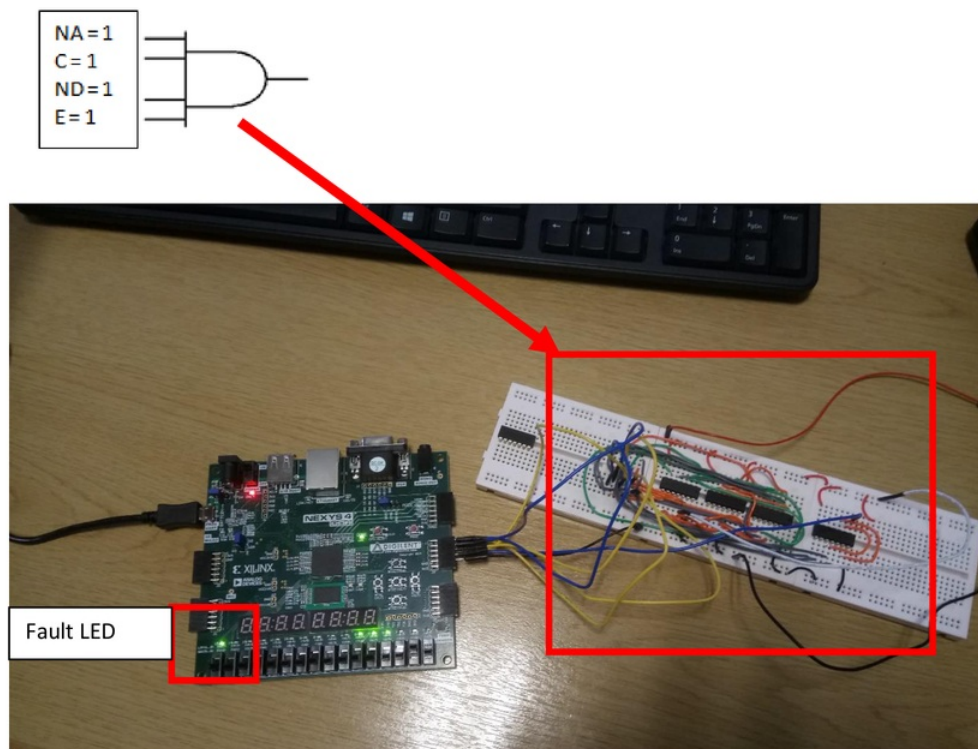


Figure 40- FPGA fault 3

The Boolean expression for the faulty circuit is:

$$NA.C.ND.'1' + B.NC.D.NF + A.C.ND.E + NB.D.NE.F$$

A table of results for the FPGA tests are shown in Figure 41. The FPGA passed all three tests by detecting the faults that occurred. If the table of results are compared with the Multisim results, they match exactly.

Boolean o/p from 6-bit CAR	Boolean required from good cct	Sim fault result from test cct	Fault detected y/n
NA.NB.NC.D.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.'0'	Y
A.B.C.ND.E.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.E + B.NC.D.NF + '0'.C.ND.E + NB.D.NE.F	Y
NA.NB.C.ND.NE.F	NA.C.ND.E + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	NA.C.ND.'1' + B.NC.D.NF + A.C.ND.E + NB.D.NE.F	Y

Figure 41- FPGA test results

Test process is very good and explains the test of the practical CUT before the PCB design and test. The faults are identified and related conditions identified.

6. Analysis of PCB design

The previously designed circuit was built on the software package Eagle. The circuit was built up using logic gates, shown in Figure 42. The software then converts the logic circuit into a routed design using 2 sides of the PCB and the component footprints of choice. It was decided that the ‘chip’ version of the components would be used when constructing the PCB.

Once the circuit has been routed the software generates various files, which are common file types that are produced from any PCB manufacturing software. This is what makes it easy for PCB manufacturers as they don't have to deal with different file types when customers use different software packages. This is like how all 3D CAD packages can export models in the form of G code, which can be understood by any CNC machine and most 3D printers. The files that were generated have to be individually inspected before sending them to the manufacturer, to ensure there aren't any errors. Each file represents a process or layer of the PCB, this will be explained with each image. The full layout, which can be thought of as a compilation of all the layers, is shown in Figure 43.

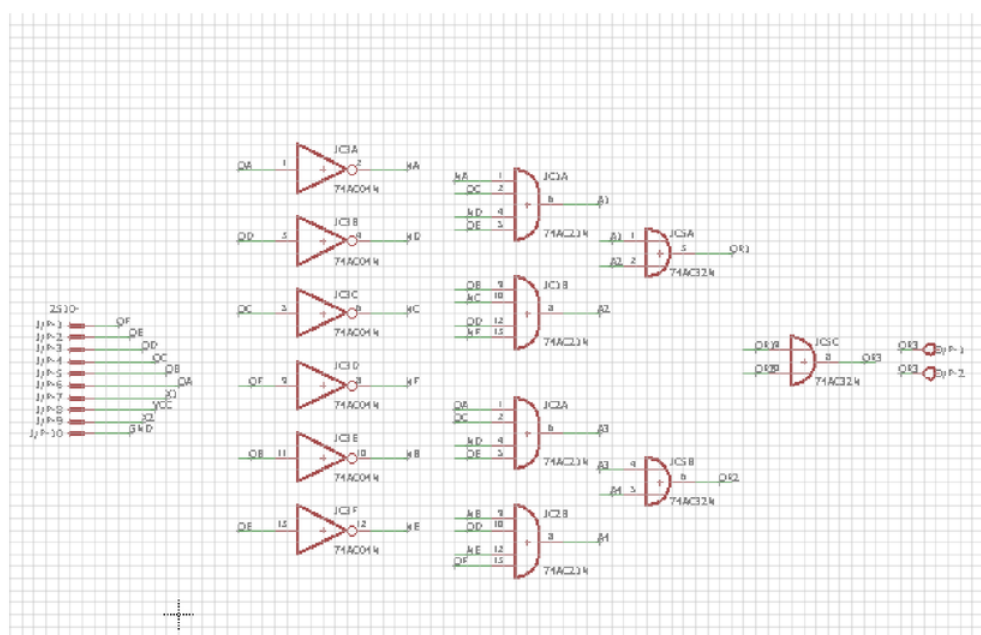


Figure 42- Eagle logic

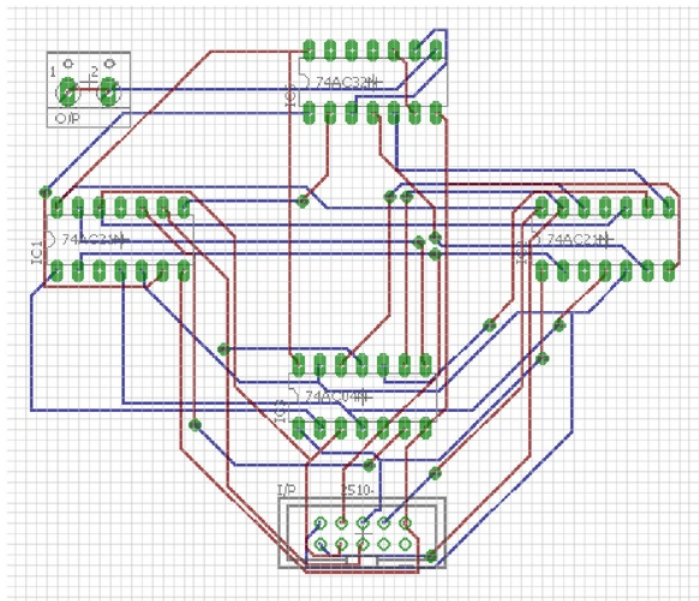


Figure 43- Eagle routing

The first file in the list is the .cmp file, (SIERRA CIRCUITS, INC., 2018) which shows the copper tracks and pads on the component side, shown in Figure 44.

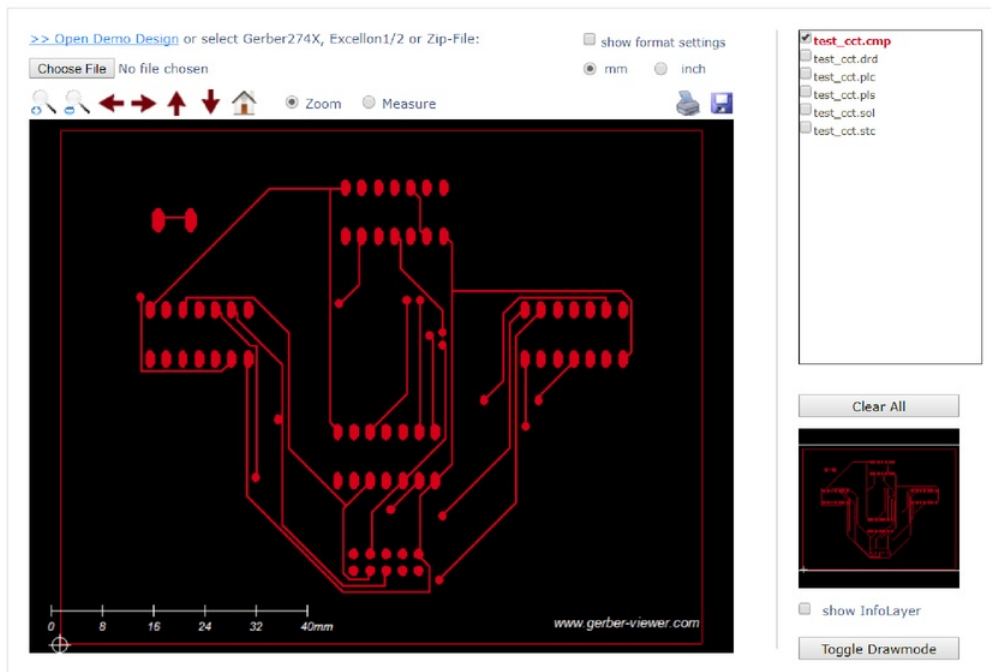


Figure 44 - eagle .cmp

The next file on the list is the .drd file (SIERRA CIRCUITS, INC., 2018). This is the layer that specifies where the holes are to be drilled through the PCB. See Figure 45.

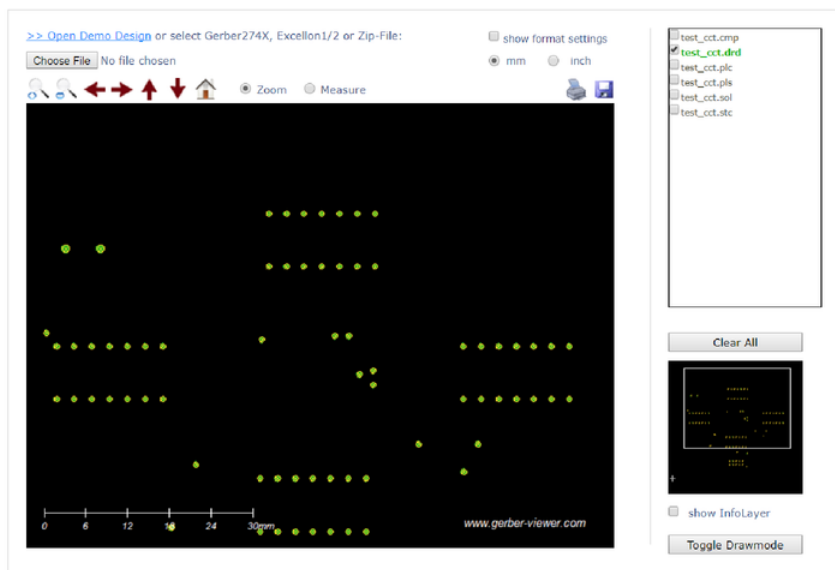


Figure 45 - eagle .drd

The .plc file is the silk-screen mask for the component side of the PCB (SIERRA CIRCUITS, INC., 2018). See Figure 46. This is how the outlines of the IC's are printed onto the PCB.

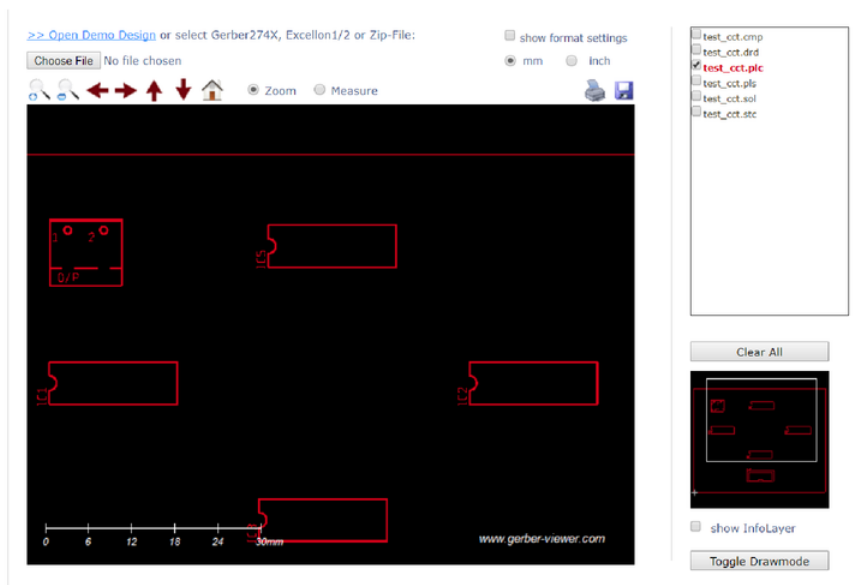


Figure 46- eagle .plc

The .pls file may look identical to the .cmp file, but this may not be the case for every design. Its purpose is as a silk screen mask for the solder side (SIERRA CIRCUITS, INC., 2018). See Figure 47.

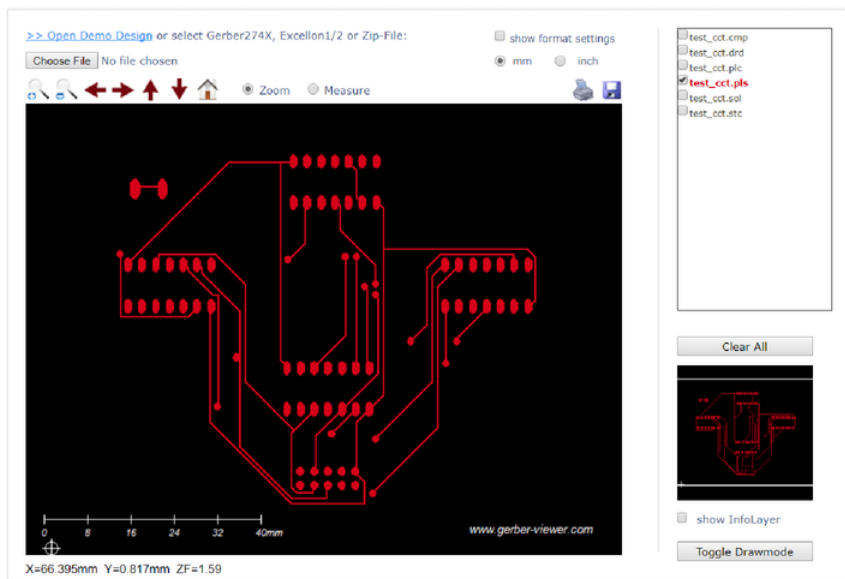


Figure 47- eagle .pls

The .sol file is where the copper needs to go on the solder side of the PCB (SIERRA CIRCUITS, INC., 2018). See Figure 48.

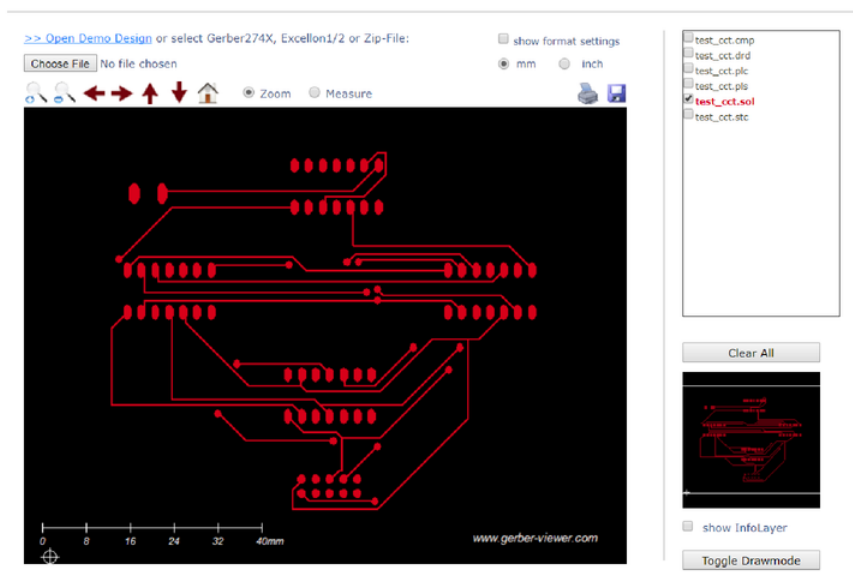


Figure 48- eagle .sol

The last file in the list is the .stc file. This is the solder mask for the component side. This is where the solder will join the components to the copper tracks. See Figure 49.

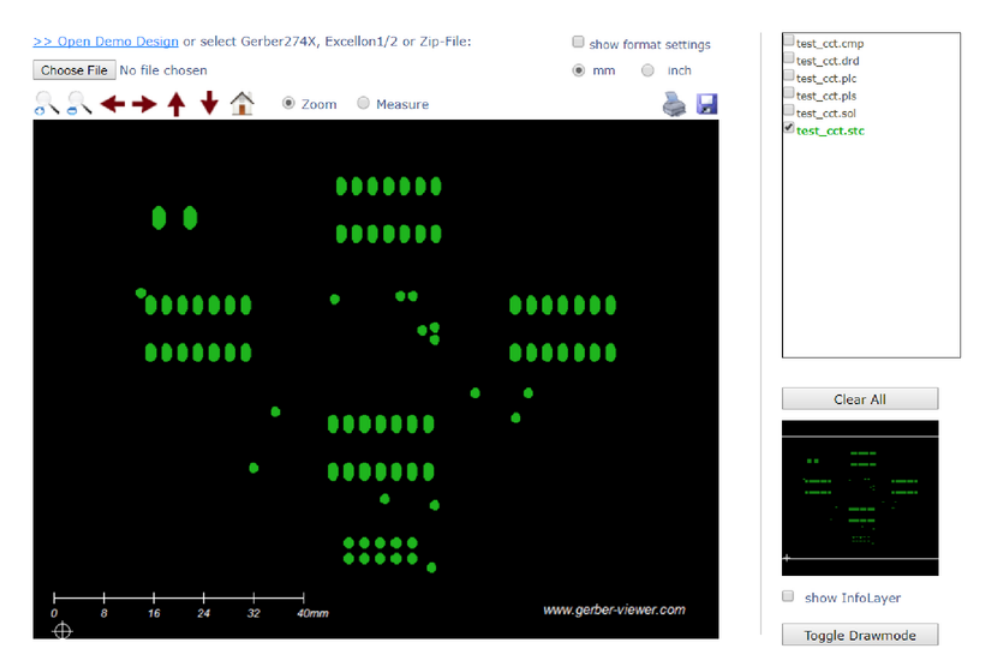


Figure 49- eagle .stc

Once all the files are combined, the PCB shown in Figure 50 is created. The PCB looks identical to the PCB in Figure 43. This is expected as the 6 files were generated from the original routing on eagle, so they are just component parts of an original all-in-one design.

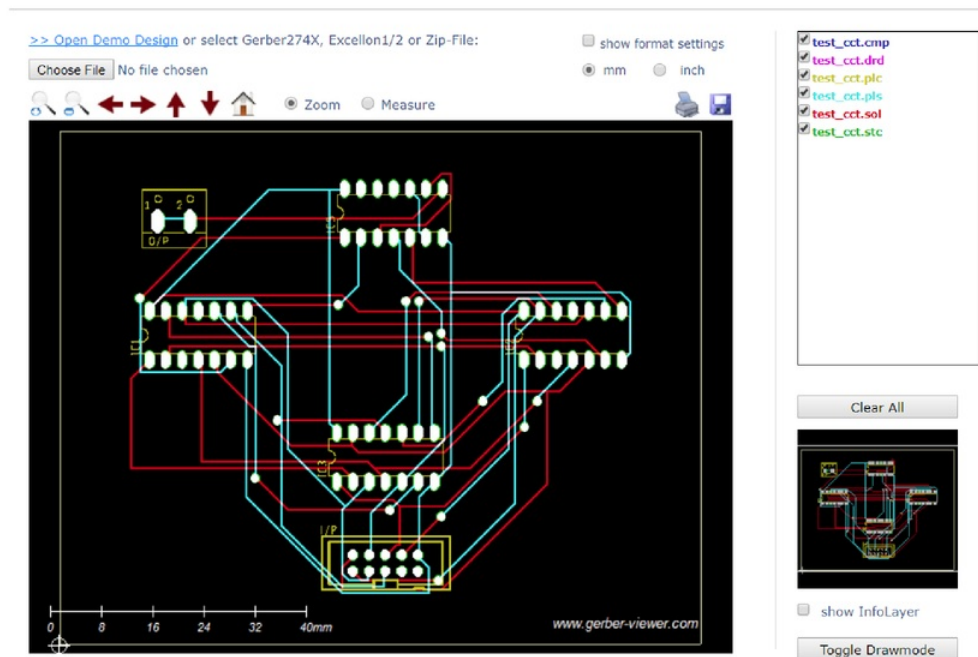


Figure 50- eagle combined files

PCB design is correct and shows the process clearly. The files needed for production could be explained in this section.

7. Practical PCB build and testing of prototype on prototyping board

The designed PCB files previously discussed was sent to a local PCB manufacturer. The boards came back as per the specifications. Both sides of the PCB are shown in Figure 51. The circuit downloaded onto the FPGA was compared with the physical build, shown in Figure 52. The circuit was run and found to be working, no faults were detected and the XOR led remained off.

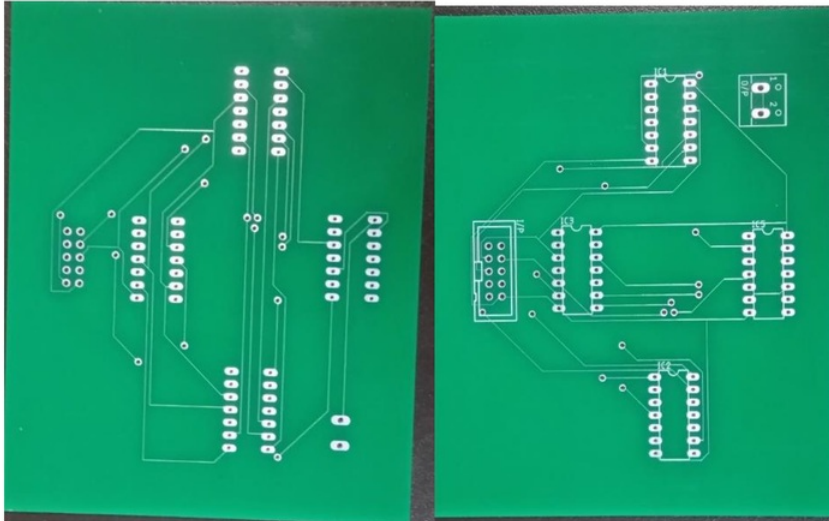


Figure 51- PCB

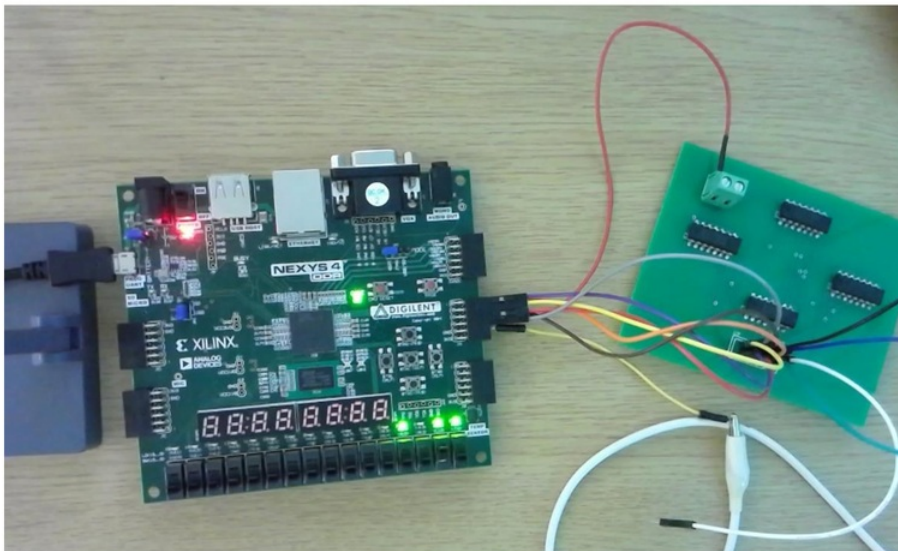


Figure 52- FPGA PCB test

To thoroughly test the produced board, a fault was put onto the PCB and tested to see if it was detected. The XOR light comes on which indicates a fault has been detected. Figure 53 shows when the FPGA detected a difference in the count. After performing these tests and getting reasonable results, the PCB design and build is a success.

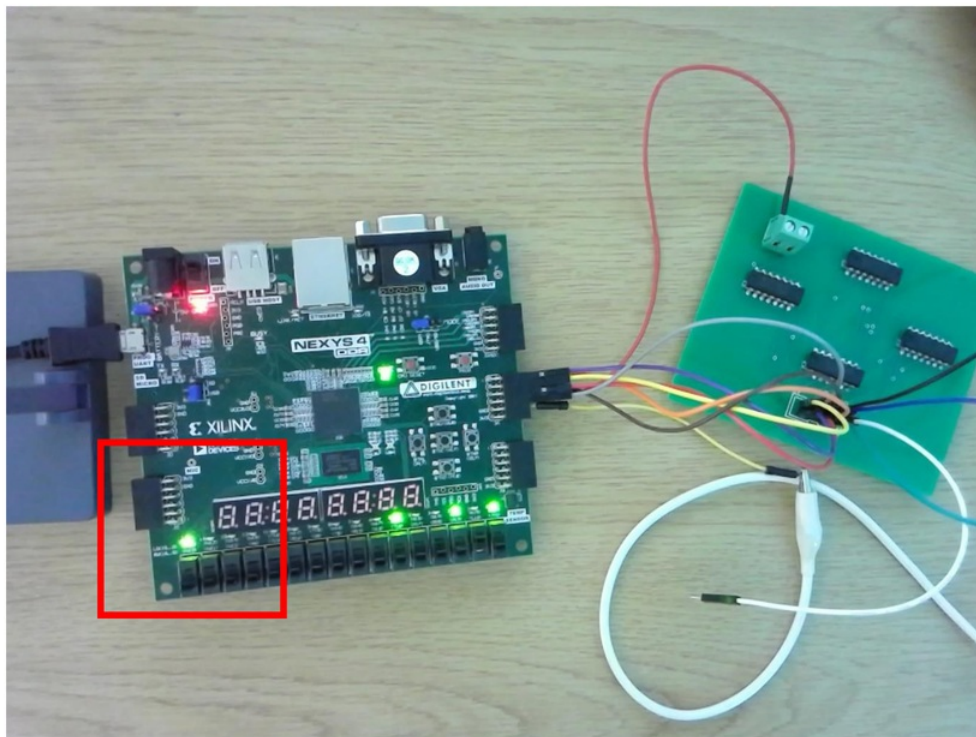


Figure 53- FPGA PCB fault

*PCB test is
well
illustrated.*

The actual components that were used in this build is as follows:

- 74ALS21 – Dual 4 input and gate (Figure 54)
- 74AC11086 – Quad 2 input XOR gate (Figure 55)
- SN74LS32N – Quad 2 input OR gate (Figure 56)
- 7404 – Hex inverter (Figure 57)

74ALS21 Dual 4-Input AND Gate

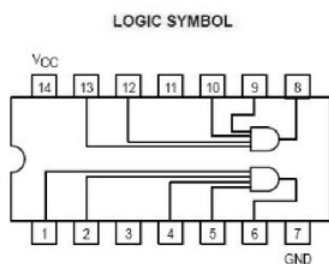


Figure 54 - 74ALS21

74AC11086

QUADRUPLE 2-INPUT EXCLUSIVE-OR GATE

SCAS081A - NOVEMBER 1989 - REVISED APRIL 1996

- Center-Pin V_{CC} and GND Configurations Minimize High-Speed Switching Noise
- EPIC™ (Enhanced-Performance Implanted CMOS) 1- μ m Process
- 500-mA Typical Latch-Up Immunity at 125°C
- Package Options Include Plastic Small-Outline (D) Packages and Standard Plastic 300-mil DIPs (N)

description

This device contains four independent 2-input exclusive-OR gates. It performs the Boolean function $Y = A \oplus B = \overline{A}B + A\overline{B}$ in positive logic.

A common application is as a true/complement element. If one of the inputs is low, the other input is reproduced in true form at the output. If one of the inputs is high, the signal on the other input is reproduced inverted at the output.

The 74AC11086 is characterized for operation from -40°C to 85°C.

D OR N PACKAGE
(TOP VIEW)

1A	1	10	1B
1Y	2	15	2A
2Y	3	14	2B
GND	4	13	V_{CC}
GND	5	12	V_{CC}
3Y	6	11	3A
4Y	7	10	3B
4B	8	9	4A

FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

74AC11086
QUADRUPLE 2-INPUT EXCLUSIVE-OR GATE

SCAS081A – NOVEMBER 1989 – REVISED APRIL 1996

logic diagram (positive logic)

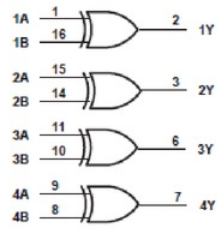


Figure 55- 74AC11086

SN74LS32N Quad 2-Input OR Gate

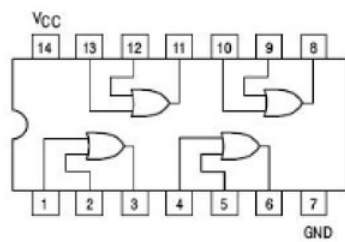


Figure 56- SN74LS32N

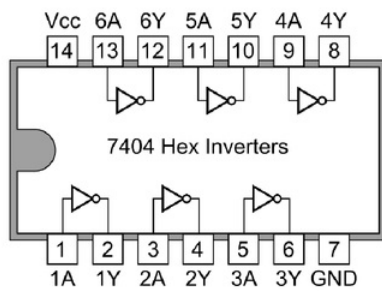


Figure 57- 7404

8. Conclusions and recommendations

The specifications laid out in the introduction have been successfully met. The entire design process was followed from start to finish.

The research shows how real-world manufacturing techniques enable the production of the latest technology. The ever-increasing density of components and complexity of layers within a PCB enables engineer to design ever more elaborate circuitry. Allowing machines to control and perform most of the processes within the manufacturing process allows line speeds to be far greater than with just human labour. The machine being used are also capable of things well beyond the scope of a human, the automatic test systems that can detect a fault that in some cases can't be seen by eye.

The theory of CAR circuits was studied and how they are able to produce pseudo random numbers that were almost impossible to predict (without knowledge of the seed and circuit connections). The fact that simple XOR gates and Flip-flops are able to produce such unique pseudorandom patterns based entirely off mathematical algorithms (rule 90, 150 etc.), really highlights the role of the complex mathematical relationship that connects even the simplest of logic gates, and what can be achieved if the algorithms are utilised in the correct order. Only once rule 90 and rule 150 were combined in the correct configuration was it possible to achieve a maximum count pseudorandom number generator, as proved throughout the assignment.

Different methods of solving the same problem was also carried out in this assignment: The circuit was simulated just using software on Multisim, with faults on circuits it was being compared with also being tested. The circuit was rebuilt using VHDL and programmed onto the FPGA board, which was compared to actual logic circuits on breadboard. This comparison between simulation and real-world hardware can be made from these tests: The two circuits performed the same function, and both were able to correctly detect faults. The accuracy of the simulation software at simulating this type of circuit is that it's accurate enough to achieve the same results as a build with components, with the added benefit of detailed analysis that can be performed in the software.

The use of Eagle to produce a Gerber file and its component files gives a good insight into the process that goes into designing and producing a circuit in the real world, as similar software and techniques are carried out in modern companies today. In fact, the same manufacturers who produce PCB's for businesses was used in this assignment.

*Good
conclusion
which
includes
relevant
arguments and*

9. References

- ALPHA & OMEGA SEMICONDUCTOR, INC. , n.d. *Recommended temperature profile for solder AOS product*. [Online]
Available at: http://www.aosmd.com/res/application_notes/package/Soldering_Profiles.pdf
[Accessed 12 2018].
- Advanced Circuits Inc, 2009. *Building a printed circuit board*. [Online]
Available at: www.4pcb.com/media/building-printed-circuit-board.pdf
- AspenCore, 2001. *Where does AOI fit in PCB manufacturing?*. [Online]
Available at: <https://www.edn.com/design/test-and-measurement/4383990/Where-does-AOI-fit-in-PCB-manufacturing->
- Bergenthal, J., 1997. *Reflow Soldering Process Considerations for Surface Mount Application*. [Online]
Available at: <http://www.kemet.com/Lists/TechnicalArticles/Attachments/28/f2102a.pdf>
- Clarydon Electronic Services Limited , 2018. *Multi-layer Printed Circuit Boards*. [Online]
Available at: <https://www.clarydon.com/manufacturing/multi-layer-pcb-fabrication>
- Kasdon Electronics, 2018. *Through-hole Printed Circuit Board Assembly*. [Online]
Available at: <https://www.kasdonpcb.com/through-hole-assembly>
- Orbotech, 2018. *Innovative, advanced technologies offer simple, intelligent and powerful performance*. [Online]
Available at: <https://www.orbotech.com/products/categories/pcb/aoi>
- process, s. m., 2015. *Introduction to Surface Mount Technology*. [Online]
Available at: www.surfacemountprocess.com/reflow-soldering-process.html
- Retronix Ltd., 2018. *PCB services - X-ray inspection*. [Online]
Available at: www.retronix.com/services/xray-inspection-service/
- Riemersma, T., 2015. *Reflow soldering profile*. [Online]
Available at: <https://www.compuphase.com/electronics/reflowsolderprofiles.htm#>
- SIERRA CIRCUITS, INC., 2018. *Gerber File Format*. [Online]
Available at: https://www.protoexpress.com/content/gerber_extensions.jsp
[Accessed 3 2 2018].

Surface Mount Technologies., 2018. *SMT solutions*. [Online]
Available at: <http://teamsmt.com/solutions/>
[Accessed 22 2 2018].

Viscom AG, 2018. *X7056-II – 3D AXI/3D AOI*. [Online]
Available at: <https://www.viscom.com/europe/products/solder-joint-inspection-aximxi/x7056-ii-3d-axi3d-aoi/>

Number of refs is appropriate.

10. Appendices

6bit hybrid wrapper.vhd

```
1  --Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
2
3  --Tool Version: Vivado v.2017.2 (win64) Build 1909853 Thu Jun 15 16:39:09 MDT 20
4  --Date       : Wed Jan 10 19:27:31 2018
5  --Host       : GR-G09-02 running 64-bit major release  (build 9200)
6  --Command    : generate_target hybrid_t4_wrapper.bd
7  --Design     : hybrid_t4_wrapper
8  --Purpose    : IP block netlist
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.VCOMPONENTS.ALL;
14 entity hybrid_t4_wrapper is
15     port (
16         CLK100MHZ : in STD_LOGIC;
17         Rst       : in STD_LOGIC;
18         clr       : in STD_LOGIC;
19         j0        : out STD_LOGIC;
20         j1        : out STD_LOGIC;
21         j2        : out STD_LOGIC;
22         j3        : out STD_LOGIC;
23         j4        : out STD_LOGIC;
24         j5        : out STD_LOGIC;
25         j6        : in STD_LOGIC;
26         q0        : out STD_LOGIC;
27         q1        : out STD_LOGIC;
28         q2        : out STD_LOGIC;
29         q3        : out STD_LOGIC;
30         q4        : out STD_LOGIC;
31         q5        : out STD_LOGIC;
32         result    : out STD_LOGIC
33     );
34 end hybrid_t4_wrapper;
35
36 architecture STRUCTURE of hybrid_t4_wrapper is
37     component hybrid_t4 is
38     port (
39         q3 : out STD_LOGIC;
40         j3 : out STD_LOGIC;
41         q2 : out STD_LOGIC;
42         j2 : out STD_LOGIC;
43         q1 : out STD_LOGIC;
44         j1 : out STD_LOGIC;
45         q4 : out STD_LOGIC;
46         j4 : out STD_LOGIC;
47         q0 : out STD_LOGIC;
48         j0 : out STD_LOGIC;
49         q5 : out STD_LOGIC;
50         j5 : out STD_LOGIC;
51         clr : in STD_LOGIC;
52         CLK100MHZ : in STD_LOGIC;
53         Rst : in STD_LOGIC;
54         result : out STD_LOGIC;
55         j6 : in STD_LOGIC
56     );
```

```

53     Rst : in STD_LOGIC;
54     result : out STD_LOGIC;
55     j6 : in STD_LOGIC
56 );
57 end component hybrid_t4;
58 begin
59 hybrid_t4_i: component hybrid_t4
60     port map (
61         CLK100MHZ => CLK100MHZ,
62         Rst => Rst,
63         clr => clr,
64         j0 => j0,
65         j1 => j1,
66         j2 => j2,
67         j3 => j3,
68         j4 => j4,
69         j5 => j5,
70         j6 => j6,
71         q0 => q0,
72         q1 => q1,
73         q2 => q2,
74         q3 => q3,
75         q4 => q4,
76         q5 => q5,
77         result => result
78     );
79 end STRUCTURE;
80

```

Hybrid.vhd

```
1  --Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
2
3  --Tool Version: Vivado v.2017.2 (win64) Build 1909853 Thu Jun 15 18:39:09 MDT 2017
4  --Date       : Wed Jan 10 19:27:31 2018
5  --Host       : GR-G09-02 running 64-bit major release  (build 9200)
6  --Command    : generate_target hybrid_t4.bd
7  --Design     : hybrid_t4
8  --Purpose    : IP block netlist
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.VCOMPONENTS.ALL;
14 entity hybrid_t4 is
15 port (
16     CLK100MHZ : in STD_LOGIC;
17     Rst : in STD_LOGIC;
18     clr : in STD_LOGIC;
19     j0 : out STD_LOGIC;
20     j1 : out STD_LOGIC;
21     j2 : out STD_LOGIC;
22     j3 : out STD_LOGIC;
23     j4 : out STD_LOGIC;
24     j5 : out STD_LOGIC;
25     j6 : in STD_LOGIC;
26     q0 : out STD_LOGIC;
27     q1 : out STD_LOGIC;
28     q2 : out STD_LOGIC;
29     q3 : out STD_LOGIC;
30     q4 : out STD_LOGIC;
31     q5 : out STD_LOGIC;
32     result : out STD_LOGIC
33 );
34 attribute CORE_GENERATION_INFO : string;
35 attribute CORE_GENERATION_INFO of hybrid_t4 : entity is "hybrid_t4,IP_Integrator,
36 attribute HW_HANDOFF : string;
37 attribute HW_HANDOFF of hybrid_t4 : entity is "hybrid_t4.hwdef";
38 end hybrid_t4;
39
40 architecture STRUCTURE of hybrid_t4 is
41 component hybrid_t4_FiveKHz_Clock_0_0 is
42 port (
43     CLK100MHZ : in STD_LOGIC;
44     Rst : in STD_LOGIC;
45     FiveKHz : out STD_LOGIC
46 );
47 end component hybrid_t4_FiveKHz_Clock_0_0;
48 component hybrid_t4_hybrid_6_bit_0_0 is
49 port (
50     clr : in STD_LOGIC;
51     clk : in STD_LOGIC;
52     q0 : out STD_LOGIC;
53     q1 : out STD_LOGIC;
54     q2 : out STD_LOGIC;
55     q3 : out STD_LOGIC;
56     q4 : out STD_LOGIC;
```

```

54     q2 : out STD_LOGIC;
55     q3 : out STD_LOGIC;
56     q4 : out STD_LOGIC;
57     q5 : out STD_LOGIC
58 );
59 end component hybrid_t4_hybrid_6_bit_0_0;
60 component hybrid_t4_inverter_gate_0_0 is
61 port (
62     a : in STD_LOGIC;
63     q : out STD_LOGIC
64 );
65 end component hybrid_t4_inverter_gate_0_0;
66 component hybrid_t4_inverter_gate_1_0 is
67 port (
68     a : in STD_LOGIC;
69     q : out STD_LOGIC
70 );
71 end component hybrid_t4_inverter_gate_1_0;
72 component hybrid_t4_inverter_gate_2_0 is
73 port (
74     a : in STD_LOGIC;
75     q : out STD_LOGIC
76 );
77 end component hybrid_t4_inverter_gate_2_0;
78 component hybrid_t4_inverter_gate_3_0 is
79 port (
80     a : in STD_LOGIC;
81     q : out STD_LOGIC
82 );
83 end component hybrid_t4_inverter_gate_3_0;
84 component hybrid_t4_inverter_gate_4_0 is
85 port (
86     a : in STD_LOGIC;
87     q : out STD_LOGIC
88 );
89 end component hybrid_t4_inverter_gate_4_0;
90 component hybrid_t4_inverter_gate_5_0 is
91 port (
92     a : in STD_LOGIC;
93     q : out STD_LOGIC
94 );
95 end component hybrid_t4_inverter_gate_5_0;
96 component hybrid_t4_inverter_gate_6_0 is
97 port (
98     a : in STD_LOGIC;
99     q : out STD_LOGIC
100 );
101 end component hybrid_t4_inverter_gate_6_0;
102 component hybrid_t4_xor_gate_0_0 is
103 port (
104     a : in STD_LOGIC;
105     b : in STD_LOGIC;
106     q : out STD_LOGIC
107 );
108 end component hybrid_t4_xor_gate_0_0;
109 component hybrid_t4_and_gate_0_0 is

```

```

108     end component hybrid_t4_xor_gate_0_0;
109     component hybrid_t4_and_gate_0_0 is
110     port (
111         a : in STD_LOGIC;
112         b : in STD_LOGIC;
113         c : in STD_LOGIC;
114         d : in STD_LOGIC;
115         q : out STD_LOGIC
116     );
117     end component hybrid_t4_and_gate_0_0;
118     component hybrid_t4_and_gate_1_0 is
119     port (
120         a : in STD_LOGIC;
121         b : in STD_LOGIC;
122         c : in STD_LOGIC;
123         d : in STD_LOGIC;
124         q : out STD_LOGIC
125     );
126     end component hybrid_t4_and_gate_1_0;
127     component hybrid_t4_and_gate_2_0 is
128     port (
129         a : in STD_LOGIC;
130         b : in STD_LOGIC;
131         c : in STD_LOGIC;
132         d : in STD_LOGIC;
133         q : out STD_LOGIC
134     );
135     end component hybrid_t4_and_gate_2_0;
136     component hybrid_t4_and_gate_3_0 is
137     port (
138         a : in STD_LOGIC;
139         b : in STD_LOGIC;
140         c : in STD_LOGIC;
141         d : in STD_LOGIC;
142         q : out STD_LOGIC
143     );
144     end component hybrid_t4_and_gate_3_0;
145     component hybrid_t4_or_gate_0_0 is
146     port (
147         a : in STD_LOGIC;
148         b : in STD_LOGIC;
149         q : out STD_LOGIC
150     );
151     end component hybrid_t4_or_gate_0_0;
152     component hybrid_t4_or_gate_1_0 is
153     port (
154         a : in STD_LOGIC;
155         b : in STD_LOGIC;
156         q : out STD_LOGIC
157     );
158     end component hybrid_t4_or_gate_1_0;
159     component hybrid_t4_or_gate_2_0 is
160     port (
161         a : in STD_LOGIC;
162         b : in STD_LOGIC;
163         q : out STD_LOGIC

```

```

162     b : in STD_LOGIC;
163     q : out STD_LOGIC
164 );
165 end component hybrid_t4_or_gate_2_0;
166 signal CLK100MHZ_1 : STD_LOGIC;
167 signal FiveKHz_Clock_0_FiveKHz : STD_LOGIC;
168 signal Rst_1 : STD_LOGIC;
169 signal and_gate_0_q : STD_LOGIC;
170 signal and_gate_1_q : STD_LOGIC;
171 signal and_gate_2_q : STD_LOGIC;
172 signal and_gate_3_q : STD_LOGIC;
173 signal clr_1 : STD_LOGIC;
174 signal hybrid_6_bit_0_q0 : STD_LOGIC;
175 signal hybrid_6_bit_0_q1 : STD_LOGIC;
176 signal hybrid_6_bit_0_q2 : STD_LOGIC;
177 signal hybrid_6_bit_0_q3 : STD_LOGIC;
178 signal hybrid_6_bit_0_q4 : STD_LOGIC;
179 signal hybrid_6_bit_0_q5 : STD_LOGIC;
180 signal inverter_gate_0_q : STD_LOGIC;
181 signal inverter_gate_1_q : STD_LOGIC;
182 signal inverter_gate_2_q : STD_LOGIC;
183 signal inverter_gate_3_q : STD_LOGIC;
184 signal inverter_gate_4_q : STD_LOGIC;
185 signal inverter_gate_5_q : STD_LOGIC;
186 signal inverter_gate_6_q : STD_LOGIC;
187 signal j6_1 : STD_LOGIC;
188 signal or_gate_0_q : STD_LOGIC;
189 signal or_gate_1_q : STD_LOGIC;
190 signal or_gate_2_q : STD_LOGIC;
191 signal xor_gate_0_q : STD_LOGIC;
192 begin
193     CLK100MHZ_1 <= CLK100MHZ;
194     Rst_1 <= Rst;
195     clr_1 <= clr;
196     j0 <= hybrid_6_bit_0_q0;
197     j1 <= hybrid_6_bit_0_q1;
198     j2 <= hybrid_6_bit_0_q2;
199     j3 <= hybrid_6_bit_0_q3;
200     j4 <= hybrid_6_bit_0_q4;
201     j5 <= hybrid_6_bit_0_q5;
202     j6_1 <= j6;
203     q0 <= hybrid_6_bit_0_q0;
204     q1 <= hybrid_6_bit_0_q1;
205     q2 <= hybrid_6_bit_0_q2;
206     q3 <= hybrid_6_bit_0_q3;
207     q4 <= hybrid_6_bit_0_q4;
208     q5 <= hybrid_6_bit_0_q5;
209     result <= xor_gate_0_q;
210     FiveKHz_Clock_0: component hybrid_t4_FiveKHz_Clock_0_0
211     port map (
212         CLK100MHZ => CLK100MHZ_1,
213         FiveKHz => FiveKHz_Clock_0_FiveKHz,
214         Rst => Rst_1
215     );
216     and_gate_0: component hybrid_t4_and_gate_0_0
217     port map (|

```

```

216 and_gate_0: component hybrid_t4_and_gate_0_0
217     port map (
218         a => inverter_gate_0_q,
219         b => hybrid_6_bit_0_q3,
220         c => inverter_gate_1_q,
221         d => hybrid_6_bit_0_q1,
222         q => and_gate_0_q
223     );
224 and_gate_1: component hybrid_t4_and_gate_1_0
225     port map (
226         a => hybrid_6_bit_0_q4,
227         b => inverter_gate_2_q,
228         c => hybrid_6_bit_0_q2,
229         d => inverter_gate_3_q,
230         q => and_gate_1_q
231     );
232 and_gate_2: component hybrid_t4_and_gate_2_0
233     port map (
234         a => hybrid_6_bit_0_q5,
235         b => hybrid_6_bit_0_q3,
236         c => inverter_gate_4_q,
237         d => hybrid_6_bit_0_q1,
238         q => and_gate_2_q
239     );
240 and_gate_3: component hybrid_t4_and_gate_3_0
241     port map (
242         a => inverter_gate_5_q,
243         b => hybrid_6_bit_0_q2,
244         c => inverter_gate_6_q,
245         d => hybrid_6_bit_0_q0,
246         q => and_gate_3_q
247     );
248 hybrid_6_bit_0: component hybrid_t4_hybrid_6_bit_0_0
249     port map (
250         clk => FiveKHz_Clock_0_FiveKHz,
251         clr => clr_1,
252         q0 => hybrid_6_bit_0_q0,
253         q1 => hybrid_6_bit_0_q1,
254         q2 => hybrid_6_bit_0_q2,
255         q3 => hybrid_6_bit_0_q3,
256         q4 => hybrid_6_bit_0_q4,
257         q5 => hybrid_6_bit_0_q5
258     );
259 inverter_gate_0: component hybrid_t4_inverter_gate_0_0
260     port map (
261         a => hybrid_6_bit_0_q5,
262         q => inverter_gate_0_q
263     );
264 inverter_gate_1: component hybrid_t4_inverter_gate_1_0
265     port map (
266         a => hybrid_6_bit_0_q2,
267         q => inverter_gate_1_q
268     );
269 inverter_gate_2: component hybrid_t4_inverter_gate_2_0
270     port map (
271         a => hybrid_6_bit_0_q3,

```

```

271     a => hybrid_6_bit_0_q3,
272     q => inverter_gate_2_q
273 );
274 inverter_gate_3: component hybrid_t4_inverter_gate_3_0
275     port map (
276         a => hybrid_6_bit_0_q0,
277         q => inverter_gate_3_q
278     );
279 inverter_gate_4: component hybrid_t4_inverter_gate_4_0
280     port map (
281         a => hybrid_6_bit_0_q2,
282         q => inverter_gate_4_q
283     );
284 inverter_gate_5: component hybrid_t4_inverter_gate_5_0
285     port map (
286         a => hybrid_6_bit_0_q4,
287         q => inverter_gate_5_q
288     );
289 inverter_gate_6: component hybrid_t4_inverter_gate_6_0
290     port map (
291         a => hybrid_6_bit_0_q1,
292         q => inverter_gate_6_q
293     );
294 or_gate_0: component hybrid_t4_or_gate_0_0
295     port map (
296         a => and_gate_0_q,
297         b => and_gate_1_q,
298         q => or_gate_0_q
299     );
300 or_gate_1: component hybrid_t4_or_gate_1_0
301     port map (
302         a => and_gate_2_q,
303         b => and_gate_3_q,
304         q => or_gate_1_q
305     );
306 or_gate_2: component hybrid_t4_or_gate_2_0
307     port map (
308         a => or_gate_0_q,
309         b => or_gate_1_q,
310         q => or_gate_2_q
311     );
312 xor_gate_0: component hybrid_t4_xor_gate_0_0
313     port map (
314         a => or_gate_2_q,
315         b => j6_1,
316         q => xor_gate_0_q
317     );
318 end STRUCTURE;
319

```

Ass1 final

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

75/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

Text Comment. Good introduction which explains the task ahead.

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

Text Comment. Research covers a wide area and includes relevant points. There is good attention to detail demonstrated.

PAGE 13

PAGE 14

PAGE 15

PAGE 16

Text Comment. The CAR prototypes are tied in with theory very well.

Text Comment. Good analysis of 4 bit CAR to prove the maximum count.

Text Comment. Good detail and explanation about this stage.

Text Comment. Test and analysis is very good and shows the fault condition results clearly.

Text Comment. analysis 4 bit hybrid proves the max count theory.

Text Comment. 6 bit test shows the max count and repeat.

Text Comment. Good explanation here.

Text Comment. VHDL CAR software is identified.

Text Comment. Good design which includes the all important x-or gate for comparison.

PAGE 34

PAGE 35

PAGE 36

Text Comment. Test process is very good and explains the test of the practical CUT before the PCB design and test. The faults are identified and related conditions identified.

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

Text Comment. PCB design is correct and shows the process clearly. The files needed for production could be explained in this section.

PAGE 43

PAGE 44

Text Comment. PCB test is well illustrated.

PAGE 45

PAGE 46

PAGE 47

Text Comment. Good conclusion which includes relevant arguments and outcomes.

PAGE 48

PAGE 49

Text Comment. Number of refs is appropriate.

PAGE 50

PAGE 51

PAGE 52

PAGE 53

GRADING FORM: CSG-RUBIC

MICHAEL LEWIS**0****RESEARCH-INFORMED LITERATURE**

Good intro. Virtual ATE theme could have been pointed out. SMT is covered in some detail. 3/5 The component technologies have been researched with good examples. 4/5 Auto inspection is explained. 3.5/5 X-ray is explained. 3.5/5

14/20**KNOWLEDGE AND UNDERSTANDING**

Design procedure sequence is explained well and is easy to follow. The development of the CAR system follows the logical steps/prototypes involved which enables the IP function blocks to be created. 5.5/7 The layers of the PCB design process is explained and tested. This also shows the test using the on line software. 5.5/7 The CAR design using FPGA IP blocks is logical and follows the sequence of each block being designed and tested in turn. 5/6

16/20**ANALYSIS**



Test results are good. Some minor points missing . The link with the test cct and LFSR CAR needed explanation. 7/10 Analysis of FPGA and PCB design is explained well and includes the necessary diagrams. 8/10

15/20

PRACTICAL APPLICATION AND DEP.



the PCB process from design to manufacture is complete and the necessary files which enable the manufacture are tested. This is illustrated with explanation. The solder process is with the use of a purpose built oven where a heat profile is setup beforehand. 12/15 FPGA testing and prototyping is explained and illustrated. The PCB was placed under test using the FPGA CAR design. The tests proved successful. 8/10

20/25

SKILLS FOR PROFESSIONAL PRAC.



The use of different software to complete the tasks was very professional. 3.5/5 Report layout is well structured and logical. 3.5/5 Conclusion could have discussed virtual ATE integration theme. 3/5

10/15