

1708159

by Jonathon Williams

A DETAILED REPORT COVERING ALL ASPECTS OF THE BRIEF. THERE WAS QUITE A LOT OF MATERIAL THAT COULD HAVE GONE INTO THE APPENDIX TO AID READABILITY.

COMPREHENSIVE FEEDBACK FROM LECTURER

MARK AGREED



FILE	11737_JONATHON_WILLIAMS_1708159_178073_1218152649.PDF (2.27M)		
TIME SUBMITTED	17-MAY-2021 09:40PM (UTC+0100)	WORD COUNT	4541
SUBMISSION ID	153632639	CHARACTER COUNT	30341

Course: BSc (Hons) Electrical and Electronic Engineering

Module: SGEE6002 Electronic Production and Test

Assignment: Pseudo Random Pattern Generators

Name: Jonathon Williams

University number: 1708159

Date submitted: 17/05/2021

Word count: 3076

CONTENTS

Contents.....	1
1 Executive Summary.....	4
2 Research.....	5
2.1 Pattern Matching.....	5
2.2 Errors in Development.....	7
2.3 Colour Recognition.....	7
2.4 Image Processing Problems and Challenges.....	9
3 Design of Assignment Requirement and constraints.....	11
3.1 CAR 90.....	11
3.2 CAR 150.....	13
3.3 Hybrid 4-bit CAR 90/150.....	14
3.4 6 Bit.....	15
4 Analysis of NI Multisim results.....	18
4.1 Fault 1 – QA Stuck at 0.....	18
4.2 Fault 2 – QA SA1.....	19
4.3 Fault 3 – QC SA0.....	21
5 VHDL Programme Analysis.....	23
5.1 4-bit CAR 90.....	23
5.2 4-Bit CAR 150.....	24
5.3 4-Bit CAR 90/150 Hybrid.....	25
5.4 Five KHz Clock Vivado.....	27
5.5 6-Bit Hybrid.....	28
6 Implementation and Testing.....	31
6.1 PCB Testing.....	32
6.1.1 Fault 1 – Q1 SA0.....	35
6.1.2 Fault 2 – QA SA1.....	36
6.1.3 Fault 3 – QC SA0.....	37
7 Conclusion.....	38
8 Bibliography.....	39

*Good
level of
detail
in the
table of
content
s*

Table of Figures

Figure 2-1: Pattern matching example	5
Figure 2-2 System architecture	6
Figure 2-3 RGB generation within cross-sectional plane	7
Figure 2-4 Example LabVIEW of colour recognition	8
Figure 2-5 Red colour detection LabView GUI	8
Figure 2-6 count vs intensity histogram.	9
Figure 2-7 Contrast stretching	10
Figure 3-1 3-Bit CAR	11
Figure 3-2 XOR 90 ruling	12
Figure 3-3 CAR 90 Multisim.....	12
Figure 3-4 Car 150 rule.....	13
Figure 3-5 CAR 150 Multisim.....	13
Figure 3-6 4-Bit hybrid CAR 90/150 layout	14
Figure 3-7 CAR 90/150 Multisim Circuit.....	15
Figure 3-8 CAR 90/150 Multisim output	15
Figure 3-9 6-bit hybrid CAR	16
Figure 3-10 6-bit Hybrid CAR logic output	16
Figure 3-11 CUT – no-fault condition.....	17
Figure 3-12 CUT no-fault condition waveform	17
Figure 4-1 CUT QA SA0.....	18
Figure 4-2 CUT QA SA0 logic analyser	19
Figure 4-3 CUT QA SA1.....	19
Figure 4-4 CUT QA SA1 logic analyser	20
Figure 4-5 CUT QC SA0.....	21
Figure 4-6 CUT QC SA0 logic analyser	21
Figure 5-1 Declaring entity for CAR 90 ruling	23
Figure 5-2 VHDL code for CAR90	23
Figure 5-3 CAR 90 Testbench waveform.....	24
Figure 5-4 CAR 150 VHDL Code.....	25
Figure 5-5 CAR 150 Testbench waveform.....	25
Figure 5-6 CAR 90/150 hybrid VHDL code	26
Figure 5-7 CAR 90/150 Testbench waveform	26
Figure 5-8 5KHz VHDL Code and testbench	27
Figure 5-9 5KHz clock waveform - $1/0.2\text{ms} = 5\text{KHz}$	27
Figure 5-10 6 Bit entity.....	28
Figure 5-11 Architecture of 6-Bit CAR.....	28
Figure 5-12 6-Bit outputs	29
Figure 5-13 6-Bit testbench code.....	29
Figure 5-14 6-Bit Hybrid CAR waveform	30
Figure 6-1 Constraints file I/O	31
Figure 6-2 6-Bit custom wrapper	32
Figure 6-3 6-Bit count	32
Figure 6-4 IPs added into one project.....	33
Figure 6-5 Final Circuit	33
Figure 6-6 Constraints file code	35
Figure 6-7 Equipment set up.....	35

Figure 6-8 Q1 SA0 – Fault.....35

Figure 6-9 Brown in – Fault cleared 36

Figure 6-10 QA SA1 - Fault 36

Figure 6-11 QA SA1 - Fault rectified.....37

Figure 6-12 QC SA0 - fault 37

Figure 6-13 QC SA0 – fault rectified.....37

Table of Tables

Table 3-1 S3-Bit CAR output.....11

Table 3-2 CAR 90 output 12

Table 3-3 CAR 150 output 13

Table 3-4 Car 90/150 Output 14

Table 4-1 Fault comparison.....22

Exte
nsiv
e
tabl
e of
figur
es

Good
use of
tables
for
compari
son and
analysis

1 EXECUTIVE SUMMARY

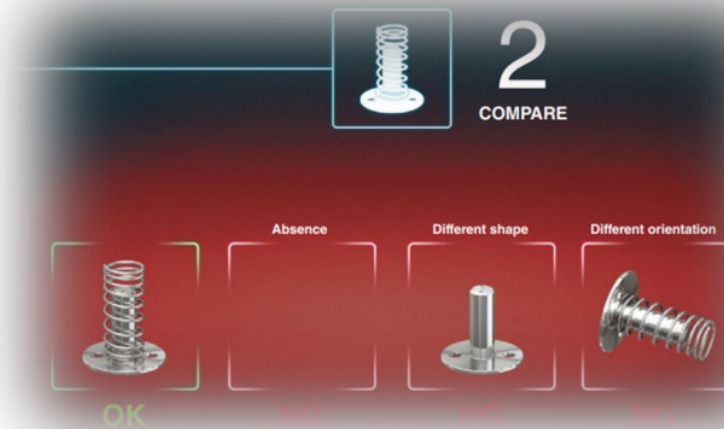
Research was conducted into different techniques in image processing, such as colour analysis, pattern matching. Pattern matching using an algorithm to compare patterns detected to those stored in a database, compared to the x method, comparing the test circuit to the known good circuit. Various image processing techniques were investigated in this project to see if they could be used in PCB manufacturing. Multisim was used to design and build the cellular automata registers, which were then produced into the Vivado. These circuits were put into practice, and the results were compared to the 90 and 150 rulings, as well as the hybrid circuits. The hybrid circuits were discovered to achieve maximum length, which corresponds to the expected theoretical result. The programmes were loaded onto a Nexys DDR board for testing, along with a PCB containing the 6-bit hybrid CAR circuit. Multiple faults were added, and the difference XOR gate successfully detected them.

Good summary that includes all the significant tasks

2 RESEARCH

2.1 PATTERN MATCHING

Pattern matching, a technique that uses an algorithm to equate images to a database, is one example of automatic machine vision. This can help with manufacturing processes by identifying product defects from captured images. Automate production lines with minimal human interaction through the use of high-resolution camera equipment. An example is shown in Figure 2-1.



Good supporting image

Figure 2-1: Pattern matching example

(Keyence, 2017)

This should have a heading

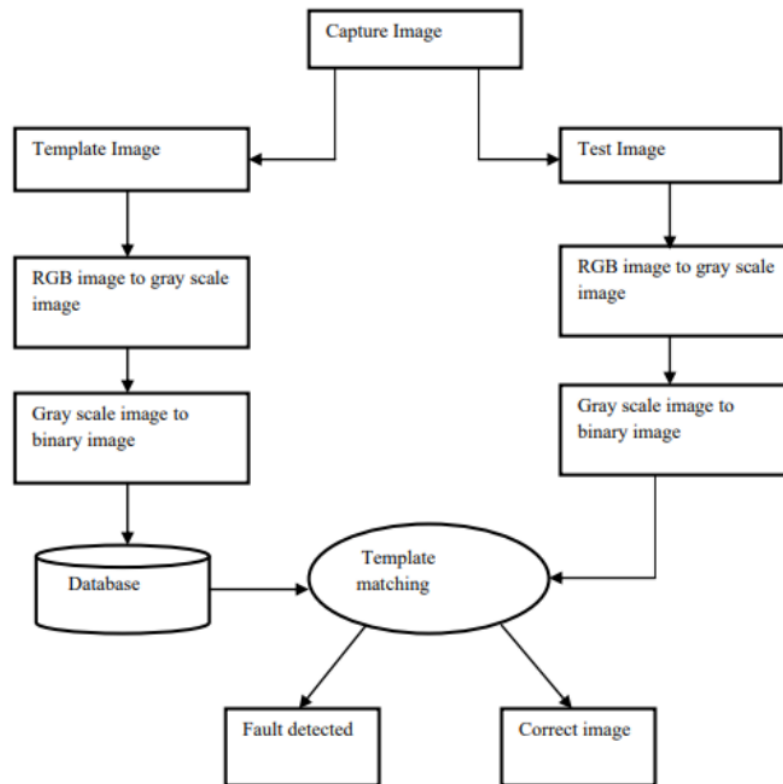


Figure 2-2 System architecture

(Chavan, et al., 2016)

Figure 2-2 depicts the digital image processing system architecture from the initial capture to the binary conversions and template matching. Various algorithms are used, and filter testing methods are used to improve them. During the automated PCB creation process, image processing is used to detect defects on the board. Finally, the final defect is identified by comparing the processed image to the prototype using an XOR gate.

A frequency-domain filter may be used to smooth the image or improve the edges by using Fourier transforms to attenuate the image in either high or low frequencies. According to R. Fisher (2003), processing images in the spatial domain is more computationally effective, but using the frequency domain can have advantages in a variety of scenarios for the desired results. This will result in fewer errors and improved accuracy when it comes to PCB fault detection.

The first search is for missing parts, which alerts the user to significant variations between the captured product and the prototype. Without a computer, faults like this may be observed, but the checks become more complicated and will verify orientation, polarity, and connections.

2.2 ERRORS IN DEVELOPMENT

Surface mount system components are located using pick-place vacuum robots that locate and place the correct components when a PCB is produced. Due to the design of vacuum equipment, these components may often be placed incorrectly. Crispin and Rankov state that "Vacuum blow-off can cause problems such as components not being positioned (too low blow-off) and components being blown out of position (too high blow-off), whereas poor X-Y mechanical alignment can cause position offset problems." (2006) This means that vacuum nozzle control can be inconsistent at times, and mechanical play and wear can affect component positioning. Components can be missing or improperly placed and oriented as a result of such issues. To detect these issues before the boards are released, this process must be supplemented with template matching image processing.

Good example of error detection in a real world application. Pattern matching is explained clearly.

2.3 COLOUR RECOGNITION

Colour strength can be detected, but it cannot be isolated and differentiated from the detected colour. To allow for colour and intensity distinction, a different value known as Hue Saturation Value must be used. According to Nayyer and Sharma (2015), there are RGB imagery requirements that are used by industry. RGB representation has the property of being insensitive to surface changes and unaffected by changes in ambient light.

Good explanation



Figure 2-3 RGB generation within cross-sectional plane

(Nayyer & Sharma, 2015)

To detect each colour, the RGB intensity must be ignored in favour of the hue saturation value. Image processing tools such as those on LabView can be utilised to carry out this task. Nevada states, "LabVIEW gives opportunity to develop code which could differentiate between colours. IMAQ Vision for LabVIEW is a library of LabVIEW VIs that can be used to develop machine vision and scientific imaging applications." (Nevada, 2014). This enables the use of RGB processing through colour matching to a predetermined array, resulting in a magnitude spectrum across the colour spectrum. Figure 2-4 shows an example of a LabVIEW colour

recognition. This one is set up to detect the colour red, Figure 2-3 shows the camera GUI and when red is detected turns on the indicator as shown.

Good explanation

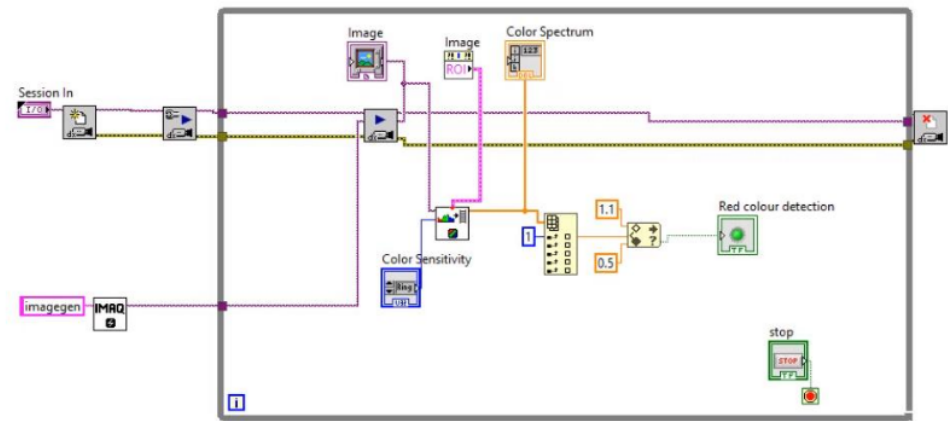


Figure 2-4 Example LabVIEW of colour recognition

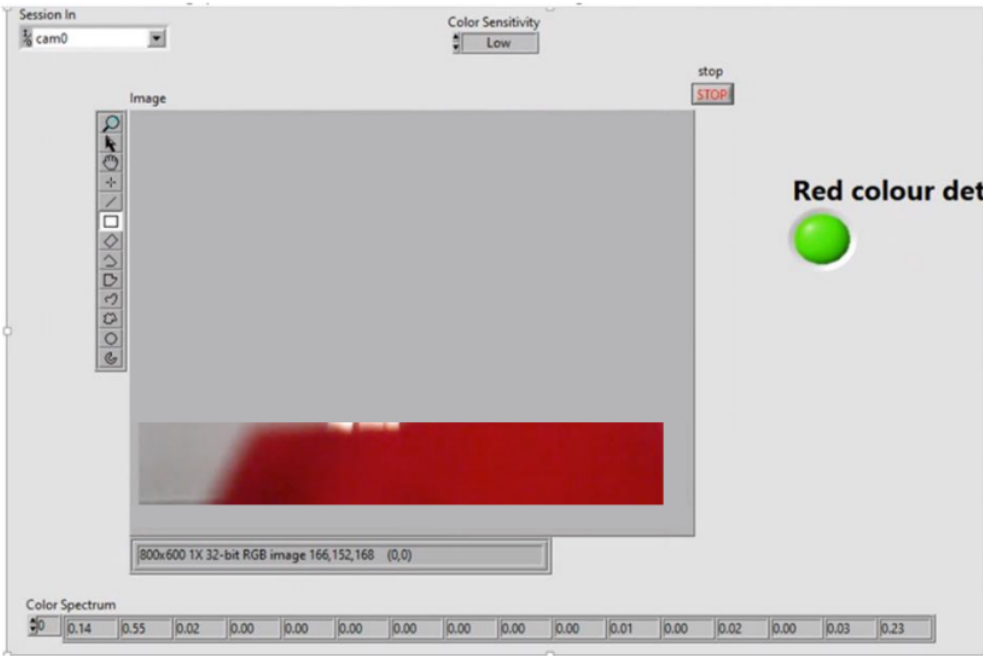


Figure 2-5 Red colour detection LabView GUI

Good
exempl

The array here
can be applica
expanded for
better this
resolution
which would
give better
colour
saturation
sensitivity. This

2.4 IMAGE PROCESSING PROBLEMS AND CHALLENGES

"There are many computational challenges in image processing," according to the Isaac Newton Institute. These include managing picture uncertainties that cannot be removed any other way, as well as various types of incomplete, noisy, imprecise, fragmentary, not completely accurate, ambiguous, contradictory, deficient, and overloading information." (2017). As a result, image processing can be computationally taxing at times in order to resolve these obstacles. It's important to find ways to improve picture quality and make them easier to understand. Images can be of lower quality due to low contrast, blurriness, or noise, according to Huang and Aizawa (1993). The processing must involve adequate algorithms, but these also have drawbacks that distract from the quality in other ways and only address minor issues.

Intensity histograms can be used to enhance issues like contrast and brightness, as shown in Figure 2-6.

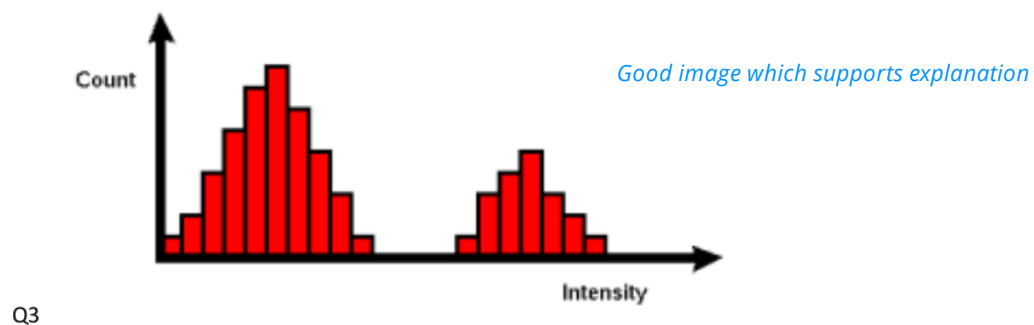


Figure 2-6 count vs intensity histogram.

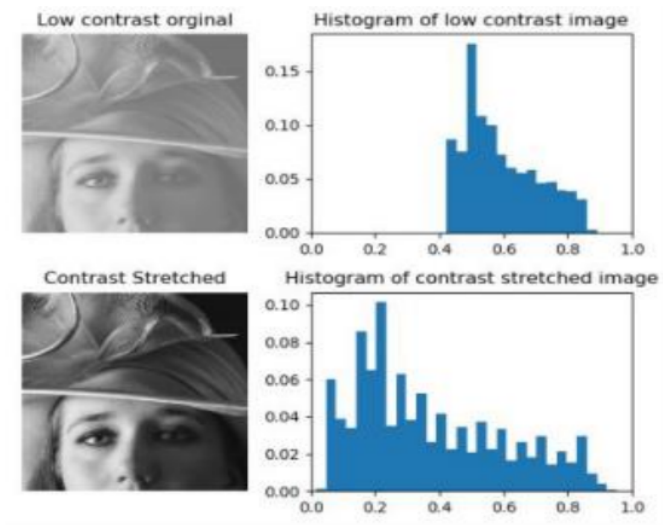
(Fisher, et al., 2003)

The histogram is created by plotting the pixel count against the intensity of an image. This can be achieved with both greyscale and individual colour pictures. These are created by scanning through an image to create a histogram, and they can be processed using techniques like contrast stretching and equalisation.

Non-linear transfer functions are used to transform the pixel intensity values to become constant across the spectrum in order to equalise a histogram; this is effective at maximising image details and minimising extreme contrasts.

Contrast stretching (normalisation) is a technique for broadening the force distribution over a range. Since this approach uses a linear scaling mechanism, the transition is less drastic than equalisation. Figure 2-7 demonstrates how contrast stretching and spreading the strength

across the entire range of values between 0 and 1 improves the picture. This improves visibility and distinguishes details and edges.



A good example

Figure 2-7 Contrast stretching

(Boomgaar, 2017)

3 DESIGN OF ASSIGNMENT REQUIREMENT AND CONSTRAINTS

A Cellular Automatic Register (CAR) generates seemingly random numbers by cycling through the register. D-type flip flops are combined to determine the sequence, with the next state predetermined by the current state and previous states of the adjacent flip flops.

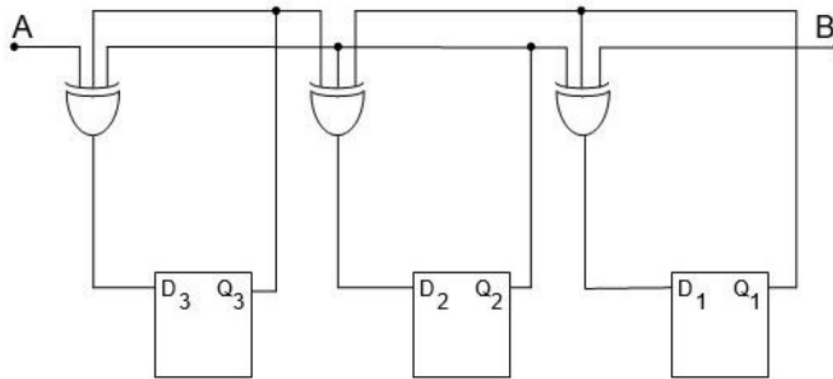


Figure 3-1 3-Bit CAR

An example of a CAR is shown in the figure above; the Inputs (A and B) can be set as either cyclic or null boundary. This is achieved by grounding them or connecting them together. The output pattern table shown in Table 3-1 below represents the states equations for null boundary conditions.

State Equations					
Q3+	=	0	$\oplus$	Q3 $\oplus$ Q2	= Q2 $\oplus$ Q3
Q2+	=	Q3	$\oplus$	Q2 $\oplus$ Q1	= Q1 $\oplus$ Q2 $\oplus$ Q3
Q1+	=	Q2	$\oplus$	Q1 $\oplus$ 0	= Q1 $\oplus$ Q2

Q3	Q2	Q1	Value
0	0	1	1
0	1	1	3
1	0	0	4
1	1	0	6
0	0	1	1

Table 3-1 3-Bit CAR output

A 6-bit hybrid CAR will be developed that consists of the CAR 90 and CAR 150 rules.

3.1 CAR 90

Looking at the tables, this is not a maximum length register; to achieve this, the connection of the flip flops will need to be in a different order. For 3-bit sequences, Wolfram has classified the cell connections using EX-OR gates, as shown in Figure 3-2 below.

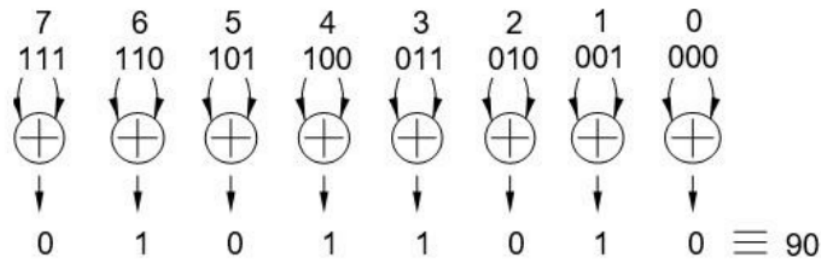


Figure 3-2 XOR 90 ruling

Figure 3-2 depicts the CAR 90 rule, the resultant rule is $Q_t + = Q_{t-1} \oplus Q_{t+1}$. The output table for a 4-bit CAR 90 is shown below in

Table 3-2.

Q4	Q3	Q2	Q1	Value
0	0	0	1	1
0	0	1	0	2
0	1	0	1	5
1	0	0	0	8
0	1	0	0	4
1	0	1	0	10
0	0	0	1	1

CAR 90 characteristics are identified

Table 3-2 CAR 90 output

A CAR 90 circuit was then built on Multisim and the result captured on a logic analyser. This is shown in Figure 3-3 below.

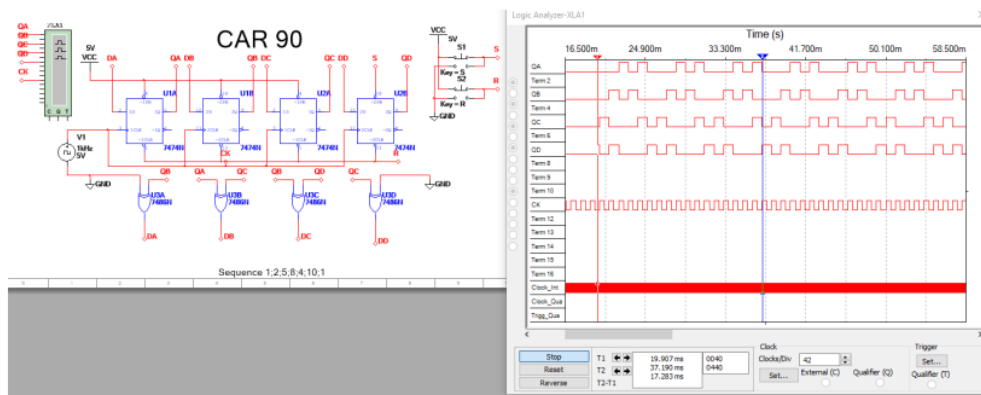


Figure 3-3 CAR 90 Multisim

A good design and test of the CAR 90 system which shows the count is not maximum

As shown above, the correct sequence has been performed and can be seen between the two markers in the logic analyser. However, as this is only using the CAR 90 rule, the maximum length has not been achieved.

3.2 CAR 150

Figure 3-4 shows the CAR 150 rule with the equation: $Q_t += Q_{t-1} \oplus Q_t \oplus Q_{t+1}$

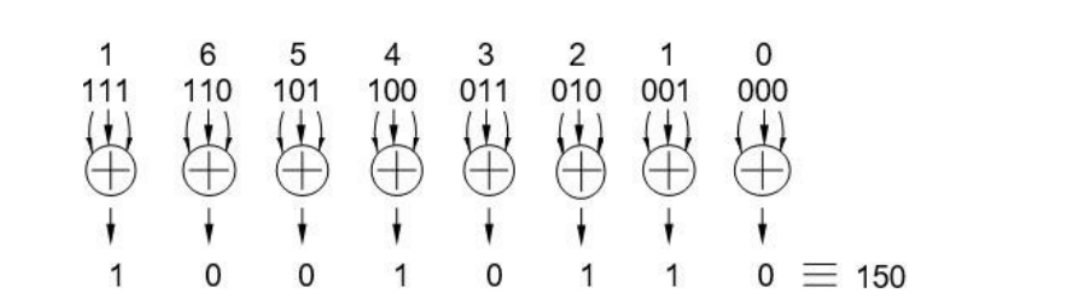


Figure 3-4 Car 150 rule

The CAR 150 rule operates with the outputs of the adjacent flip flops along with the current flip flop state combined into an EXOR gate. The table for the rule 150 CAR output is shown below.

Q4	Q3	Q2	Q1	Value
0	0	0	1	1
0	0	1	1	3
0	1	0	0	4
1	1	1	0	14
0	1	0	1	5
1	1	0	1	13
0	0	0	1	1

CAR 150 characteristics are identified

Table 3-3 CAR 150 output

This was built on Multisim, as shown in Figure 3-5.

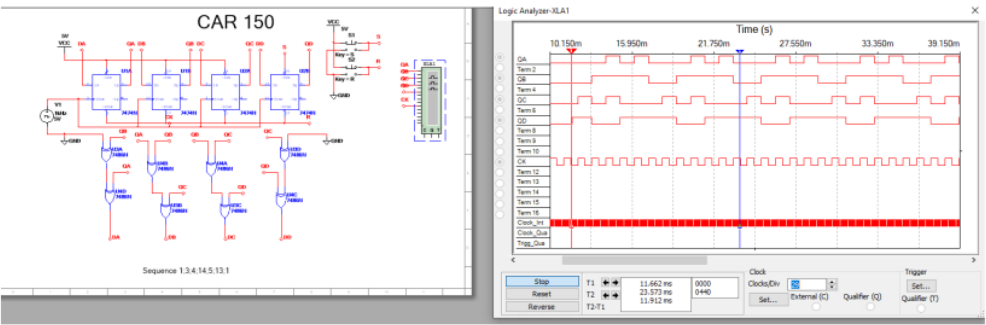
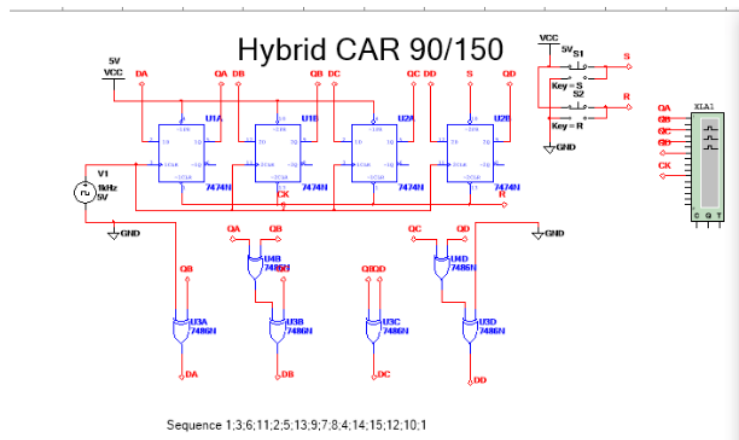


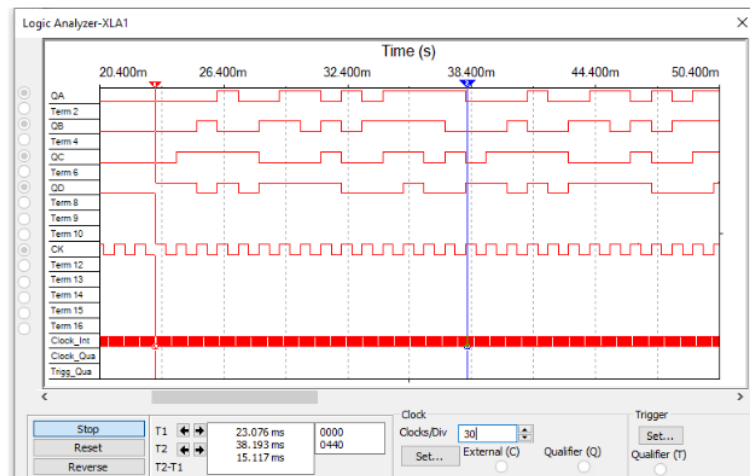
Figure 3-5 CAR 150 Multisim

Design and test demonstrate that the



Design is correct

Figure 3-7 CAR 90/150 Multisim Circuit



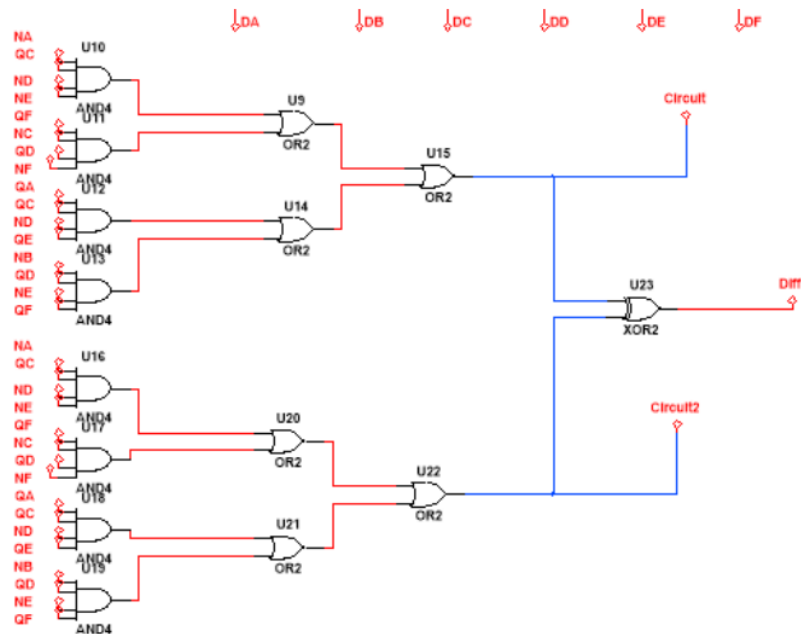
The waveform shows the full and random count

Figure 3-8 CAR 90/150 Multisim output

As shown in the figures above, the count is random, but it is maximum.

3.4 6 Bit

Using Multisim, a 6-bit hybrid CAR was developed, as shown in the below figures. A 6-bit hybrid Car was developed using Multisim and sequence viewed using the logic analyser.



The test cct has no fault implemented and the test shows this

Figure 3-11 CUT – no-fault condition

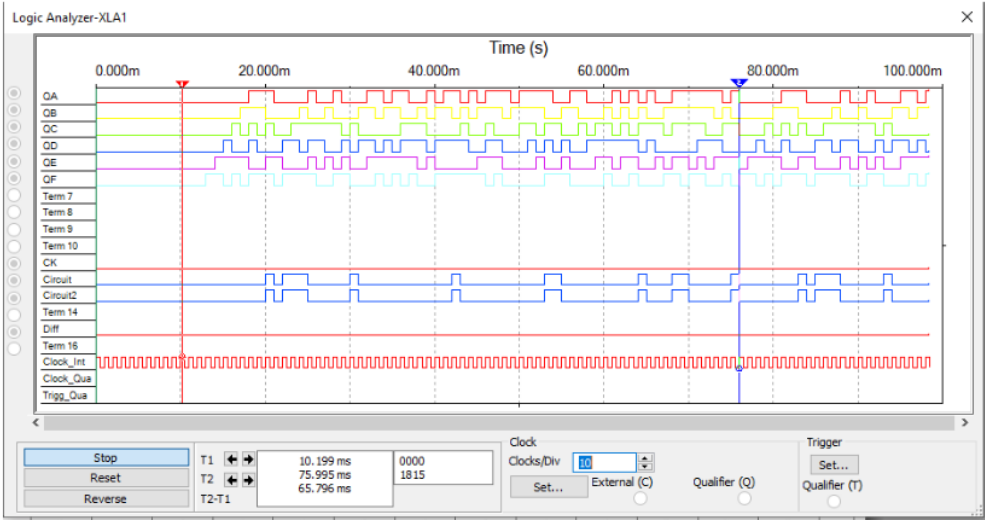


Figure 3-12 CUT no-fault condition waveform

No fault condition is demonstrated

4 ANALYSIS OF NI MULTISIM RESULTS

The CUT is tested against a good known working circuit. A differential between the circuits is picked up using the XOR gate as a comparator. The XOR gate is connected to the pin labelled *Diff* on the logic analyser.

4.1 FAULT 1 – QA STUCK AT 0

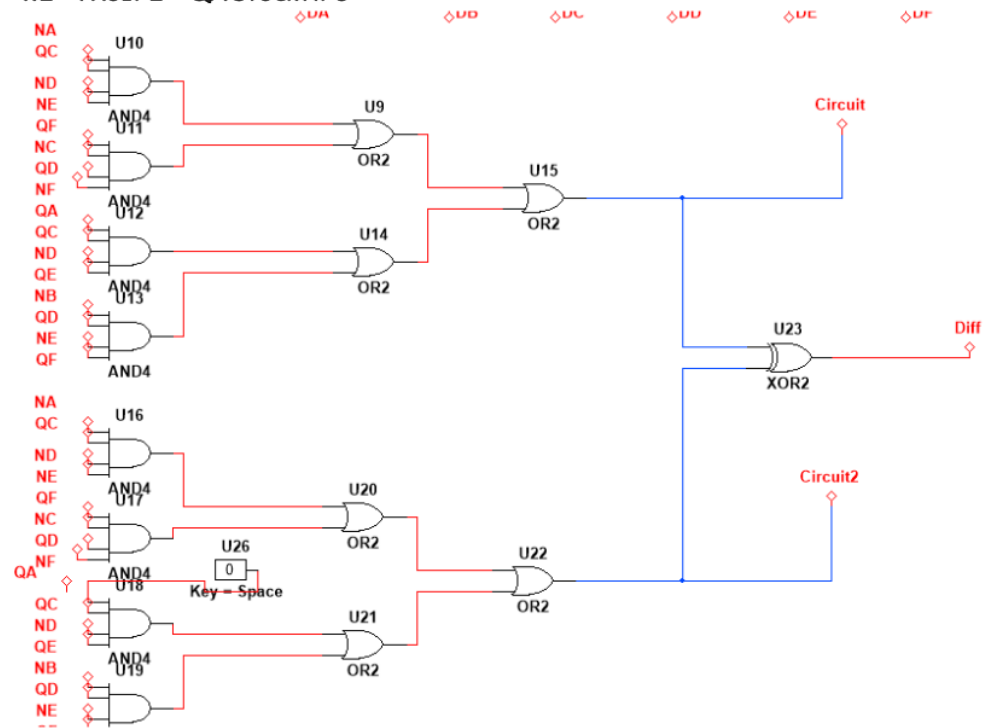
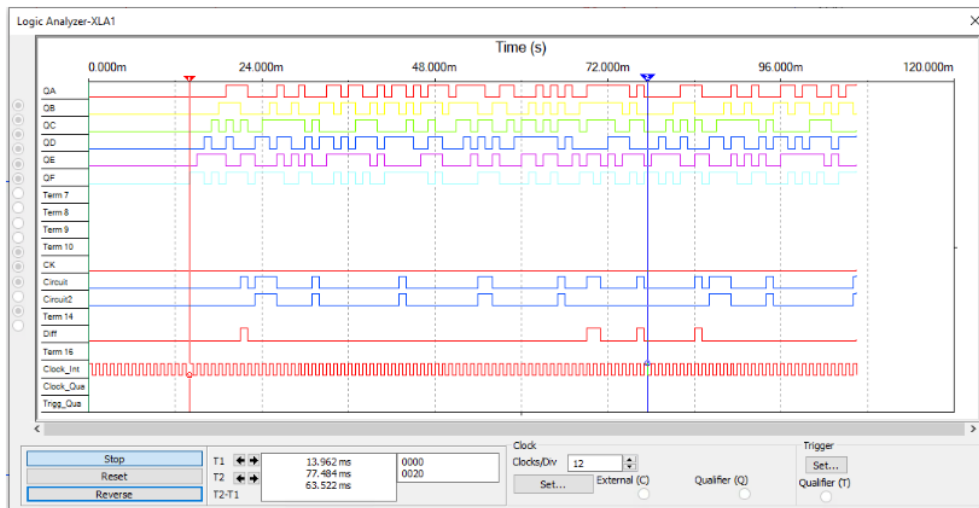


Figure 4-1 CUT QA SA0

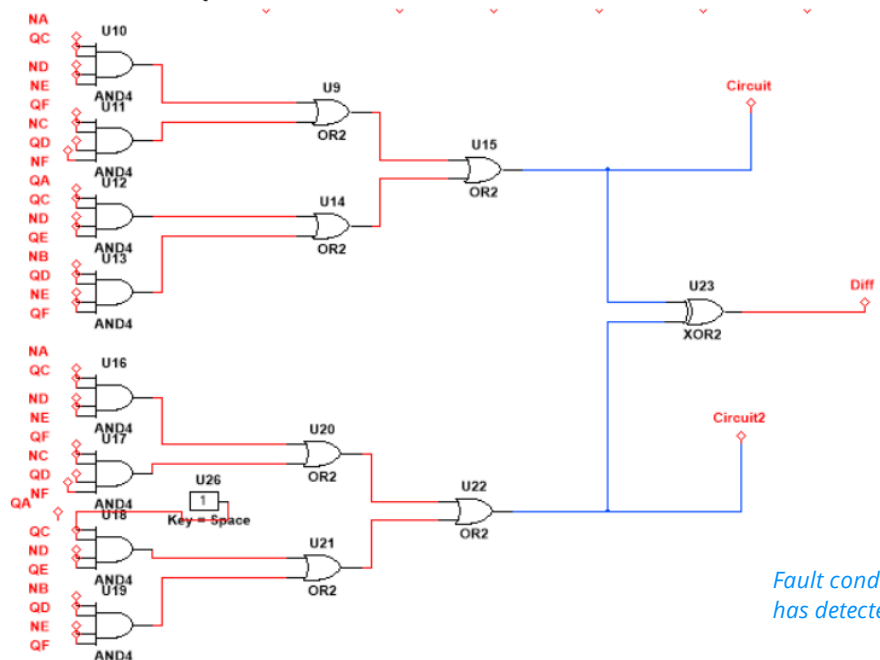


Fault condition is detected

Figure 4-2 CUT QA SA0 logic analyser

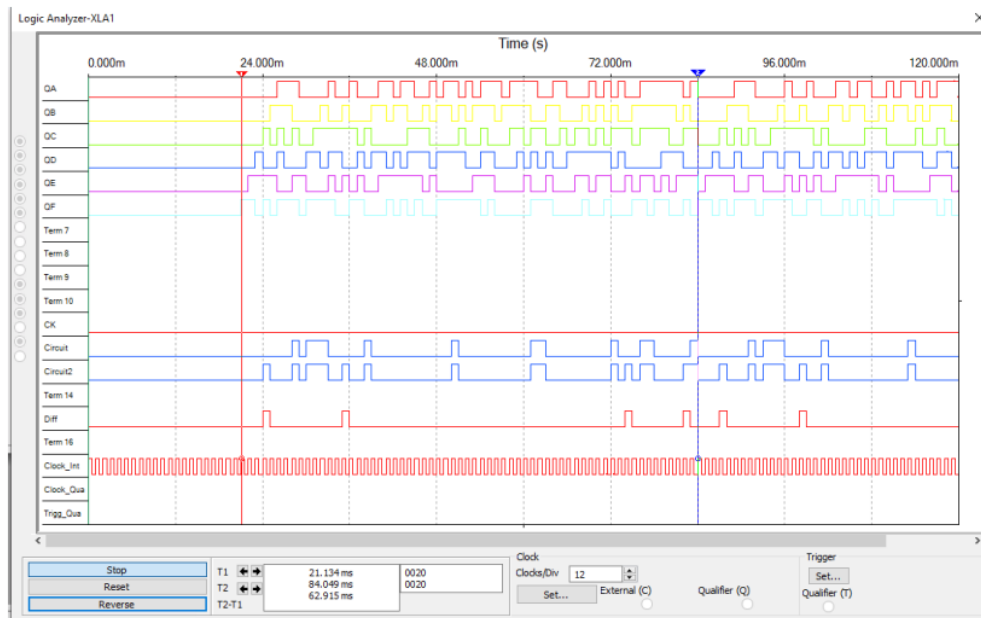
As shown in Figure 4-2, where the *Diff* input is at logic 1, this is where faults were detected by the XOR gate.

4.2 FAULT 2 – QA SA1



Fault condition is set and test result has detected this

Figure 4-3 CUT QA SA1



*Fault
condition is
detected*

Figure 4-4 CUT QA SA1 logic analyser

4.3 FAULT 3 – QC SA0

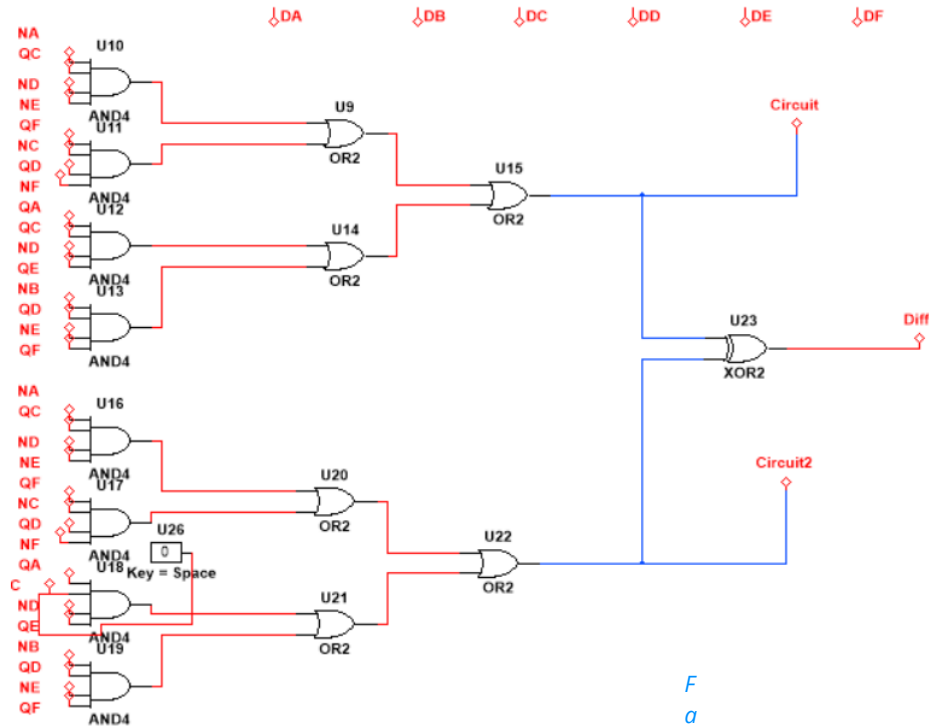


Figure 4-5 CUT QC SA0

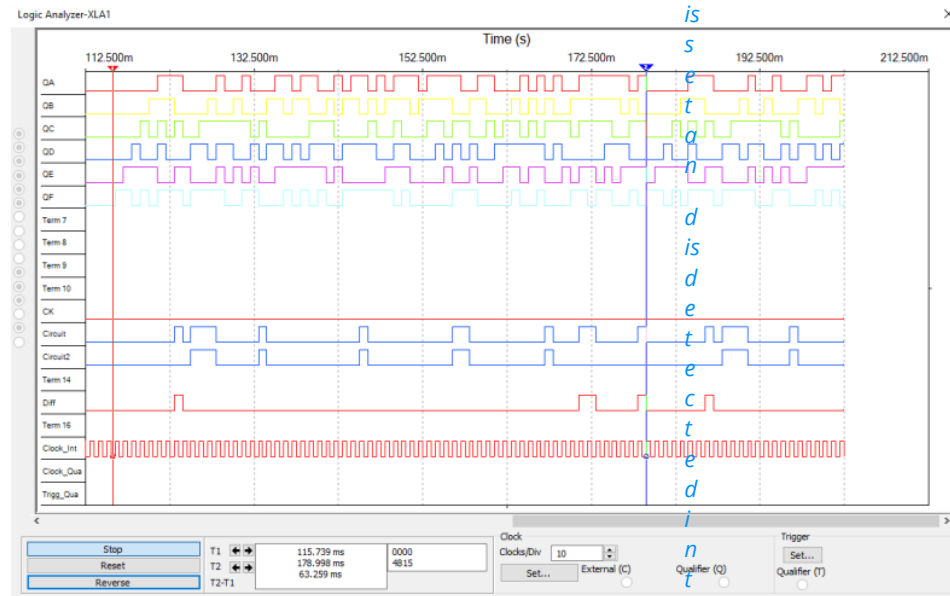


Figure 4-6 CUT QC SA0 logic analyser

F
a
u
l
t
i
s
d
e
t
e
c
t
e
d

Fault detected

The test circuit has picked up the fault in all three tests, according to the three fault results. For all three fault tests, a table containing the 6-bit hybrid CAR result and test circuit result for the fault closest to the end of the sequence has been created. This demonstrates why a fault occurred and how the Boolean expressions differ.

Fault	6-Bit Hybrid Outcome	Test Outcome	Fault
QA SA0	A.NB.C.ND.E.NF	NA.NB.C.ND.E.NF	Yes
QA SA1	NA.B.C.ND.E.F	A.B.C.ND.E.F	Yes
QC SA0	A.NB.C.ND.E.NF	A.NB.NC.ND.E.NF	Yes

Table 4-1 Fault comparison

*Good use of table
for analysis*

5 VHDL PROGRAMME ANALYSIS

5.1 4-BIT CAR 90

Figure 5-1 shows the entity for the CAR 90 VHDL program; the ports are mapped for the clock and clear function whilst also implementing the Q variables in standard logic.

```
entity cr90rule is
  port(
    clr : in STD_LOGIC;
    clk : in STD_LOGIC;
    q0,q1,q2,q3 : out std_logic
  );
end cr90rule;
```

Entity is correct

Figure 5-1 Declaring entity for CAR 90 ruling

The architecture for the CAR 90 logic is shown in Figure 5-2. The adjacent flip flop outputs are mapped on either side through an XOR gate on each of the four flip flops (D), which corresponds to the 90 ruling. For example, D(2) has qs3 and qs1 input into the XOR as these are mounted on either side.

```
--SET BEHAVIOR SEQUENCE
architecture Behavioral of cr90rule is
  signal D, qs: STD_LOGIC_VECTOR(3 downto 0);
begin
  --CAR 90 ALGORITHM
  D(3) <= '0' xor qs(2);
  D(2) <= qs(3) xor qs(1);
  D(1) <= qs(2) xor qs(0);
  D(0) <= qs(1) xor '0';

  -- SET CLK AND START SEQUENCE 0001 FOR 4 BIT
  process(clk, clr)
  begin
    if clr = '1' then
      qs <= "0001";
    elsif clk'event and clk = '1' then
      qs <= D;
    end if;
  end process;

  --SET OUTPUTS TO ARRAY ELEMENTS
  q0 <= qs(0);
  q1 <= qs(1);
  q2 <= qs(2);
  q3 <= qs(3);
```

Feedback algorithm is correct

Set sequence start is correct at 0001

Output array connections are correct

Figure 5-2 VHDL code for CAR90

For setting the clock and starting the flip flop circuit, the "SET CLK" code shown above is used. Before the sequence can begin, a bit of 1 must be set, and q0 must be set to 1, as shown in the code. A testbench code was developed to test the code.

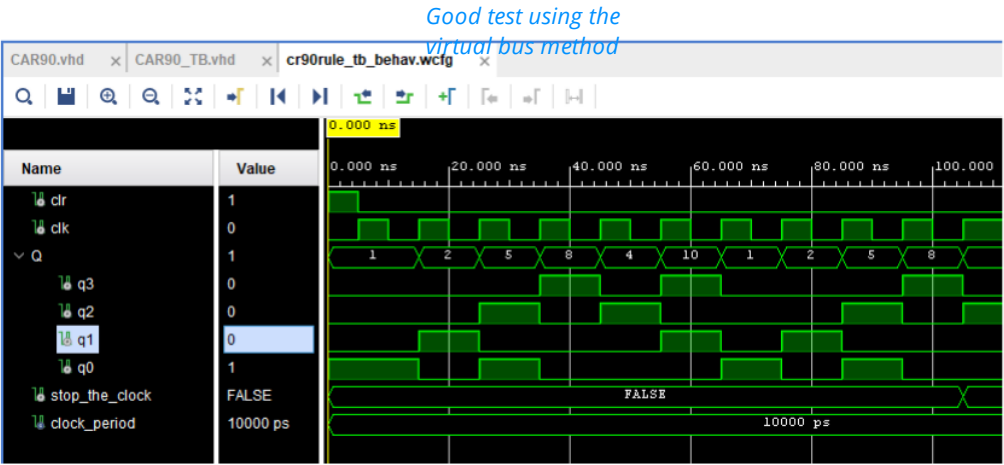


Figure 5-3 CAR 90 Testbench waveform

Figure 5-3 shows the simulation for the 4-bit CAR 90 using Vivado software. The count is seen as 1, 2, 5, 8, 4, 10. This matches the sequence of the Multisim 4-bit CAR 90

5.2 4-Bit CAR 150

The 150 ruling for the 4-bit CAR was simulated using Vivado software. The VHDL code for the CAR architecture is shown in Figure 5-4 below. The 150 ruling is denoted by the four flip flops, which include the state of the adjacent flip-flops as well as the current state of its own into the XOR gate.

```

--SET BEHAVIOUR SEQUENCE
architecture Behavioral of cr150_2 is
    signal D, qs: STD_LOGIC_VECTOR(3 downto 0);
begin
    --SET CAR 150 ALGORITHM SEQUENCE
    D(3) <= '0' xor qs(2) xor qs(3);
    D(2) <= qs(3) xor qs(2) xor qs(1);
    D(1) <= qs(2) xor qs(1) xor qs(0);
    D(0) <= qs(1) xor qs(0) xor '0';

    -- SET CLK AND START SEQUENCE 0001 FOR 4 BIT
    process(clk, clr)
    begin
        if clr = '1' then
            qs <= "0001";
        elsif clk'event and clk = '1' then
            qs <= D;
        end if;
    end process;

    --SET OUTPUTS TO ARRAY ELEMENTS
    q0 <= qs(0);
    q1 <= qs(1);
    q2 <= qs(2);
    q3 <= qs(3);

```

Entity is missing

Feedback algorithm is correct

Start sequence is correct at 0001

Output array is connected to output variables

Figure 5-4 CAR 150 VHDL Code

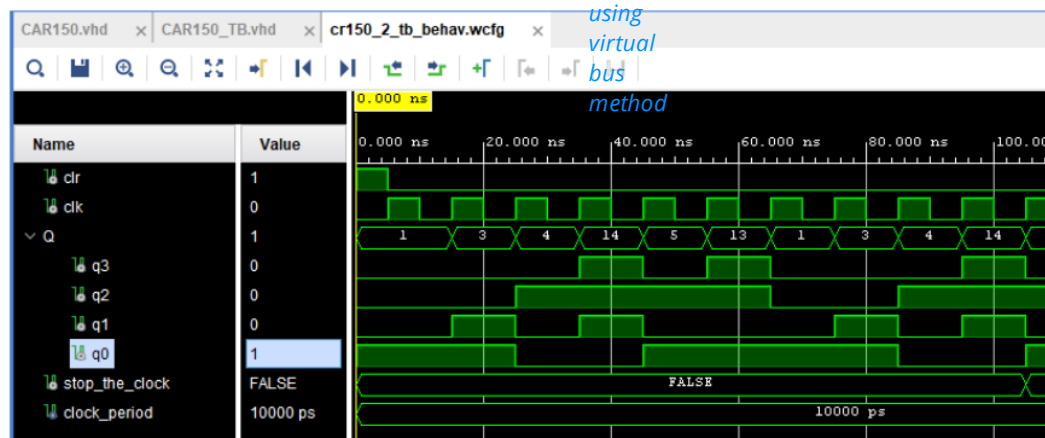


Figure 5-5 CAR 150 Testbench waveform

As shown above, the random count is 1,3,4,14,5,13, which matches that of the Multisim. However, the count is still not maximum.

5.3 4-BIT CAR 90/150 HYBRID

The Hybrid CAR code was added, as shown below.

Entity is missing

```
--SET THE HYBRID CAR 90/150 SEQUENCE ALGORITHM
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';

-- SET THE CLK AND START SEQUENCE USING 0001 FOR 4 BIT
process(clk, clr)
begin
    if clr = '1' then
        qs <= "0001";
    elsif clk'event and clk = '1' then
        qs <= D;
    end if;
end process;

q0 <= qs(0);
q1 <= qs(1);
q2 <= qs(2);
q3 <= qs(3);

end Behavioral;
```

Hybrid algorithm is correct

Start sequence is correct at 0001

Outputs are configured correctly

Figure 5-6 CAR 90/150 hybrid VHDL code

As shown in Figure 5-6, the 90 and 150 ruling is combined, alternating between for each flip-flop.

The clock and process start code is kept the same. The resultant testbench waveform can be seen in Figure 5-7.

Good test which shows the full random count

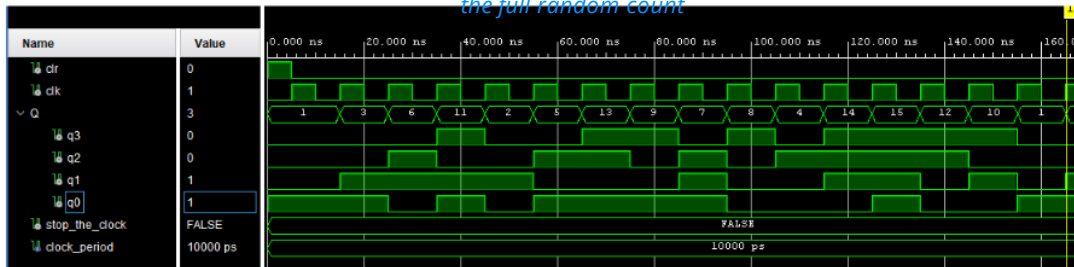


Figure 5-7 CAR 90/150 Testbench waveform

The sequence is now 1, 3, 6, 11, 2, 5, 13, 9, 7, 8, 4, 14, 15, 12, 10. This completes the maximum sequence of 15, matching the Multisim simulation sequence.

5.4 FIVE KHZ CLOCK VIVADO

The 5KHz clock was implemented into Vivado; Figure 5-8 show the code and the testbench. In the code, the 100MHz clock is divided into intervals of 10,000; this is then divided by two in the test bench code to get the 5KHz. $100\text{ MHz} / 10,000 = 10\text{KHz}$. Only the positive cycle is taken from the pulse. As a result, you achieve a 5KHz clock.

```
begin

process(CLK100MHZ,Rst)
begin
if Rst = '1' then
templ <= '0';
cntl <= 0;
elsif rising_edge (CLK100MHZ) then
if(cntl= 10000) then
templ <= not templ;
cntl <= 0;
else
cntl <= cntl +1 ;
end if;
end if;

end process;
FiveKHz <= templ;

end Behavioral;
```

```
UUT : FiveKHz_Clock

Port Map ( CLK100MHZ => CLK100MHZ,
Rst => Rst,
FiveKHz => FiveKHz
);

--Clock Process definitions
CLK100MHZ_process :process
begin
CLK100MHZ <= '0';
wait for CLK100MHZ_period/2;
CLK100MHZ <= '1';
wait for CLK100MHZ_period/2;
end process;
```

Figure 5-8 5KHz VHDL Code and testbench

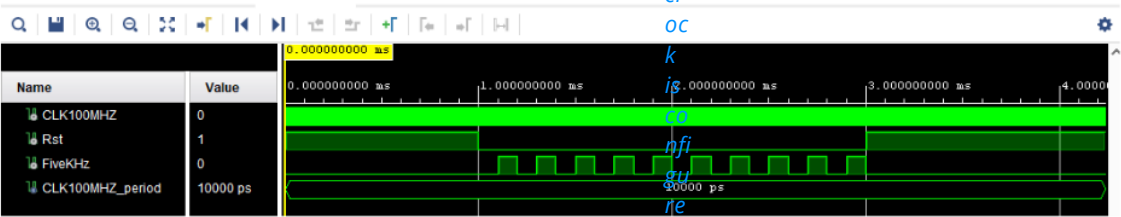


Figure 5-9 5KHz clock waveform - 1/0.2ms = 5KHz.

5
K
H
Z
cl
oc
k
is
CO
nfi
su
re
d
Test
shows
the
5KHz
signal

5.5 6-BIT HYBRID

Following the 90/150 ruling, the 6-bit hybrid CAR was added to Vivado. The alternating between 90 and 150 ruling, as seen in the figures below, is the most important part of the code for the six-bit Hybrid. The flip flops D5 to D0 start with 90, states on both sides as XOR inputs, then the 150 ruling with the same and also the current state. This is done for the remaining six flip-flops.

```
entity hybrid_6_bit is
    port(
        clr : in STD_LOGIC;
        clk : in STD_LOGIC;
        q0,q1,q2,q3,q4,q5 : out std_logic
    );
end hybrid_6_bit;
```

Entity is correct

Figure 5-10 6 Bit entity

```
architecture Behavioral of hybrid_6_bit is
    signal D, qs: STD_LOGIC_VECTOR(5 downto 0);
begin

    D(5) <= '0' xor qs(4);
    D(4) <= qs(5) xor qs(4) xor qs(3);
    D(3) <= qs(4) 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';
```

Algorithm is extended to 6 bit

Figure 5-11 Architecture of 6-Bit CAR

As seen above, the architecture has been kept the same as the previous CARs, in terms of it is XOR'd with the one previous, itself and its successor, but has been extended to accommodate the extra two bits.

Explanations are clear

```

-- Three D flip-flops
process(clk, clr)
begin
    if clr = '1' then
        qs <= "000001";
    elsif clk'event and clk = '1' then
        qs <= D;
    end if;
end process;

q0 <= qs(0);
q1 <= qs(1);
q2 <= qs(2);
q3 <= qs(3);
q4 <= qs(4);
q5 <= qs(5);

end Behavioral;

```

Start sequence is correct at 000001

Outputs correctly configured

Figure 5-12 6-Bit outputs

Figure 5-13 shows the timer element of the testbench code.

```

-- Put initialisation code here

clr <= '1';
wait for 5 ns;
clr <= '0';
wait for 500 ns;

-- Put test bench stimulus code here

stop_the_clock <= true;
wait;

end process;

clocking: process
begin
    while not stop_the_clock loop
        clk <= '0', '1' after clock_period / 2;
        wait for clock_period;
    end loop;
    wait;
end process;

```

Timer for test bench as this can cause the count to be incomplete in the test

Figure 5-13 6-Bit testbench code

The important parts of this code are the clr and wait commands. They are critical for testing. These times were changed to 2ns and 700ns to get the whole count in the testbench 0 - 63.

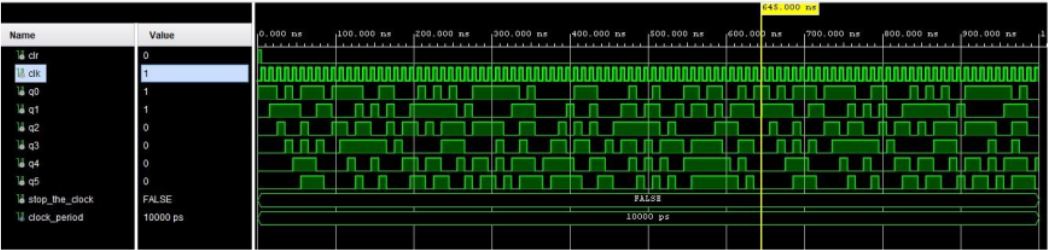


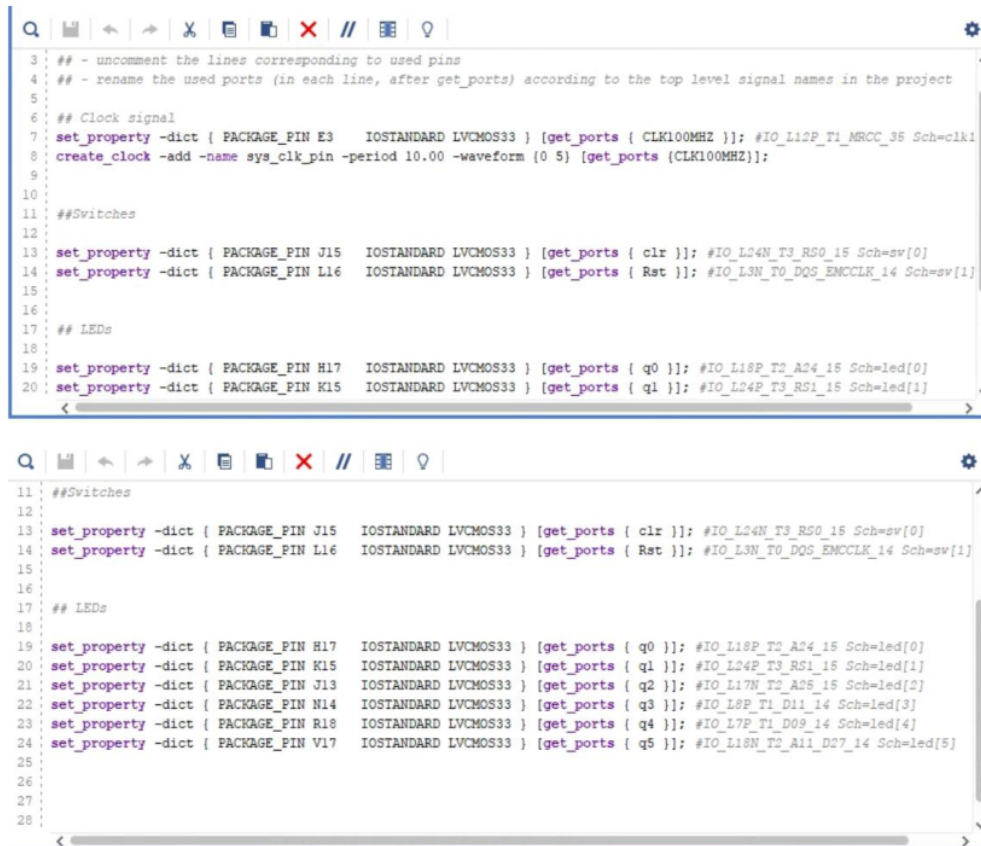
Figure 5-14 6-Bit Hybrid CAR waveform

Modified timer has given the full random count in the test

6 IMPLEMENTATION AND TESTING

The IP circuit was built on Vivado specifically for the Nexys board it will be implemented on. IPs were created for the hybrid 6-bit CAR and 5KHz clock, along with the AND, OR and XOR gates.

The I/O were mapped to a constraints file, as seen in Figure 6-1.



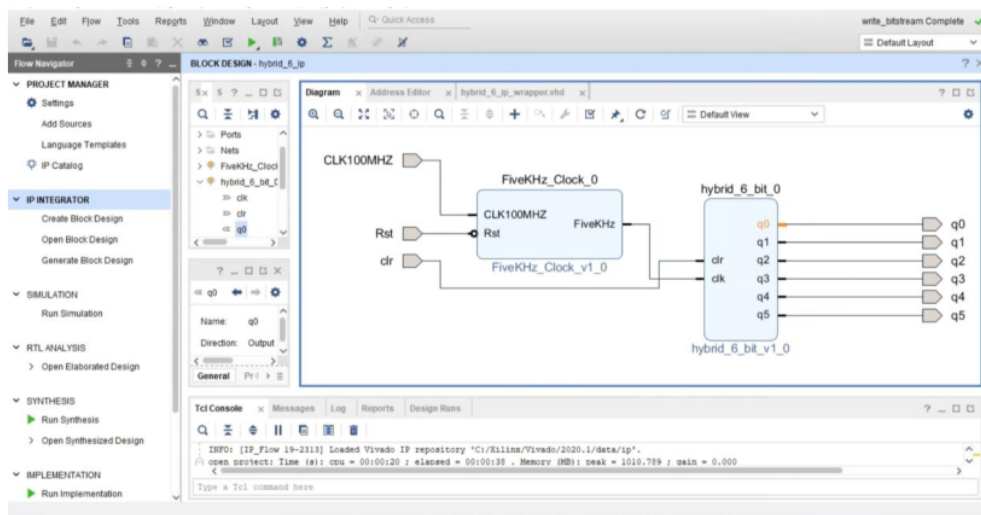
```
3 ## - uncomment the lines corresponding to used pins
4 ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6 ## Clock signal
7 set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk1
8 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
9
10
11 ##Switches
12
13 set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { clr }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
14 set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { Rst }]; #IO_L3N_T0_DQS_ENOCCLK_14 Sch=sv[1]
15
16
17 ## LEDs
18
19 set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { q0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
20 set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports { q1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
21
22
23
24
25
26
27
28
```

```
11 ##Switches
12
13 set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { clr }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
14 set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { Rst }]; #IO_L3N_T0_DQS_ENOCCLK_14 Sch=sv[1]
15
16
17 ## LEDs
18
19 set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { q0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
20 set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports { q1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
21 set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 } [get_ports { q2 }]; #IO_L17N_T2_A25_15 Sch=led[2]
22 set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports { q3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
23 set_property -dict { PACKAGE_PIN R18      IOSTANDARD LVCMOS33 } [get_ports { q4 }]; #IO_L7P_T1_D09_14 Sch=led[4]
24 set_property -dict { PACKAGE_PIN V17      IOSTANDARD LVCMOS33 } [get_ports { q5 }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
25
26
27
28
```

Figure 6-1 Constraints file I/O

The 6-bit Hybrid was made into a custom wrapper and then downloaded onto the board, as shown in the below figures. Figure 6-3 shows this is working.

Constraints file is set up correctly for the ports



A good design for the hybrid cct

Figure 6-2 6-Bit custom wrapper

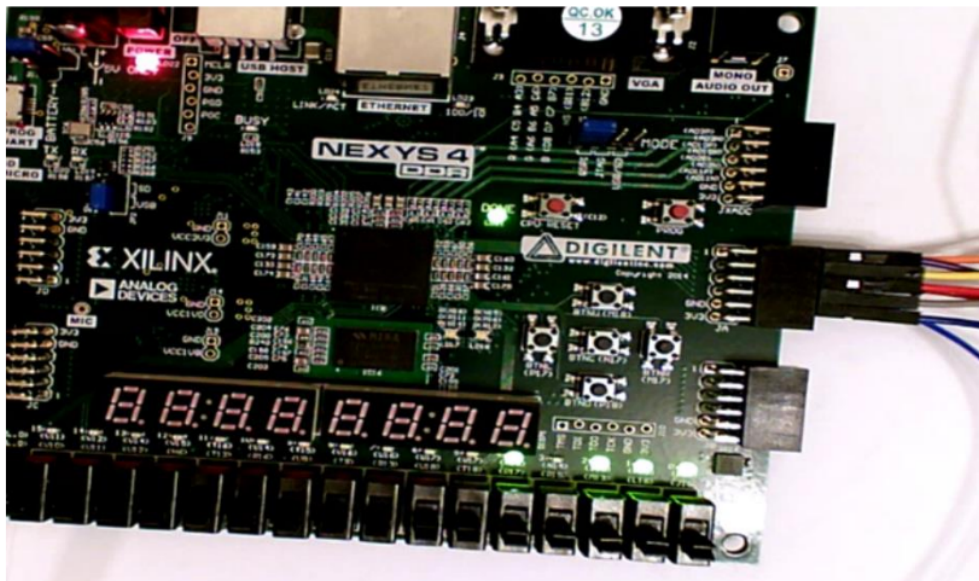


Image shows test evidence

Figure 6-3 6-Bit count

6.1 PCB TESTING

All individual IPS added into the one project as shown below.

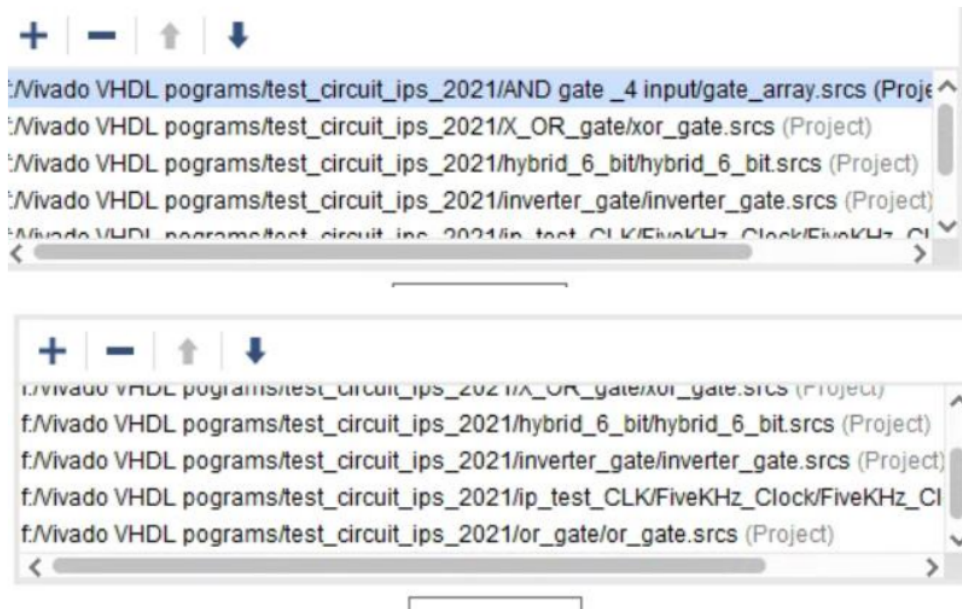


Figure 6-4 IPs added into one project

IP sources form VHDL

These were then connected to satisfy the Boolean equation for the 6-bit Hybrid, as seen in Figure 6-5.

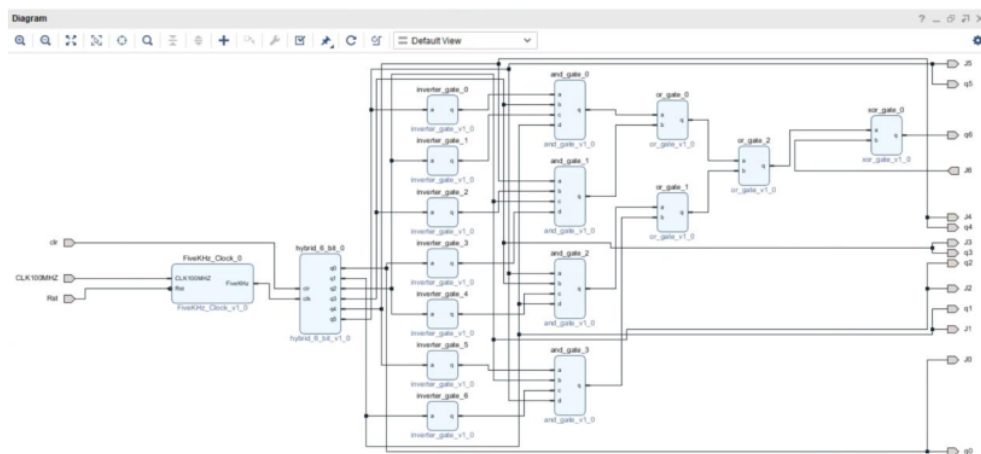


Figure 6-5 Final Circuit

IP cct design is correct and the X-OR is set up to take the practical cct output for comparison

The constraints file to map the IO can be seen below.

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in
the project
```

```

## Clock signal
set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];

##Switches

set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { clr }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { Rst }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]

## LEDs

set_property -dict { PACKAGE_PIN H17      IOSTANDARD LVCMOS33 } [get_ports { q0 }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15      IOSTANDARD LVCMOS33 } [get_ports { q1 }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13      IOSTANDARD LVCMOS33 } [get_ports { q2 }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14      IOSTANDARD LVCMOS33 } [get_ports { q3 }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18      IOSTANDARD LVCMOS33 } [get_ports { q4 }];
#IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17      IOSTANDARD LVCMOS33 } [get_ports { q5 }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN V11      IOSTANDARD LVCMOS33 } [get_ports { q6 }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

##Pmod Headers

##Pmod Header JA

set_property -dict { PACKAGE_PIN C17      IOSTANDARD LVCMOS33 } [get_ports { J0 }];
#IO_L20N_T3_A19_15 Sch=ja[1]
set_property -dict { PACKAGE_PIN D18      IOSTANDARD LVCMOS33 } [get_ports { J1 }];
#IO_L21N_T3_DQS_A18_15 Sch=ja[2]
set_property -dict { PACKAGE_PIN E18      IOSTANDARD LVCMOS33 } [get_ports { J2 }];
#IO_L21P_T3_DQS_15 Sch=ja[3]
set_property -dict { PACKAGE_PIN G17      IOSTANDARD LVCMOS33 } [get_ports { J3 }];
#IO_L18N_T2_A23_15 Sch=ja[4]
set_property -dict { PACKAGE_PIN D17      IOSTANDARD LVCMOS33 } [get_ports { J4 }];
#IO_L16N_T2_A27_15 Sch=ja[7]
set_property -dict { PACKAGE_PIN E17      IOSTANDARD LVCMOS33 } [get_ports { J5 }];
#IO_L16P_T2_A28_15 Sch=ja[8]
set_property -dict { PACKAGE_PIN F18      IOSTANDARD LVCMOS33 } [get_ports { J6 }];
#IO_L22N_T3_A16_15 Sch=ja[9]

```

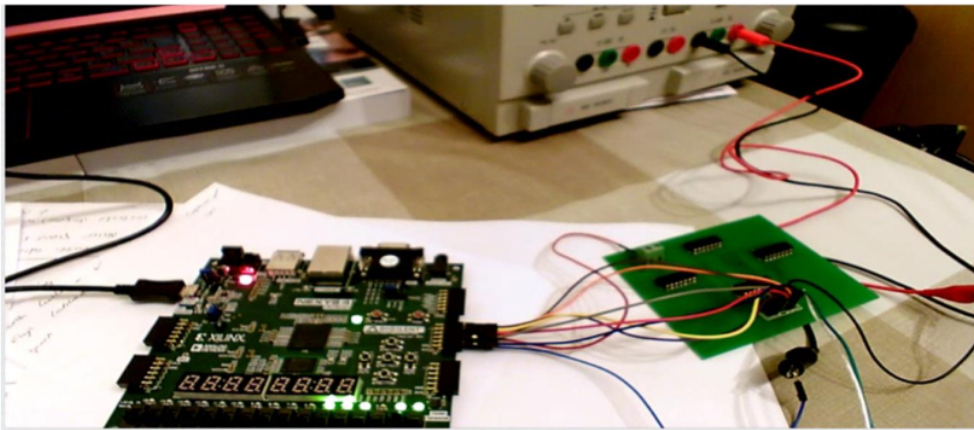
Port
mapping is
correct

```
#set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { JA10 }];  
#IO_L22P_T3_A17_15 Sch=ja[10]
```

Figure 6-6 Constraints file code

The difference in the code displayed in Figure 6-6 is the addition of the Pmod ports to allow the Nexys board to be connected to the PCB board, which is under test if a fault is detected, LED 15 (far left) illuminates.

The Equipment was set up as shown in Figure 6-7. The test results are shown in Figure 6-8 to Figure 6-13.



Good image
of test setup

Figure 6-7 Equipment set up

6.1.1 Fault 1 – Q1 SA0

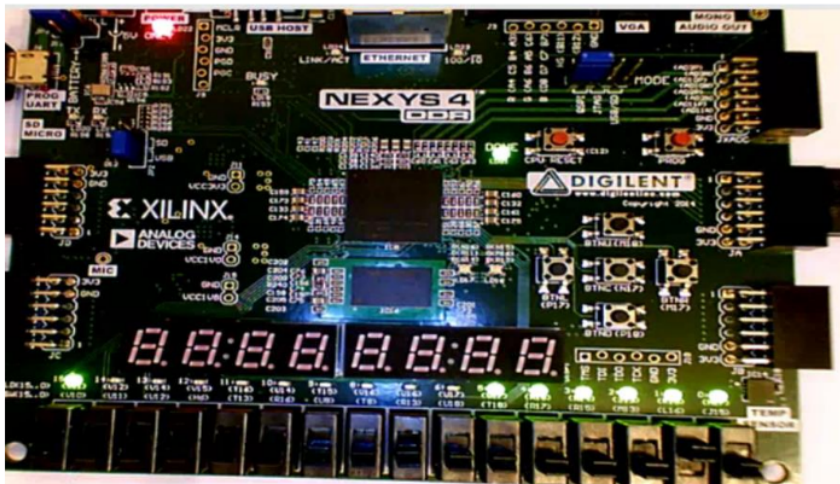


Figure 6-8 Q1 SA0 – Fault

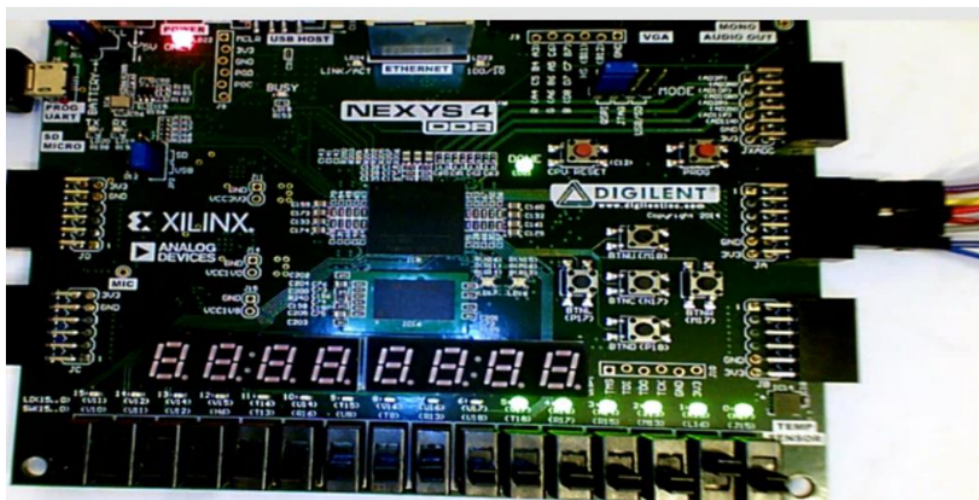


Figure 6-9 Brown in – Fault cleared

6.1.2 Fault 2 – QA SA1

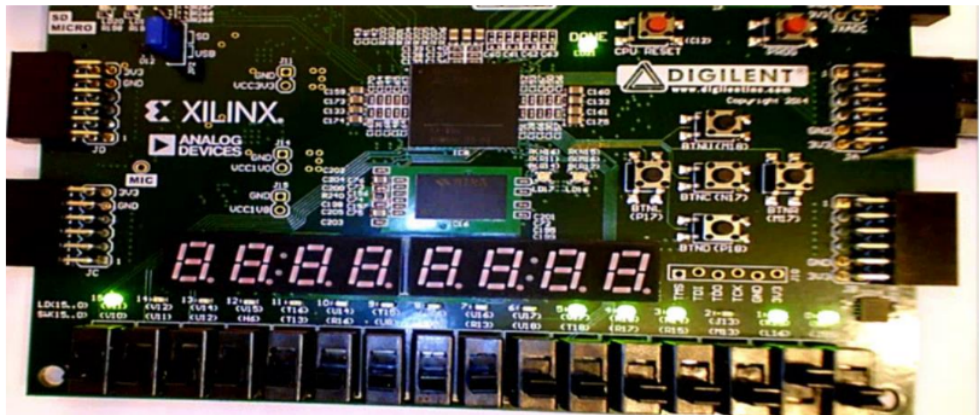


Figure 6-10 QA SA1 - Fault

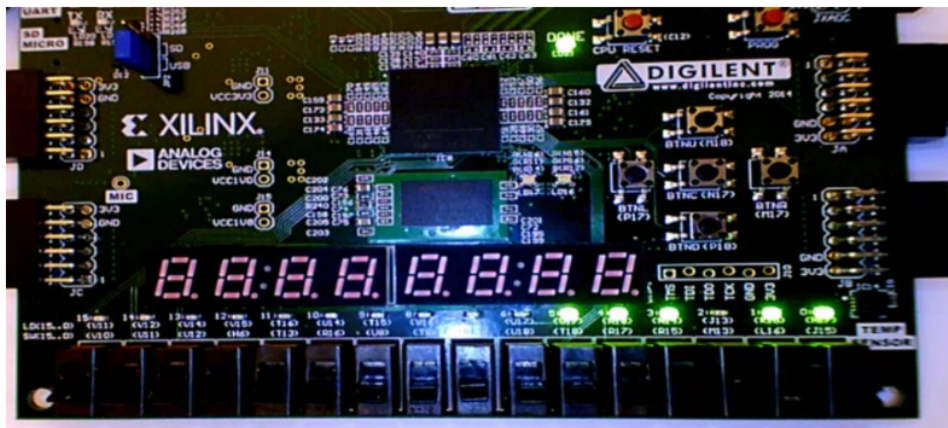


Figure 6-11 QA SA1 - Fault rectified

6.1.3 Fault 3 – QC SA0

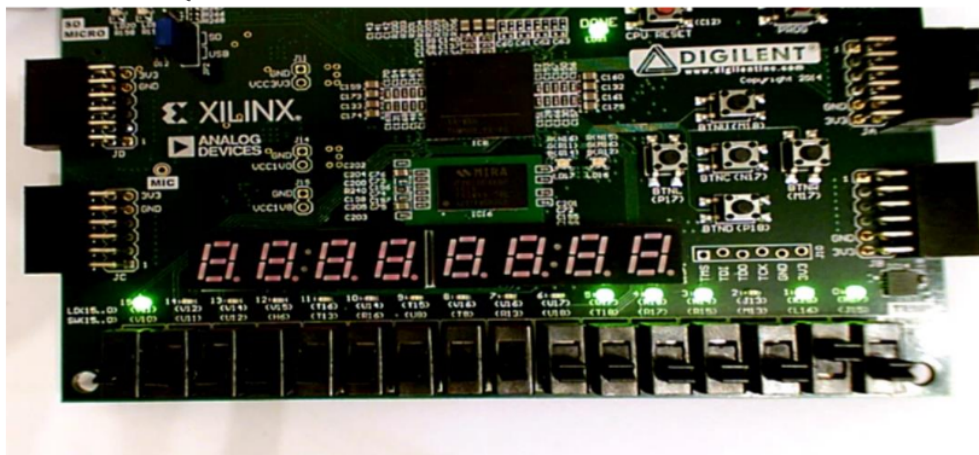


Figure 6-12 QC SA0 - fault

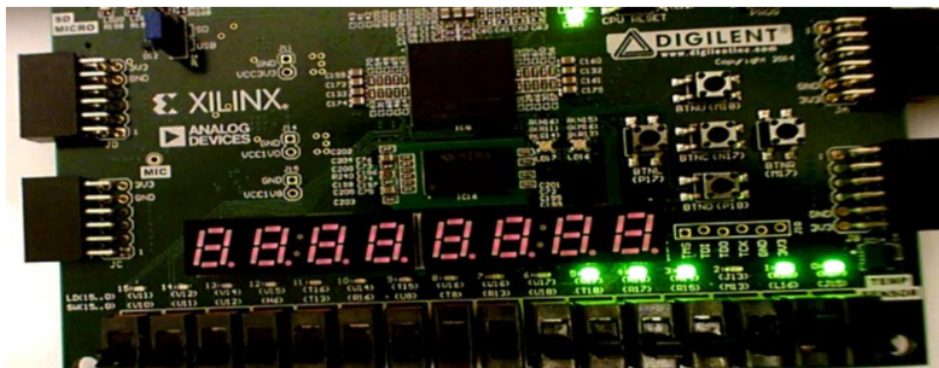


Figure 6-13 QC SA0 – fault rectified

7 CONCLUSION

Simulation, testing and research have been used throughout this assignment. The research looked into PCB design and manufacturing techniques, as well as some of the issues that these processes can cause. Image processing methods were also investigated, with a focus on their application in PCB automated manufacturing.

Using Multisim simulation, 90 and 150 ruling CAR circuits were designed and developed, eventually leading to a simulated test circuit for the 6-bit hybrid CAR. This was tested and evaluated using a Boolean logic test circuit to successfully detect the multiple faults implemented, which were detected using an XOR gate.

VHDL code was used in the Vivado software to create and progress through the 4-bit CAR circuits, eventually resulting in the 6-bit hybrid CAR. This circuit was broken down into IP blocks and downloaded onto the Nexys DDR board using the Boolean logic expression. Overall, the Nexys board was used to detect fault differences between it and the PCB with the 6-bit Hybrid implemented on it, resulting in successful testing.

All test circuits built performed as expected; the non-hybrid circuits were unable to achieve the maximum count where the 90/150 hybrid could. This result ties in with research and CAR design expectations.

A good conclusion

8 BIBLIOGRAPHY

Boomgaar, R., 2017. *Image Stretching*. [Online]

Available at:

<https://staff.fnwi.uva.nl/r.vandenboomgaard/IPC20172018/LectureNotes/IP/PointOperators/ImageStretching.html>

[Accessed 1st May 2021].

Chavan, R. R. et al., 2016. Quality Control of PCB using Image Processing. *International Journal of Computer Applications*, 141(5), pp. 28-32.

Crispin, A. J. & Rankov, V., 2006. *Automated inspection of PCB components using a genetic algorithm template-matching approach*. London: Springer.

Fisher, R., Perkins, S., Walker, A. & Wolfart, E., 2003. *Intensity Histogram*. [Online]

Available at: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/freqfilt.htm>

[Accessed 1st May 2021].

Fisher, R. P. S. W. A. & W. E. 2., 2003. *Frequency Filters*. [Online]

Available at: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/freqfilt.htm>

[Accessed 1st May 2021].

Huang, T. S. & Aizawa, K., 1993. Image Processing: Some Challenging Problems. *Images of Science: Science of Images*, Volume 90, pp. 9766-9769.

Isaac Newton Institute, 2017. *Computational Challenges in Image Processing*. [Online]

Available at: <https://gateway.newton.ac.uk/event/ofbw32>

[Accessed 1st May 2021].

Keyence, 2017. *AI Series Pattern Matching Sensor Catalogue*. [Online]

Available at:

https://www.keyence.com/mykeyence/downloadFromDLList?downloadAssetId=AS_88117&downloadZipFlag=0

[Accessed 1st May 2021].

Nayyer, K. & Sharma, B., 2015. Use and Analysis of Colour Modes in Image Processing. *International Journal of Advances in Scientific Research*, 1(08), pp. 329-330.

Nevada, B., 2014. An Image Processing Technique for Colour Detection and Distinguish Patterns with Similar Colour: An aid for Colour Blind People. *International Conference on Circuits, Communication, Control and Computing*, pp. 333-336.

*A good
number of
relevant
sources*

FINAL GRADE

GENERAL COMMENTS

Instructor

72/100

PAGE 1

PAGE 2

Text Comment. Good level of detail in the table of contents

PAGE 3

PAGE 4

Text Comment. Extensive table of figures

Text Comment. Good use of tables for comparison and analysis

PAGE 5

Text Comment. Good summary that includes all the significant tasks

PAGE 6

Text Comment. Good supporting image

PAGE 7

Text Comment. This should have a heading

PAGE 8

Text Comment. Good explanation supported with relevant sources

Text Comment. Good example of error detection in a real world application. Pattern matching is explained clearly.

Text Comment. Good explanation

PAGE 9

Text Comment. Good explanation

Text Comment. Good example application for this research section

Text Comment. The array here can be expanded for better resolution which would give better colour saturation sensitivity. This should have been explained

PAGE 10

Text Comment. Good image which supports explanation

Text Comment. Good coverage of histogram equalisation to improve image quality

PAGE 11

Text Comment. A good example

PAGE 12

Text Comment. CAR formula is identified

PAGE 13

Text Comment. CAR 90 characteristics are identified

Text Comment. A good design and test of the CAR 90 system which shows the count is not maximum

PAGE 14

Text Comment. CAR 150 characteristics are identified

Text Comment. Design and test demonstrate that the count is not maximum

PAGE 15

Text Comment. The hybrid system is identified and theory shows the count should be random and maximum.

PAGE 16

Text Comment. Design is correct

Text Comment. The waveform shows the full and random count

PAGE 17

Text Comment. 6 bit hybrid system is designed and configured correctly

Text Comment. Outputs of both ccts are identical here. No fault condition

Text Comment. The system shows maximum count is achieved

PAGE 18

Text Comment. The test cct has no fault implemented and the test shows this

Text Comment. No fault condition is demonstrated

PAGE 19

PAGE 20

Text Comment. Fault condition is detected

Text Comment. Fault condition is set and test result has detected this

PAGE 21

Text Comment. Fault condition is detected

PAGE 22

Text Comment. Fault is set and is detected in the test results

Text Comment. Fault detected

PAGE 23

Text Comment. Good use of table for analysis

PAGE 24

Text Comment. Entity is correct

Text Comment. Feedback algorithm is correct

Text Comment. Set sequence start is correct at 0001

Text Comment. Output array connections are correct

PAGE 25

Text Comment. Good test using the virtual bus method

PAGE 26

Text Comment. Entity is missing

Text Comment. Feedback algorithm is correct

Text Comment. Start sequence is correct at 0001

Text Comment. Output array is connected to output variables

Text Comment. Good test using virtual bus method

PAGE 27

Text Comment. Entity is missing

Text Comment. Hybrid algorithm is correct

Text Comment. Start sequence is correct at 0001

Text Comment. Outputs are configured correctly

Text Comment. Good test which shows the full random count

PAGE 28

Text Comment. 5 KHZ clock is configured

Text Comment. Test shows the 5KHZ signal

PAGE 29

Text Comment. Entity is correct

Text Comment. Algorithm is extended to 6 bit

Text Comment. Explanations are clear

PAGE 30

Text Comment. Start sequence is correct at 000001

Text Comment. Outputs correctly configured

Text Comment. Timer for test bench as this can cause the count to be incomplete in the test

PAGE 31

Text Comment. Modified timer has given the full random count in the test

PAGE 32

Text Comment. Constraints file is set up correctly for the ports

PAGE 33

Text Comment. A good design for the hybrid cct

Text Comment. Image shows test evidence

PAGE 34

Text Comment. IP sources form VHDL

Text Comment. IP cct design is correct and the X-OR is set up to take the practical cct output for comparison

PAGE 35

PAGE 36

Text Comment. Port mapping is correct

Text Comment. Good image of test setup

Text Comment. PCB tests show the random count for 6 bit CAR and the fault detection LED set and cleared

Text Comment. A good conclusion

Text Comment. A good number of relevant sources

GRADING FORM: CSG-RUBIC

JONATHON WILLIAMS

0

RESEARCH-INFORMED LITERATURE



Machine vision and pattern matching have been covered in good detail with good relevant sources used in the text. 7.5/10 Colour recognition researched the possibilities of colour depth recognition and use a good example to support this. The array could have been explained further. Good relevant sources used. 6.5/10

14/20

KNOWLEDGE AND UNDERSTANDING



Design of Msim 6 bit CAR using the prototype method with each stage tested proved to be a good method and the resulting design was proven in the test stage. The VHDL design followed the same pattern to create the final 6 bit design. The system design tests proved the system. 8/10 IP design using the FPGA development board implemented the IP components derived from the VHDL stage of development. The IP gate designs were created using the same method. Not covered in report due to simplicity, but there should be an example in the appendix. 7/10

15/20

ANALYSIS



Test results analysis of 6 bit hybrid CAR showed that the random and maximum count could be achieved only by combining the two rules 90 and 150. The theory showed that this was the case and was proved through testing carried out. The explanations are fairly accurate about this. 6/8 There is a comprehensive analysis of the FPGA results which show operational conditions.. 5.5/8 Analysis of PCB test results has covered the fault set and cleared conditions as required. 2.5/4

14/20

PRACTICAL APPLICATION AND DEP.



CAD implementation and testing of prototype designs has been carried out using a good sequential method and enabled the final hybrid design to be implemented. The test procedures showed how the single rule systems could be implemented as a hybrid and used to drive a test

19/25

cct. 8/10 FPGA implementation required the design of SOC components for an IP system design. This has been carried out firstly with errors and through test and analysis the errors were resolved to give a good working end product that was used with the practical testing. 11/15

SKILLS FOR PROFESSIONAL PRAC.



Deployment of the software and on line resources has been carried out to a very good standard. 3.5/5
Report layout is good and all parts of the report are in the correct sequence. 3.5/5
Conclusion covers all the main point. 3/5

10/15