7 PROGRAMMING THE FPGA

Simon Bevan

7.1 VHDL

The FPGA chip can be programmed using a language called VHDL. VHDL is a hardware description language for describing digital designs. It originated from a government programme in the development of Very High Speed Integrated Circuits. VHDL is like a general programming language with extensions to model both concurrent and sequential flows of execution and the concept of delayed assignment of values. The code has a very unique structure, which is related to the fact that it is still part of a circuit.

7.2 The Coincidence Logic

By numbering the scintillators as follows (fig.77) the logic for the VHDL can be discussed.

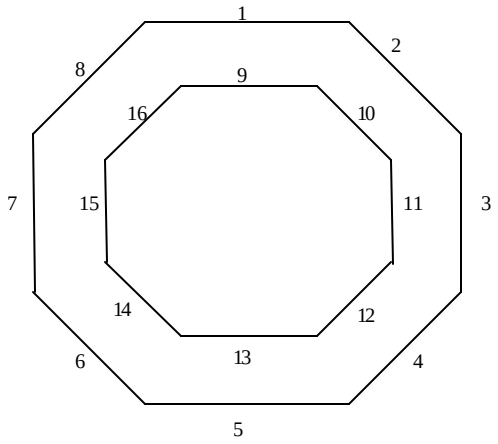

Fig 77: Numbering of scintillaors

It was originally proposed that there could be eight possible logics that could be programmed, each having different count rates. All could be tested and the most suitable chosen.

Preliminary ideas on the logic - The eight possible logics

The first logic is to consider every possible direction that a muon could strike from. This logic allows muons to be detected from all the possible geometries that the detector allows. For example, for muons hitting scintillator 1, there are five possible directions for 4 way coincidence (fig.78, overleaf)

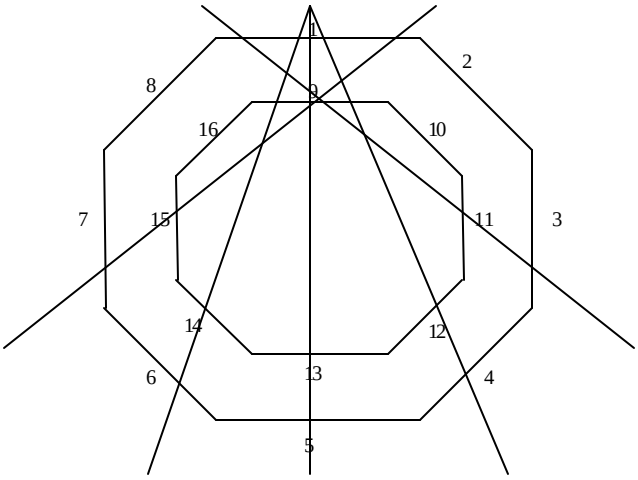

Fig 78 : Possible directions for muons going through scintillator 1

Similarly for muons hitting scintillator 8, there are four possible directions for 4-way coincidence.

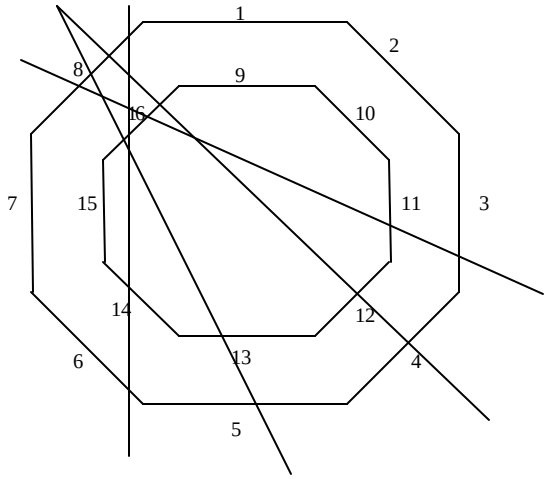

Fig 79 : Possible directions for muons going through scintillator 2

This logic allows for it to be further split: -

All scintillators
- Both scintillator layers
- Inner/Outer Ring

Top Half of Detector
- Both layers

where the top half of the detector means that only the scintillators in the top half 1, 2, 3, 7, 8, 9, 10, 11, 15, 16 of the detector are struck and the results extrapolated.

Another possible detection logic would be to only consider the directions of 0, 45, 90, and 135 degrees. This can be broken down the same way as above but there would be a total of 4 different coincidences only.

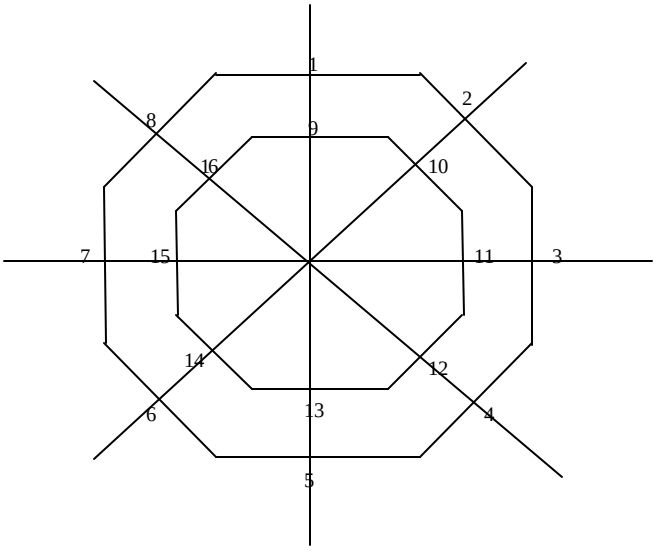

Fig 80 : Possible directions for 0, 45, 90, 135 degrees only setup

Analysis of the eight logics

It was decided that using only the top half and extrapolating would be misleading and inaccurate, as from fig 78 it can be seen that there are five possibilities for a coincidence between scintillators one and nine.

It was also decided that considering only the inner or outer ring would not be the best way to detect for two coincidences. The reason for considering two scintillators instead of four is so that the any efficiency that is lost in introducing more scintillators is reduced. The most efficient combination of scintillators will not be known until the detector is assembled and tested. In allowing logic that considers all possible 2-way coincidences, the most efficient combination will give a signal, and hence the number of particles detected in two way rather than four-way coincidence will increase. In defining the logic in this way, 3-way coincidence can also be considered.

The final decision that was made was to use the 0, 45, 90, 135-degree logic rather than all directions. The reason for this was that this allowed a code to be written that was easily upgradable rather than trying to write a complex code that had possibilities to go wrong. The limit of time was also a factor in this decision. Allowing for every possible track, 180 different coincidences would have to be defined rather than 36 for the 4-direction logic.

The LED Display

There were two possible designs that were considered: -

A Grid

This design involves a grid of LEDs arranged in a square. The LEDs, which match specific scintillator panels, will light when the scintillator is hit by a muon. This also allows for the muon path to be traced by the LEDs

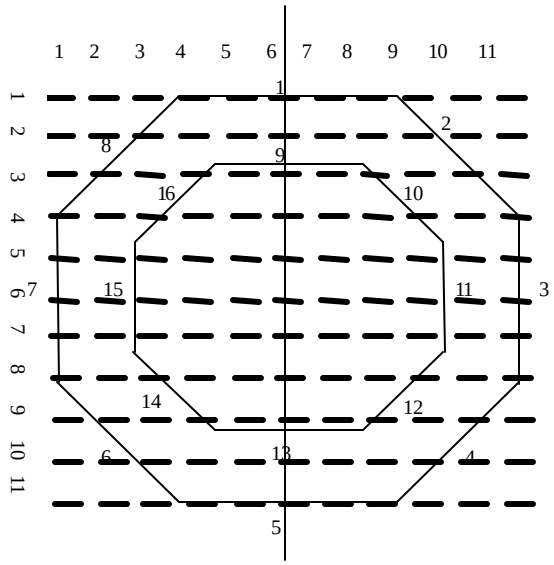

Fig 81: Possible LED set-up

This is demonstrated in fig 81 above. All LEDs would light up along path of the arrow. The number of LEDs does not have to be 121; any number could be used.

Octagon in the middle

For this design, each scintillator panel has an array of LEDs which corresponds to it and each track has a further three LEDs along the path. This is best achieved by having an array of LEDs in an octagon shape in the middle of the panel(see fig.82, overleaf).

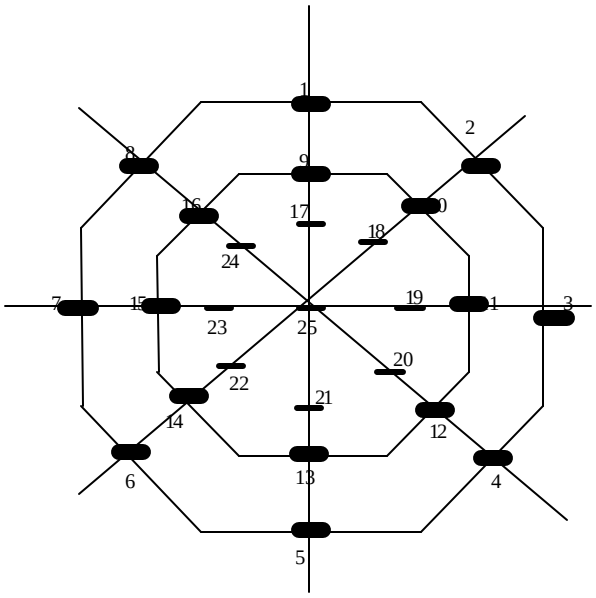

Fig 82: Octagonal LED set-up. The thick lines represent arrays of LEDs and the thin lines represent point LED's. The numbers represent the numbering given to the LEDs in the code.

Analysis of Designs

The design that was chosen was the octagon design. The main reason for this was that the number of LEDs was far less than the grid design. Although the grid design allows more scope for the extension to encompass all possible directions, the octagon design also allows for extra tracks but with some tracks not using five LEDs. In having only 25 LEDs, each LED could be driven off a separate pin on the FPGA, rather than having to drive two or three LEDs off one pin. This would have made the code more complex to write as careful mapping would have to be done to find the best combinations of LEDs. Also current budgeting had to be considered. Each LED draws current from the power supply, having large amounts of LEDs would draw a larger current from the supply. With the supply used in the final design, having large amounts of LEDs was not an option.

The LEDs were placed in the front plate of the detector. This meant that holes had to be drilled for each one. Again drilling large amounts of holes is time consuming, weakens the plate and is limited by the size of the plate. The octagon design has none of these problems. The holes were drilled all the way through the plate, and the LEDs slotted in from behind and glued into position.

7.3 Possible coincidences

For the logic that is described above the possible combinations of scintillators are: -
4way coincidence:
Track1 - 1, 9, 13, 5
Track2 - 2, 10, 14, 6

Track3 - 3, 11,15, 7
Track4 - 8, 16,12, 4

3way coincidence:

Track1 -	1, 9, 13	or 1, 9, 5	or 13, 5, 9	or13, 5, 1
Track2 -	2, 10, 14	or 2, 10, 6	or 14, 6, 2	or 14, 6, 10
Track3 -	3, 11, 7	or 3, 11, 15	or 7, 15, 3	or 7, 15, 11
Track4 -	8, 16, 12	or 8, 16, 4	or 12, 4, 8	or 12, 4, 16

2way coincidence:

Track1 -	1, 13	or 1, 5	or 9, 5	or9, 13
Track2 -	2, 14	or 10, 6	or2, 6	or 14, 10
Track3 -	3, 7	or 3, 15	or11, 15	or 7, 11
Track4 -	8, 12	or 8, 4	or 12, 16	or4, 16

N.B. In 2way coincidence logic, for the same reason as discounting the top half logic, the two possible coincidences of the scintillators opposite to each other, e.g. 1,9 or 5,13, are not included.

Having all of these possibilities gives the group a lot of redundancy on the final design. If the detector gives a count rate that is too high or too low, the logic can be adjusted accordingly.

The Veto

The original idea was that if a scintillator was struck, the surrounding scintillators would be turned off for a short amount of time. The thought behind this was that by turning the scintillators off it ensures that only the muon that has been detected first causes the coincidences. This was considered a misleading way of showing cosmic rays as it may exclude some of the rays. Instead of this method a veto was defined. The veto was defined to deal with the noise in a sensible way. The veto asks if 3 or more tracks are hit at the same time and if so, turns all LEDs off. This ensures that if there is a random noise event, instead of all the LED's flashing, they are all turned off and not counted.

Counting the Hits

One of the original criteria was to design a detector that could be used to find the angular flux of cosmic rays. It was therefore important to be able to use the detector to count the hits in all four directions so it was decided to define five counters. These counters are: -

1. No counter
2. Counting all hits in the vertical plane i.e. the coincidence of 1, 9, 13, 5 or corresponding 2, 3way coincidences.
3. Counting all hits in the 45 degree plane i.e. the coincidence of 2, 10, 14, 6 or corresponding 2, 3way coincidences. N.B. Only one 45 degree angle was used, rather than two and dividing by two.
4. Counting all hits in the horizontal plane i.e. the coincidence of 8, 12, 16, 4 or corresponding 2, 3way coincidences.

5. Counting the total number of hits. This sums the hits from the above three counters.

The buttons

The chip also has eight buttons, with each button representing a binary bit. There are therefore 256 different buttons that can be defined. For this experiment, sixteen of these buttons were used, each representing a type of logic (2, 3, or 4 way coincidence) and the type of counter required (no counting, and vertical, horizontal, 45 degree, and total hit counters). The sixteenth button represents a test code, where the possible tracks are cycled through sequentially. The buttons are defined as: -

Button	Function	Button	Function
10010000	4way coincidence with no counting	01000010	3way coincidence with horizontal counter
10001000	4way coincidence with vertical counter	01000001	3way coincidence with total hit counter
10000100	4way coincidence with 45 degree counter	10010000	2way coincidence with no counting
10000010	4way coincidence with horizontal counter	10001000	2way coincidence with vertical counter
10000001	4way coincidence with total hit counter	10000100	2way coincidence with 45 degree counter
01010000	3way coincidence with no counting	10000010	2way coincidence with horizontal counter
01001000	3way coincidence with vertical counter	10000001	2way coincidence with total hit counter
01000100	3way coincidence with 45 degree counter	11111111	Cycles through all tracks sequentially

Fig.83: The buttons and their function

7.4 Programming the logic in VHDL

For the full code see Appendix IX

Basic Structure of VHDL

Primary building blocks of VHDL models are entities and architectures. The entity details the interface of a component, while the architecture describes its function.

Entity – A declaration that defines the interface between the design and the outside world i.e. an analogy would be the pins in an integrated circuit.

```
entity coincide is
 port(
   clk_in        : in  std_logic;
   rst_in        : in  std_logic;
   hit_ch_in     : in  std_logic_vector(16 downto 1);
   button_in     : in  std_logic_vector(8 downto 1);
   leds_out      : out std_logic_vector(25 downto 1);
   veto_out      : out std_logic
   );

end coincide;
```

It can be seen from the above that all the in and out functions that the unit is to handle are defined. The std_logic means standard logic, where no numbers after means a one bit logic (on or off), and the downto code represents the number of bits. For example, the board has eight buttons that can be on or off, which is an 8-bit logic. The button in the code can be seen as being defined as - in std_logic_vector (8 down to 1).

Architecture – Required for each entity. Describes the relationship between the ports and the entity. All variables in the architecture are defined by a *Signal*, which is the link between the outside world and the chip world. All signals are internal and cannot be read as inputs or outputs.

```
architecture bhv of coincide is

 signal clk, rst : std_logic;
 signal veto             : std_logic;
 signal hit_ch           : std_logic_vector(16 downto 1);
 signal button           : std_logic_vector(8 downto 1);
 signal leds             : std_logic_vector(25 downto 1);
 signal hit_pair         : std_logic_vector(8 downto 1);
 signal track            : std_logic_vector(4 downto 1);
 signal counter          : std_logic_vector(31 downto 0);
 signal gate             : std_logic;
 signal rising           : std_logic;
 signal led_event                : std_logic;
 signal tebl_pb, tebl_led        : std_logic_vector(7 downto 0);
 signal tebl_disp        : std_logic_vector(15 downto 0);
 signal loopcount                : std_logic_vector(4 downto 0);
 signal led_on1                  : std_logic;
 signal led_off1         : std_logic;
 signal led_on2                  : std_logic;
 signal led_off2         : std_logic;
 signal led_on3                  : std_logic;
 signal led_off3         : std_logic;
 signal led_on4                  : std_logic;
 signal led_off4         : std_logic;
```

```
signal led_on_timer          : std_logic_vector(25 downto 0);
signal led_off_timer         : std_logic_vector(25 downto 0);
```

After all the variables have been defined the code can then be started. Firstly the inputs and outputs have to defined as signals, i.e. internal variables.

```
begin

 button <= button_in;

 clk     <= clk_in;
 rst     <= rst_in;
 veto_out <= veto;
 leds_out <= leds;
 hit_ch   <= hit_ch_in;
```

Next, the logic that the chip is going to execute needs to be defined. Here it is to be repeated every clock cycle. On the FPGA board this has a period of 20ns. The full code contains for every button the appropriate code for the coincidences that could happenand the counter. As an example, button “00100001” is going to be used . This represents a two way coincidence counting all signals.

```
 async_button : process (clk)
 begin

if button(8 downto 1) = "00100001" then

    track(1)   <= (hit_ch(1) and hit_ch(13)) or (hit_ch(1) and hit_ch(5)) or (hit_ch(9)
                  and  hit_ch(5)) or (hit_ch(9) and hit_ch(13));
    track(2)    <= (hit_ch(2) and hit_ch(6)) or (hit_ch(2) and hit_ch(14)) or
                  (hit_ch(10) and hit_ch(6)) or (hit_ch(10) and hit_ch(14));
    track(3)    <= (hit_ch(3) andhit_ch(7)) or (hit_ch(3) and hit_ch(15)) or
                  (hit_ch(11) and hit_ch(7)) or (hit_ch(11) and hit_ch(15));
    track(4)    <= (hit_ch(12) and hit_ch(16)) or (hit_ch(8) and hit_ch(12)) or
                  (hit_ch(8) and hit_ch(4)) or (hit_ch(16) and hit_ch(4));
```

This first section represents the coincidences. Firstly the unit asks which buttons are on. If this sequence of buttons is on, tracks 1 to 4 are defined, where track is an internal signal which will later be used to define the LEDs that are to be lit. Here it can be seen that there are four possible combinations of two scintillators for all tracks, as was defined earlier.

Next the counter is defined. What the code asks is if the the reset button on the FPGA has been hit. If so, set the counter to zero, if not, if there has been a hit on tracks 1, 2, or 3 so increment the counter by one.

```
 if rst='1' then

    counter<=(others=>'0');
    gate   <='0';
```

```
  rising <='0';

 elsif rising_edge(clk) then

  gate<=(track(1) or track(2) or track(3) or track(4) );

  if ((track(1)='1' or track(2)='1' or track(3)='1') and gate='0' and veto ='0') then
    rising<= '1';
  else
    rising<='0';
  end if;

  if rising='1' then
    counter<= counter+'1';

  end if;
 end if;


 tebl_disp<=counter(25 downto 10);
 tebl_led <= tebl_pb;
```

What the gate and rising terms do is to make sure that the counter is only incremented by one for every hit as the hits are going to be longer than one clock cycle. The tebl_disp, and tebl_led are the terms which send the counter to the display on the board.

For no counter buttons, the counter is set to *counter <= "00000000000000000000000000000000"*, which just means that the counter is always zero.

After all the possible coincidences have been defined, the correct coincidence need to shown on the LEDs.

```
 leds(1) <= track(1) and (not veto);
 leds(2) <= track(2) and (not veto);
 leds(3) <= track(3) and (not veto);
 leds(4) <= track(4) and (not veto);
.
……...

 leds(21) <= track(1) and (not veto);
 leds(22) <= track(2) and (not veto);
 leds(23) <= track(3) and (not veto);
 leds(24) <= track(4) and (not veto);
 leds(25) <= (track(1) or track(2) or track(3) or track(4)) and (not veto);
```

Each LED is defined as a track and not veto. For example , track 1 would light up LEDs 1, 5, 9, 13, 17, 21 and 25. LED 25 is the central LED and lights up for every track.

Finally the veto is defined as: -

veto <= (track(1) and track(2) and track(3)) or (track(2)and track(3)and track(4))or (track(1)and track(3)and track(4)) or (track(1)and track(2)and track(4));

The pulse stretcher

The LEDs in the present code light up for the duration that a track is defined. This is no more than a few clock cycles, i.e. in the nanosecond order of magnitude. This is far too quick for any human eye to detect. The pulses therefore need to be stretched to allow the LEDs to stay on for a period of time that can be detected by the eye. The period that was decided was half a second.

To do this the code looks for track signals and keeps them on for half a second. Looking for track signals rather than single scintillator hits means that no noise will be mistaken for a signal.

Extensions to the code

Extra Directions

With all direction logic, there are 18 possible four way coincidence directions that can be detected: -

(1,9,13,5)	(8,16,14,6)	(2,10,12,4)	(7,15,13,5)	(3,11,13,5)
(1,9,14,6)	(8,16,13,12)	(2,10,13,5)	(7,15,12,4)	(3,11,14,6)
(1,9,12,4)	(8,16,12,4)	(2,10,14,6)	(7,15,11,3)	
(1,9,15,7)	(8,16,11,3)	(2,10,15,7)		
(1,9,11,3)	(8,16,10,2)			

As for the four way logic, this can split into two and three way coincidences. Hence each coincidence above has four possible 3way coincidences, and 4 possible 2way coincidences. This results in 180 possible coincidences.

The code already defined can be extended to incorporate this logic by defining each of the new coincidences, eg tracks 5-20, for each button and adding the logic to the LED outputs.

Extra veto

The veto used in the present code could be improved by allowing for the possibility of the number of channels that go off simultaneously to be changed. To do this a for loop would have to introduced that loops through all the hit channels and counts them. Then if the number is above the defined limit a veto is activated.

Improved Counter

The counter could be improved by allowing each of the four counters to always be counting even if the corresponding button is not pressed. Then when the button is

pushed the number is displayed on the screen. This would allow the numbers to be cycled through and compared, rather than having to start the counter from zero each time the direction is changed. Again this could be done with a for loop, which loops each button and counts every track then carries on counting.

7.5 ModelSim

One of the major advantages of programming the logic on a computer is that a testbench can be created. A testbench models the inputs and outputs on the FPGA board. The testbench used in this project included modeling of scintillators, the clock, the tracks, the LEDs, the button, and the counter. By using the coincidence unit code that is to be downloaded onto the chip, ModelSim loads the code and acts as it would do on the chip. Then using the testbench the scintillators can be hit at the desired time and all the outputs and inputs seen on the screen. An important aspect of the testbench is that the code is sequentially executed, rather than all at the same time as in the coincidence unit code. The method of defining hits at set times was used rather than defining a random hit generator, as it allowed every track to be rigorously tested for every button. ModelSim allowed every input, output and internal signal that was defined in the code to be displayed in a waveform on the computer screen.

Three testbenches were created: one to test 4 way coincidences, one to test 3way coincidences, and one to test 2 way coincidences. The code for these testbenches can be seen in Appendix X.

For each testbench, all the buttons were tested. For every button, all the coincidences, the counters, random hits, and the veto were tested. All of the results of these can be seen in Appendix XI.

As an example testbench, three way coincidence with all counting will be used. The output waveform for this can be seen in fig 84, overleaf. Although the display is compressed to fit on the page, and the text hard to read, the important detail can be seen clearly. The features in fig 84 are : -

1 – represents the scintillators 1- 16
2 – represents the internal tracks being defined
3 - represents the LED's 1-25
4 – represents the counter (counts in binary in the test bench)

As before, everything that is to be used in the simulation must be defined in a similar way to the coincidence logic. Every variable that is redefined in the test must be the same as it is in the code that is being tested.

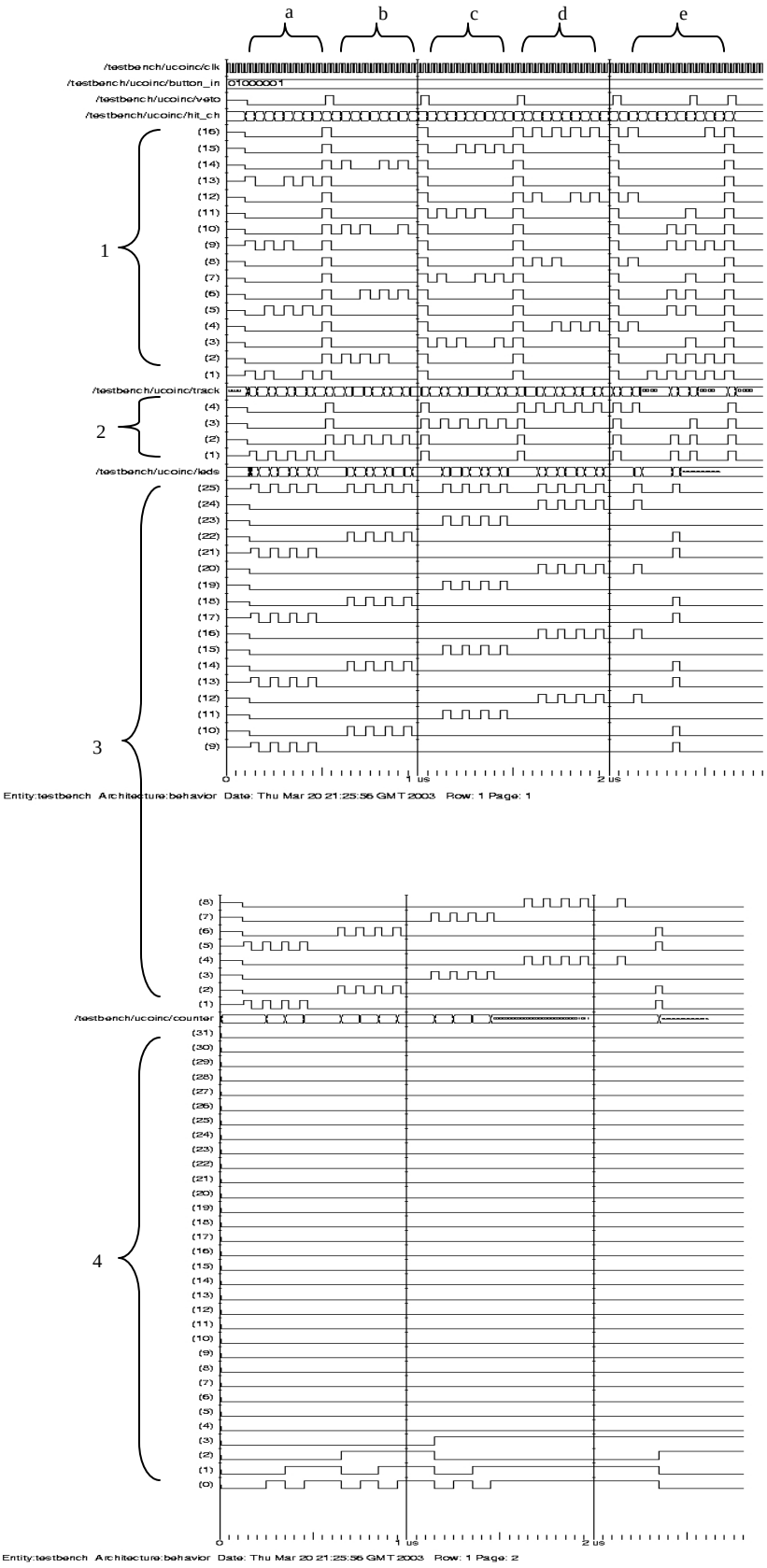

Fig 84 – Testbench output display

```
architecture behavior of testbench is

 component coincide
  port(
   clk_in    : in  std_logic;
   rst_in    : in  std_logic;
   hit_ch_in : in  std_logic_vector(16 downto 1);
   button_in : in  std_logic_vector(8 downto 1);
   leds_out  : out std_logic_vector(25 downto 1);
   veto_out  : out std_logic
   );

 end component;

 signal clk    : std_logic := '0';
 signal rst    : std_logic := '0';
 signal hit_ch : std_logic_vector(16 downto 1);
 signal leds   : std_logic_vector(25 downto 1);
 signal veto   : std_logic;
 signal button : std_logic_vector(8 downto 1);

begin

 Ucoinc : coincide
  port map(
   clk_in    => clk,
   rst_in    => rst,
   hit_ch_in => hit_ch,
   button_in => button,
   leds_out  => leds,
   veto_out  => veto
  );
```

Here the clock is defined as it will be on the FPGA board. The clock defines the period of every test. It is only defined as 10ns as it is on for 10ns and off for 10ns, hence a 20ns period.

```
 -- clock generator
 clk <= not clk after 10 ns;
```

Here the button is defined. This was changed for every test. As the tests had to be thorough to test for any possible errors in the code, every button had to be tested and every possible coincidence for every button.

```
  button(8 downto 1) <= "01000001";
```

Here the code starts. With the reset being set to one to simulate the reset button being preesed, waiting for 50 ns and the reset button being turned off. Veto is also defined as zero.

```
process
begin

  rst <= '1';
  wait for 50 ns;
  rst <= '0';
  wait for 50 ns;

  veto <= '0';
```

This is the code which simulates hit chanels being activated. Here scintillators 1, 9, and 13 are ‘hit’. This can be seen in section 1a of fig.84, where the logic pulses can be clearly seen. In section 2a of fig 84, it can seen that these three hits represent track 1, and in section 3a LEDs 1, 5, 9, 13, 17, 21, 25. The counter has not incremented as it counts rising edges, and the pulse goes straight to be defined, and hence no rising edge. In a real situation the scintillators will be at zero rather than undefined.

```
  hit_ch(1) <= '1';
  hit_ch(2) <= '0';
  hit_ch(3) <= '0';
  hit_ch(4) <= '0';
  hit_ch(5)  <= '0';
  hit_ch(6)  <= '0';
  hit_ch(7)  <= '0';
  hit_ch(8)  <= '0';
  hit_ch(9)  <= '1';
  hit_ch(10) <= '0';
  hit_ch(11) <= '0';
  hit_ch(12) <= '0';
  hit_ch(13) <= '1';
  hit_ch(14) <= '0';
  hit_ch(15) <= '0';
  hit_ch(16) <= '0';

wait for 50 ns;
```

The wait for 50ns is an important trait of the testbench, where the signals can stay in the postion that they are defined for the desired time. After this time has elapsed the hit channels need to be turned off to represent the muon travelling through the detector. Again the testbench is asked to wait 50ns before it does anything else.

```
  hit_ch(1)  <= '0';
  hit_ch(2)  <= '0';
  hit_ch(3)  <= '0';
  hit_ch(4)  <= '0';
  hit_ch(5)  <= '0';
  hit_ch(6)  <= '0';
  hit_ch(7)  <= '0';
  hit_ch(8)  <= '0';
  hit_ch(9)  <= '0';
```

```
hit_ch(10) <= '0';
hit_ch(11) <= '0';
hit_ch(12) <= '0';
hit_ch(13) <= '0';
hit_ch(14) <= '0';
hit_ch(15) <= '0';
hit_ch(16) <= '0';

wait for 50 ns;
```

Here the next set of coincidences are defined 1, 5, 9

```
hit_ch(1)  <= '1';
hit_ch(2)  <= '0';
hit_ch(3)  <= '0';
hit_ch(4)  <= '0';
hit_ch(5)  <= '1';
hit_ch(6)  <= '0';
hit_ch(7)  <= '0';
hit_ch(8)  <= '0';
hit_ch(9)  <= '1;
hit_ch(10) <= '0';
hit_ch(11) <= '0';
hit_ch(12) <= '0';
hit_ch(13) <= '0';
hit_ch(14) <= '0';
hit_ch(15) <= '0';
hit_ch(16) <= '0';
```

This process is repeated for all coincidences. This is shown in sections a, b, c and d of fig.84, where tracks 1, 2, 3, and 4 are tested receptively. Fig 84 shows the four possible combinations of three scintillators for each track.

Things to note in fig. 84 are the veto between every track, e.g. between sections 1 and 2 it can be seen that every hit channel is activated. The third line down on fig. 84 represents the veto. It can be seen that when every track is activated, the veto is activated, and although all four tracks are defined, no LEDs are lit and the counter does not count.

Section (e) represents the testing of the code for different situations. The first tracks to be activated represent a four-scintillator hit. As expected track 4 is still activated and the corresponding LEDs lit. Next one scintillator is activated. No tracks, LEDs or the counter are activated. This occurred similarly for the test of the next two tracks. Three tracks are then activated and the veto stops any LEDs or counter from activating, as should happen. Finally random scintillators are activated and nothing happens, as was expected.

A final point to note is that the counter does not count track four, as was explained above. This can be seen clearly in section (d).

7.6 Summary of Logic

Using an FPGA board, a coincidence unit was created for the detector by programming the desired logic in VHDL and downloading the programme to the chip. The logic that was programmed allowed for coincidences in four planes, 0, 45, 90, and 135 degrees as demonstrated in fig. 80.

For each of these directions three different types of coincidence were defined: two, three, and four way. This gave a possibility of 180 different coincidences in the unit. Also for each coincidence, five different counters were defined: no counter, counting in 0, 45, and 90 degree planes, and a total hit counter.

To make each of the 2, 3, and 4 way coincidence for each counter easily accessible, fifteen buttons were defined, which are detailed in fig.83. In addition to these fifteen buttons, a sixteenth button was defined as a test for the apparatus. The sixteenth button cycled through each possible track lighting up the corresponding LEDs.

As well as defining the logic performed on the inputs, the code displayed the coincidences on an LED display. The design for the LED display can be seen in fig. 82. In showing the path on the LED display, each of the input pulses had to be stretched for 0.5 seconds.

Noise also needed to be considered. As the detector is an extremely sensitive piece of equipment, it is likely to suffer from random noise. The coincidence logic will handle some of this noise, but an additional piece of code was defined called a veto. This tested if three or more tracks went off at one time. If this happened, the counter would not be incremented and the LEDs would not light.

Before any code was actually downloaded onto the FPGA board, it was tested in a programme called ModelSim. This allowed the code to put in a 'real' situation, where it was receiving a clock, and 'hits' from muons. The programme allowed the output from the coincidence unit to be displayed on screen. Using this, the code was thoroughly checked for errors. The results of these tests can be seen in Appendix XI.

The code was also programmed in a style that is easily extendable to incorporate coincidences in all possible geometries, different styles of counters, and different types of veto.

The coincidence logic was successfully downloaded onto the FPGA board.