C4 DESIGN AND TEST OF FINITE STATE MACHINES

In order to complete this laboratory, you ***must*** *do the preparatory work listed in section 2. This laboratory has two 3 hour sessions. You should try to reach the end of section 5 by the end of the first session. If you are running out of time, you should at least have done sections 4, 5 and 6. Note that there is a lot of circuit construction to be done – see the appendix.*

1. Introduction

All the processes involved in the design and manufacture of any electronic system (digital or otherwise) are subject to error, whether due to human mistakes or manufacturing imperfections. Checking that the design process has been correctly carried out (design verification) and that the product has been correctly built (manufacturing test) are essential parts of the production process. The generation of test patterns for manufacturing test can be difficult even for quite simple circuits, but the difficulties can be minimised by applying the principles of design for testability (DFT). The added DFT structures can assist in the design verification stage as well. This experiment is intended to illustrate some of the aspects of testing and DFT as applied to finite state machines (FSMs). These are of fundamental importance in digital systems, virtually all of which contain at least one FSM.

In the discussions that follow, an important distinction is made between the internal nodes of a circuit and the primary inputs and outputs (PIs and POs). The difference is simply that *the PI/POs of an integrated circuit are available for driving or monitoring, whereas the internal nodes are not.* The difference is important because the basic problem in testing is the access (or lack of it) to the internal nodes of the circuit; the main point of DFT is to provide means for improving this access. The aim of this exercise is to show how DFT structures can be built into an integrated circuit (IC) to assist testing. Because it is impractical to insert faults into an IC, we will simulate the behaviour of ICs using discrete components.

An FSM, as its name implies, is a device that can exist in one of a finite number of states and remain in that state indefinitely after the inputs have been removed. We will consider exclusively synchronous FSMs, in which the state is represented by the values of a set of state variables, stored in flip-flops driven by a common clock. The state is constant throughout each clock cycle, and changes at the next active edge of the clock.

The operation of the FSM is characterised by the transitions between the state before a clock edge (the present state) and the state after the clock edge (the next state). The transitions are controlled by external inputs in accordance with an algorithm defined by the design requirements: this is the origin of the term "algorithmic state machine" (ASM).

A generalised structure for an ASM is shown in Figure 1. The set of transitions can be described in the form of a state transition diagram or table, or (more efficiently) by an ASM chart. The state transitions are determined by the next-state logic, which is a block of combinational logic providing the inputs to the state variable flip-flops. The action of the next-state logic can therefore be described in the form of a set of next state equations, each of which expresses the next value of an individual state variable as a function of the present state (i.e. of all the state variables) and the external inputs.

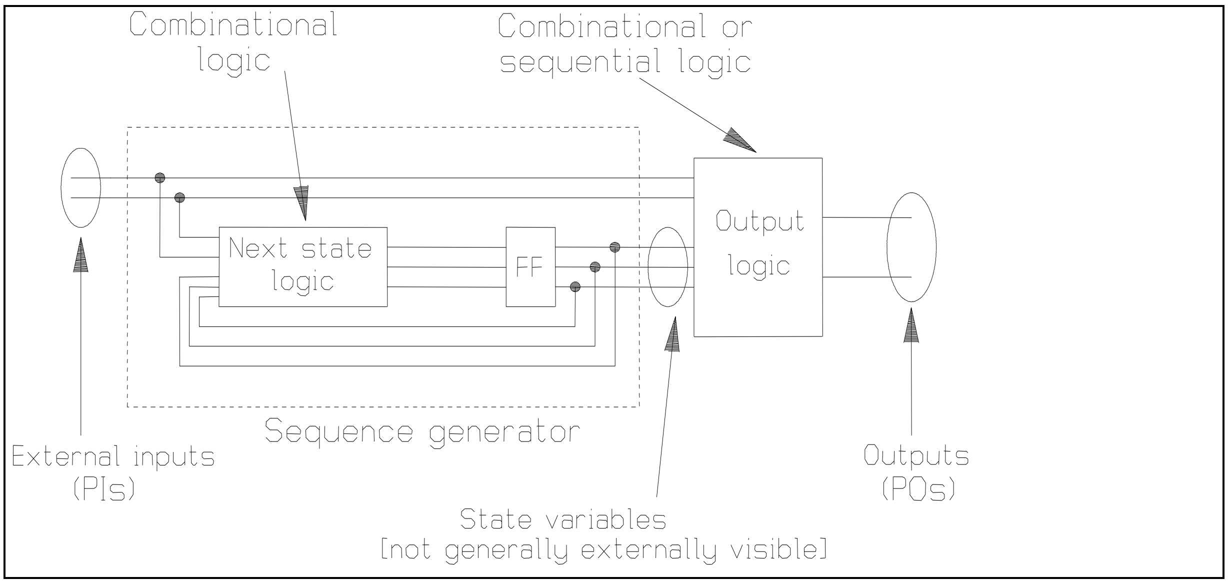

Figure 1

The state variable flip-flops, together with the next-state logic, form a sequence generator. The outputs from the ASM could be simply some or all of the set of state variables; more generally, the outputs are formed as functions of the state variables and the external inputs. These output functions are generated by the output logic, and can be combinational or sequential or both.

This experiment is intended to provide experience in the design and analysis of ASMs, and to introduce some of the concepts of testing and Design For Testability (DFT) as applied to ASMs.

2. Preparation

In order to gain maximum benefit from this experiment it is important to give preliminary consideration to the issues *before* coming to the laboratory.

The following sections of the experiment should be completed before coming to the laboratory.

Section 3	(a), (b), (c), (d), (e)
Section 5	(a)
Section 6	(a), (b)
Section 7	(a), (b)

In addition:

(a) Ensure you understand dependency notation so as to be able to interpret the circuit diagrams. (Also, remember de Morgan's Law: $\overline{A.B} = \bar{A} + \bar{B}$.)

(b) Revise ASM charts (First Year notes and supplementary reading if necessary) to ensure that you know the meanings of the symbols and the corresponding circuit implementation, paying particular attention to the timing implications.

(c) Read about
  - (i) test pattern generation for combinational logic circuits, based on the single-stuck-at fault model;
  - (ii) initialisation of FSMs;
  - (iii) scan design.

(d) Plan what you are going to do. A number of circuits have to be constructed. What ICs will you need? Sketch how they will be arranged on the breadboard(s). Annotate the circuit diagrams with IC numbers and pin numbers – see Appendix.

3. Functional Testing

Functional testing is based on the concept of checking that the circuit performs its intended function. For an FSM, the function of the circuit is encapsulated in the state transitions and the outputs that should be delivered in each state.

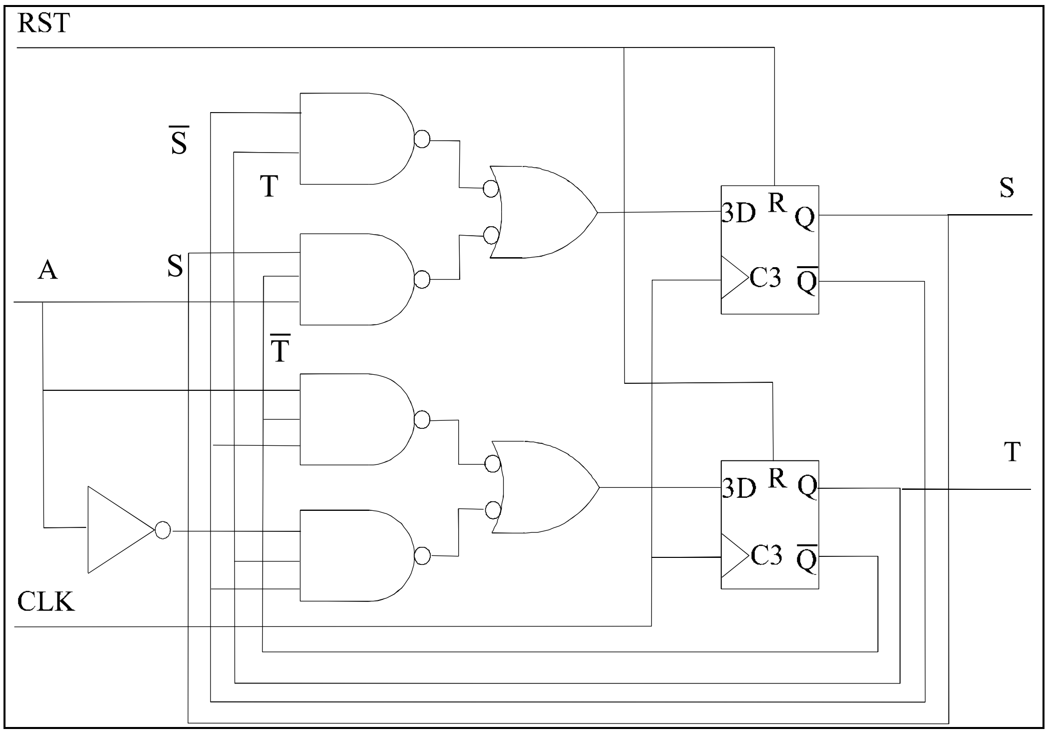

Figure 2

The circuit shown in Figure 2 is an FSM with two flip-flops, and without any output logic. Notice that this circuit has a Reset signal.

(a) Is the reset synchronous or asynchronous?

(b) How would the other sort of reset be symbolised using dependency notation and how would it affect the operation of the reset in terms of timing?

(c) Write down the next-state equations for *S* and *T*.

(d) Draw up an ASM chart describing the operation of the circuit.

(e) Develop a sequence of tests for the circuit, based on verifying each state transition in the ASM. Bear in mind that access is limited to the PIs (including CLK) and the POs. Your sequence **must** be suitable for application by automatic test equipment: i.e. every step from switch-on to decision must be totally unambiguous. The sequence must therefore start with a Reset, since it is not possible to predict what state the circuit will assume at switch on. You can, of course, use Reset again within the sequence wherever convenient.

(f) Build the circuit, using NAND gates for the next-state logic, and verify that your test sequence finds no fault in it. Note that any unused gate input **must not** be left floating: it can be connected to another input or to one of the power supply lines.

4. Fault Detection and Diagnosis

In production testing of Printed Circuit Boards (PCBs), it is necessary not only to establish that a particular circuit is not working correctly (fault detection), but also to identify the location of the fault so that it can be repaired (fault location, or fault diagnosis). This is not generally true of Integrated Circuits, which cannot be repaired.

The first step in this process is to decide what faults might appear in the product, so that the test pattern generation can be directed towards these faults. For many years, the standard *fault-model* has been the single-stuck fault model, in which it is assumed that if there is a fault it will directly affect only a single node of the circuit, and that the fault effect will take the form of the node being stuck at either logic 0 (X/0) or logic 1 (X/1).

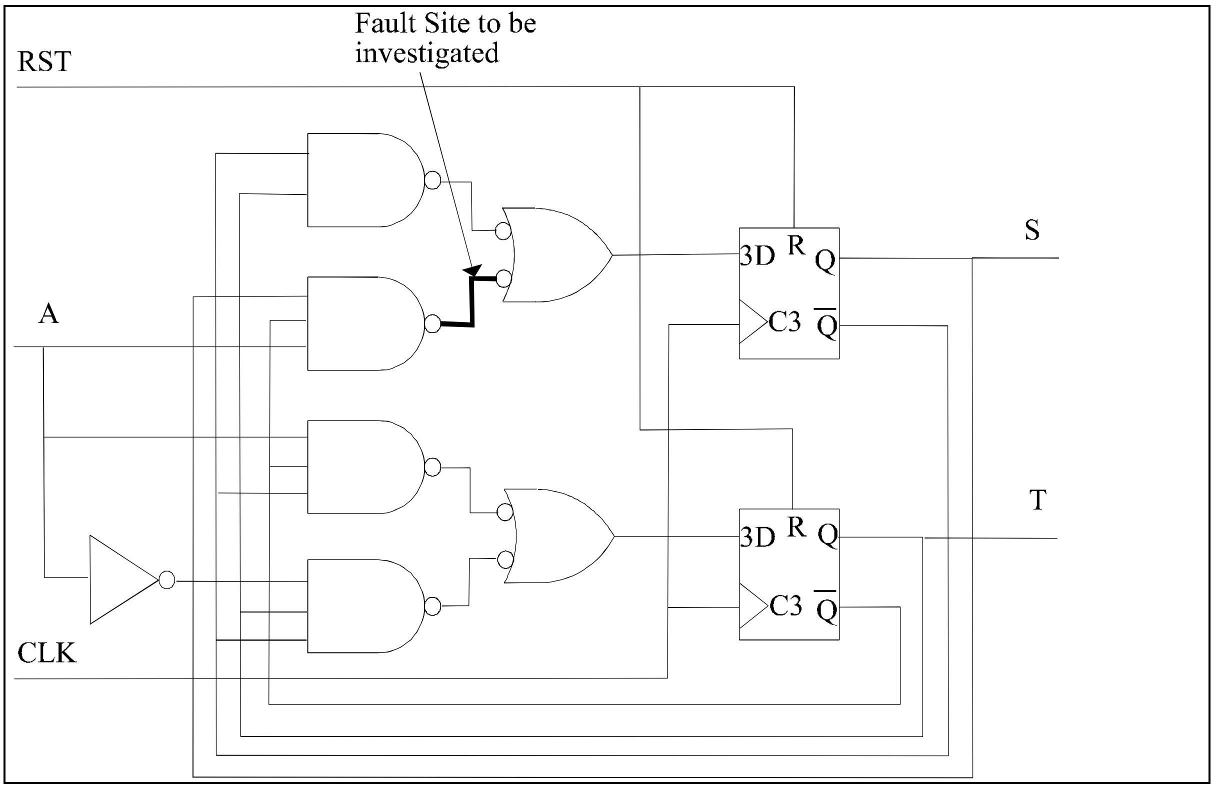

Figure 3

The effects of faults on the test sequence with relation to detection and diagnosis will be demonstrated with respect to the particular fault-site indicated in Figure 3.

To apply a fault, first remove the normal connection between the gates. The input to the second gate can then be grounded, to represent a stuck at 0 fault, or connected to the positive power supply (V_{DD}), to represent a stuck at 1 fault.

(a) Apply a stuck at 1 fault and step through the test sequence. Is the fault detected? If so, can you deduce where the fault is by knowing at which point in the sequence an error was observed?

(b) Apply a stuck at 0 fault, step through the test sequence, and answer the same questions again.

5. Combinational Output Logic

Extend the FSM of Figure 2 by adding the combinational logic block shown in Figure 4 at the outputs. We now have an additional PI (*B*). The POs are now *K* and *L*. *S and T are no longer POs and cannot be directly observed.*

(a) Redraw the ASM chart to include the input, *B*, and the outputs *K* and *L*.

(b) Apply the stuck at 1 fault (as previously defined) and step through the test sequence. [Notice that you will have to change your fault-free output data to reflect the different set of POs.] Does your test sequence detect the fault? If so, is it possible to diagnose where the fault is?

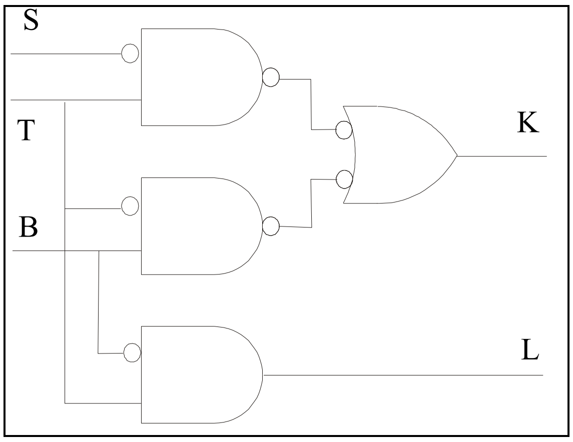

Figure 4

6. Sequential Output Logic

You should aim to reach this point by the start of the second laboratory session.

The output logic of an FSM is not necessarily purely combinational. By designating specific flip-flops to store significant data at one point in the state sequence for use at a later time we can clarify the operation of the sequence generator. More importantly, values stored in output logic flip-flops are not state variables, so that by effectively transferring flip-flops to the output logic the sequence generator is reduced in size (remember that removing one flip-flop halves the number of states).

There is a small price to be paid for this simplification of the design process: since data stored in the output logic will, in general, be required to be maintained over more than one clock cycle, the flip-flop used must be able not only to be loaded with 1 or 0, as in a D-type flip-flop, but also to retain (HOLD) its present value. This can be achieved by providing the flip-flop with an enable input as symbolised in Figure 5(a): *F* and *G* are functions of the state variables and the external inputs. There are various ways in which an enable facility could be implemented.

(i) An enable input can be accommodated in the standard circuitry of a flip-flop without increasing the number of gates. Most register chips have this facility built-in, but discrete flip-flops very rarely do.

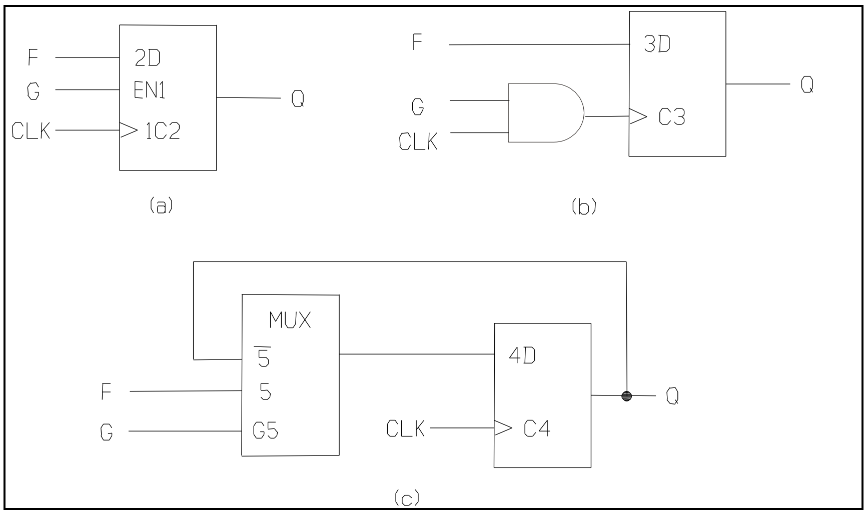

Figure 5

(ii) A D-type flip-flop can be used with the enable used to allow or prevent the CLK from reaching the flip-flop, as shown in Figure 5(b). This is the literal interpretation of Figure 5(a), but **GATING THE CLOCK IS BAD PRACTICE!!** Why? (Consider the flip-flop setup time requirement, bearing in mind that the CLK signal is common to the whole FSM.)

(iii) A D-type flip-flop can be used with a multiplexer as shown in Figure 5(c). This is a clean synchronous design, giving $Q^+ = GF + \overline{G}Q$. When it comes to the detailed circuit design of the output logic, this multiplexer function could be incorporated into the circuitry that generates *F* and *G*.

If the FSM developed in section 4 (Figure 2) is further extended by adding the output circuitry indicated in Figure 6, we have introduced an additional PI, *C*, and PO, *N* instead of the original POs, *T* and *S*[1].

(a) What circuitry, using a standard D-type flip-flop (i.e. *without* an enable input), is required to implement Figure 6?

(b) Add the necessary notations to the original ASM to represent the additional output requirements.[2]

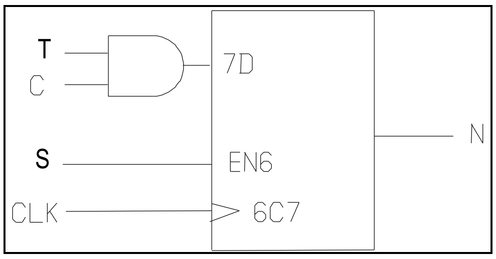

Figure 6

(c) Build the circuit (disconnect the circuitry of Figure 4, but save it for later and add the circuitry of Figure 6), and verify that your previous test sequence finds no fault. (You will need to add the fault-free values of *N* to each of your test vectors.) Is this test sequence an adequate test of the output logic? If not, what vectors need to be added?

(d) Insert the stuck at 1 fault previously defined, and step through your test sequence. Is the fault detected? If so, is it possible to diagnose the fault location?

7. Scan Design.

Testability problems can be mitigated by using the techniques of design for testability (DFT). The testing of FSMs, in particular, is made much simpler by the use of scan design, the principle of which is illustrated in Figure 7.

[1] Remember: if this were an integrated circuit, we could only observe the POs. Here, we also restrict ourselves to only being able to observe the POs.

[2] It's possible that this will not have been covered in lectures by the time of the lab. Consider the following: In which clock cycle does N change? Does N revert to its previous value at the end of that cycle? Therefore, is the ASM notation that you have previously seen, e.g. N=X.Y, appropriate? If not, we need a new symbol: N←X.Y.

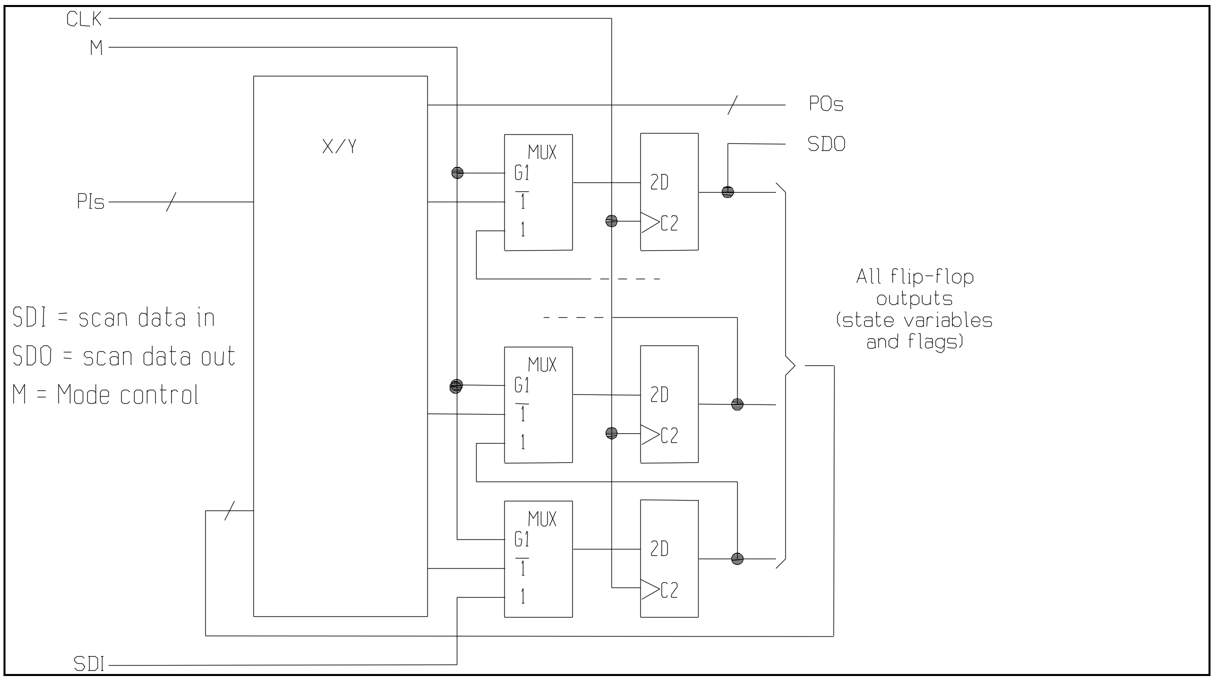

Figure 7

The following points should be noted.

(i) All flip-flops (including any in the output logic as well as those in the FSM) are treated alike. Multiplexers (MUX) are inserted in front of each flip-flop. This allows each flip-flop to take one of two signals.

(ii) The mode input (*M*) is a required additional PI to the circuit. When *M* = 0, the circuit is in its normal operating condition; when *M* = 1, the circuit is in scan mode. Scan mode means that the flip-flops are all connected together to form a shift register or scan path.

(iii) The scan path allows the state of each flip-flop to be set, regardless of the behaviour of the FSM. We simply set *M* = 1 and clock in the values we want. We would then set *M* = 0, to allow the FSM to move to the next state. Similarly, we can read out the state that we're in by setting *M* = 1 again and clocking the flip-flop values out through *SDI*.

Do the following.

(a) Redesign your circuit (complete with all the output circuits added previously) as a scan design. Your total circuit should have PIs *A, B, C, SDI* and *M* (as well as *CLK*), and POs *K, L, N, SDO*. Does it make any difference how the flip-flops are ordered in the scan chain?

(b) Find tests for the stuck at 0 and stuck at 1 faults on the node considered in section 4.

(c) Build the circuit. Insert the stuck at 0 and stuck at 1 faults of (b) in turn, and step through the two tests, noting carefully the sequence of operations needed to observe all the test results for the combinational logic.

(d) Note that this structure, although inserted as an aid for manufacturing test, making it easy to apply structural tests to the combinational logic, it is also a valuable aid to design verification. Confirm this by verifying that, in the presence of the stuck at 1 fault (simulating a missing connection in the design), it is easy to check the state transitions and identify the incorrect logic equation.

8. Discussion of Results

In the light of the results you have obtained in this experiment, you should be able to comment on the following points.

(a) The relative merits of the state transition table (or diagram) and the ASM chart for representing the action of an FSM (particularly one with output logic).

(b) The effectiveness of transition verification as a basis for testing an FSM.

(c) Ways of verifying transitions from redundant states, and the relative costs of making provision for them.

(d) The ease of fault detection and fault diagnosis in non-scan-path FSMs with and without output logic.

(e) The extent to which flip-flops can be exchanged between the sequence generator and the output logic (i.e., can all FSMs be designed without any sequential output logic? Is it possible to design a FSM with all of its flip-flops in the output logic?)

(f) The objections to gating the clock as a design practice.

(g) The benefits of scan design in the design verification phase (i.e., while the design is still only a computer model in a simulator).

Appendix – Wiring up Logic Circuits

This exercise requires several logic circuits to be built on "breadboards". Most people hate wiring up circuits in this way. It is, however, a useful skill and will be needed for the design exercises. Circuits built in this way can be very frustrating to debug, but you can make your task easier and make it easier for a supervisor or demonstrator to help if you plan your task and use some common sense.

- Plan what you are going to do. Sketch the layout of the ICs (in your log book, of course). Annotate the circuit diagrams in the handbook with IC numbers and pin numbers.
- Use different coloured wires. Adopt a convention, e.g. Red for V_{DD}, Black for Ground. Use different colours for different signals. Write the colours on the circuit diagram.
- Keep wire lengths as short as possible. If you change the circuit, it may be better to throw old pieces of wire away and cut new pieces that to try to reuse inappropriate lengths.
- Start wiring up at the start of the lab. You have three hours for each part. If you start wiring up two hours into the lab, you will run out of time.
- The output circuits for parts 5 and 6 can be built on a separate breadboard to that of the main FSM. Therefore you can build the two parts independently and concurrently.
- In case you didn't get the hint about de Morgan's Law in section 2 – a NAND gate can be drawn as an AND gate with an inverted output or as an OR gate with inverted inputs. You don't need extra inverters.