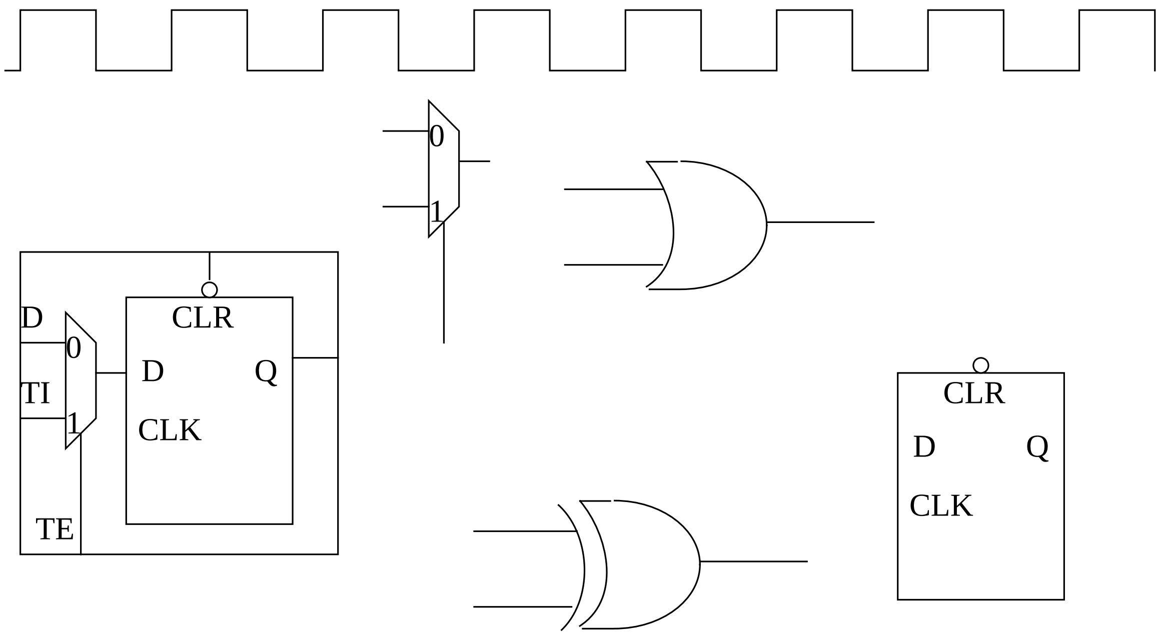
What is DFT, Why DFT, HOW DFT

This document is for professionals who are new to dft or to those students who want to know what DFT is all about.

Comments welcome on yuktimittal@yahoo.com



DFT : by the chip-makers, of the chip-makers, for the chip makers

Abstract:
This document describes how to make an Digital Core DFT-able. The reader, should be able to implement DFT logic on an Digital Core after reading this document. It gives an introduction to what DFT is, and why it is needed, and then describes how to design/implement it. It is assumed that the design is consist of a single digital IP, with and a single Analog block to generate the desired clock frequencies needed by the Digital Core. This document is an answer to several questions which might be lingering in brains of people who may have come across this word DFT, or any of its components e.g scan chains, boundary scan, JTAG, TAP controller BIST etc., and also who haven't heard of any of this terms before. It is intended to help a reader fully understand and implement a practical working DFT technique on a Digital Chip, rather than just writing frustrating theoretical principles, such as found in many text books, which gives us more questions than answers.
The way this document is written is different from conventional writing, in the sense, that first a need is described and then its solution, rather than just start writing without the reader being able to appreciate the need.

Introduction to DFT:
The first question is what is DFT and why do we need it?
A simple answer is DFT is a technique, which facilitates a design to become testable after production. Its the extra logic which we put in the normal design, during the design process, which helps its post-production testing. Post-production testing is necessary because, the process of manufacturing is not 100% error free. There are defects in silicon which contribute towards the errors introduced in the physical device. Of course a chip will not work as per the specifications if there are any errors introduced in the production process. But the question is how to detect that. Since, to run all the functional tests on each of say a million physical devices produced or manufactured, is very time consuming, there was a need to device some method, which can make us believe without running full exhaustive tests on the physical device, that the device has been manufactured correctly. DFT is the answer for that. It is a technique which only detects that a physical is faulty or is not faulty. After the post-production test is done on a device, if it is found faulty, trash it, don't ship to customers, if it is found to be good, ship it to customers. Since it is a production fault, there is assumed to be no cure. So it is just a detection, not even a localization of the fault. That is our intended purpose of DFT. For the end customer, the DFT logic present on the device is a redundant logic.To further justify the need of DFT logic, consider an example where a company needs to provide 1 Million chips to its customer. If there isn't any DFT logic in the chip, and it takes for example, 10 seconds (Its very kind and liberal to take 10 seconds as an example, in fact it can be much larger than that) to test a physical device, then it will take approx. three and a half months just to test the devices before shipping. So the DFT is all about reducing three and a half months to may be three and a half days. Of course practically many testers will be employed to test the chips in parallel to help reduce the test time.
By now one may have justified the need of DFT, but a question lingers, how do we do it? and how does it actually looks like?
How does it works? How to design it? At What stage of normal ASIC design, it starts?

We will take some practical examples to explain how its all done. We will start with very small circuits, understand basic principles, and terminology, then handle complex designs.

Example 1. The purpose of this example is to explain what is functional testing, structural testing, functional simulation, test simulation or fault simulation, fault models, and at what stages of the design these are associated with?

Problem: Design a multiplexer circuit, with the truth table given in Table 1 below, and make the device test-able.

Table 1:

i0	i1	sel	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

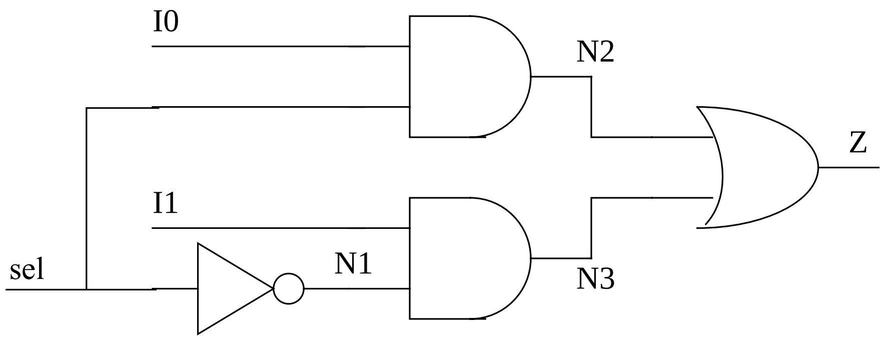


Fig 1. A combinational Logic.

Fig 1 shows a solution of the Problem. And Fig 2 shows a VHDL code for the problem, synthesizing this code will produce circuit equivalent to what is given in Fig 1.
We need to verify it. how do we do that? We give all possible input values to i0, i1, i2 and observe the value of Z, if found according to the expectations, we can say that the design works. A testbench is written for this purpose. The design is then simulated using this testbench. This process is called **functional verification**. Most of us know that.
Before the device is sent to production, we try to think upon what can possibly go wrong during the production process? and how we will detect it, after the device is manufactured.May be a short circuit, may be a open circuit? any thing might be possible, during the fabrication of this multiplexer. There is a need to 'model' the possible faults that might occur during the fabrication process. This is called **fault modelling**. Let us try to device a method which will let us detect, if anything went wrong with the net say N2 shown in the figure, during production. One may argue that the same set of input sequence which was used to verify the functionality of the design during simulation, may be applied to the device after fabrication. Yes, this can be done, Of course the multiplexer will not work as per the truth table in case any fault is introduced in the fabrication process, and the fault will be detected. But that is not what we are going to do. We have a different method to identify the fault. Its a very small design to appreciate the need of what is explained below, but you will be soon able to appreciate it.

If we will be asked to prove that nothing went wrong to the Net say N2, we must be able to
i), Drive N2 to '0' and then observe the value of N2, that it is indeed '0', which will make sure that N2 was not accidently shorted to power net and its not stuck at logic '1'. We may also say that we are trying to evaluate **stuck at '1' fault** at net N2.
ii). Drive N2 to '1' and then observe the value of N2, that it is indeed '1', which will make sure that N2 was not accidently shorted to ground net, and is not stuck at logic '0'. We may also say that we are trying to evaluate **stuck at '0' fault** at net N2.
Stuck at '0' and stuck at '1' are called **fault models**. The ability to drive the net to '0' and '1' is called '**control ability'**, and the ability to observe the effect of controlling is called **'absorb-ability'**.
Now we will write a new testbench which will be able to do the tasks said above, and re-simulate the design using these new testbench.
Let us now try to write such a testbench which will
i). detect a stuck at '1' fault at net say N2. For this
the testbench should put sel = '1', i0 = '0', i1 can be '0' or '1'. This should drive N2 to '0', and Z will have the value of N2. Since it is just a simulation, Z will indeed sampled as '0'. These set of input values when applied after production to the device, will be able to detect a stuck at '0' fault at net N2. The set of input values applied are also called Test-Patterns. Test-Patterns are generated before the production of the device. These Test-Patterns are used in a pre-production simulation called fault-simulation. If the observed output at Z fails to match the expected output for a particular set of input value or pattern, we generally say that this pattern has failed. Note that a test-pattern will never fail during a pre-production simulation, or fault-simulation. These so called test-patterns will be re-used after the production to run what is called production-test.
ii) detect a stuck at '0' fault at net say N2.
We have just tried to detect one of the possible numerous faults that might develop during the fabrication process. In actual, the Test-Patterns should target every possible fault in the design. At times it might not be possible to target every possible fault in the design. The ratio of the faults targeted to the possible number of faults is called fault-coverage.

Fault Simulation: A design needs to be evaluated for how many faults are ‘detectable’ through fault modelling? A simulation can be performed so that an deliberate error is introduced in the design, and then the patterns are run to check, if this deliberately introduced error can be detected by the PFTs. This process is done for each possible fault in the design. This is done to evaluate what is called ‘**fault coverage**’ of the design. This is what is termed as ‘**fault simulation**’. At the end of a fault simulation we get a percentage ratio of what is called ‘**fault-coverage**’ which is = total number of faults detected/total number of faults possible * 100. It is expressed as a percentage.

Notes:

- A single set of input values. i.e a test pattern may be able to detect multiple faults.
- A single fault may be detected by multiple patterns, in that case we have a choice of which pattern to use. For example to detect stuck at '1' fault at net say N2, we may use sel = '1', i0 = '0' and i1 = '0' OR sel = '1', i0 = '0' and i1 = '1'. How do decide which choice to use is beyond the scope of this document.
- In this example we haven't added any extra circuit/logic to make this device DFT-able, it is already DFT-able. In fact we haven't done any design for test, but we just have done some Patterns for Test. or PFT or simply production-test-patterns.

Summary :

- To be able to detect ‘stuck-at’ faults, on any node/net in the design, that node/net must be controllable, and observable using the chip I/O pins.

- Since all the internal nets/nodes in the design in this example were 'controllable' and 'observable', using the chip boundary pins i0, i1, sel and Z there is no need to put extra logic, or DFT, just the PFT will solve our purpose.
- A **fault simulation** is the simulation of the device using PFT, before the production, with the aim to evaluate what is called ‘**fault coverage**’
- Production Testing of the produced device is running those PFT using a tester on every physical device.
- Stuck at '0' and stuck at '1' are called fault models.
- While doing PFT for the design, we have no consideration to what the design's intended function is. This is an important factor, which makes DFT/PFT independent of the intended functional specifications of the device.
- The process of DFT starts once we have netlist for the Design in general.

We now try to extend our approach to sequential circuits as well. Again we will take a small example to show how to do PFT and also DFT this time.

Example 2.
Problem : Make the sequential design in Fig 2(a) Test-able. Also give a complete set of PFT for the fault-simulation which will be able to identify a stuck at '0' fault at net ‘n4’ shown in Fig 2(a)
Here we have not given any intended function of this sequential circuit which has got 3 filp-flops inside it. Since from DFT point of view, the intended functionality does not matters at all.

Given that we already know how to do production-test-patterns for a combinational logic, we try to break the given circuit into two parts, its combinational logic blocks and its flip-flops. The Fig 2(a) shows three combinational blocks which are en-circled. Out of those 3 combinational blocks one has a dark 'bold' circle around it. We shall first try to concentrate upon the combinational logic inside this dark 'bold' circle. This has got 3 inputs i0, i1, i2 and one output co2. Had we got access to i0, i1,i2, we could have easily done PFT for this block. So we shall now try to make i0, i1, i2 control-able via the chip boundary, and co2 observable via the chip boundary. We will do similar kind of arrangements for all inputs and outputs to all the combinational blocks identified in the design.
For this we replace all the flip-flops in the counter, with special flip-flops, which are just identical to the one we had replaced, but now with a multiplexer inside the flip flop, and two more input pins called TE (Test enable) and TI (Test Input) for the flip-flop. As shown in Fig 3(a) The multiplexer can select between TI and D to be clocked and produced to Q when the relevant edge of clock arrives. Now we connect these Flip-flops to from a shift register, as shown by RED color in Fig 3(a). Three more pins are added to the circuit
i). scan_in : it is the serial input for the so-formed shift register,
ii). scan_out: it is the serial output for the so-formed shift register,
iii). scan_enable: it is the control which enables to so-formed shift register.

The patterns for the Sequential Shift Register.
A known data can be shifted inside the shift register using the pin 'scan_in' on the chip boundary, while 'scan_enable' is held '1', shifted out, using the 'scan_out', 'scan_enable' is still '1' and then compared against what was shifted in. This will help us to determine that the shift register or so-called **scan chain** is working fine.
The patterns for the combinational logic.
Using the shift register or scan chain, we can control every node inside our design, and we can also observe the effect of controlling each node inside the design. The control and observation both are now done using the pins present at the chip boundary. We shall now try to generate a set of input values which will be able to detect a stuck at ‘0’ fault at net ‘n4’, shown in the Fig 3(a)

To detect a stuck at ‘0’ fault at net ‘n4’ we need to
Drive ‘n4’ to logic ‘1’, which means drive i0 = 1, i1 = 1 so that we have a control over ‘n4’.
Drive i2 = ‘0’, so that the effect of our control is observable at ‘co2’
To put i0 = 1, we should have ‘n2’ = 1, which is a function of chip pin ‘enable’ and ‘net0’. net0 is a direct output from FF0. So in the shift-register the FF0 should be = ‘1’
To put i1 = 1. Since i1 = net1, which is a direct output from FF1,in the scan-chain, the FF1 should be = ‘1’
To put i3 = 0. Since i3 = net2, which is a direct output from FF2, in the scan-chain, the FF2 should be = ‘0’

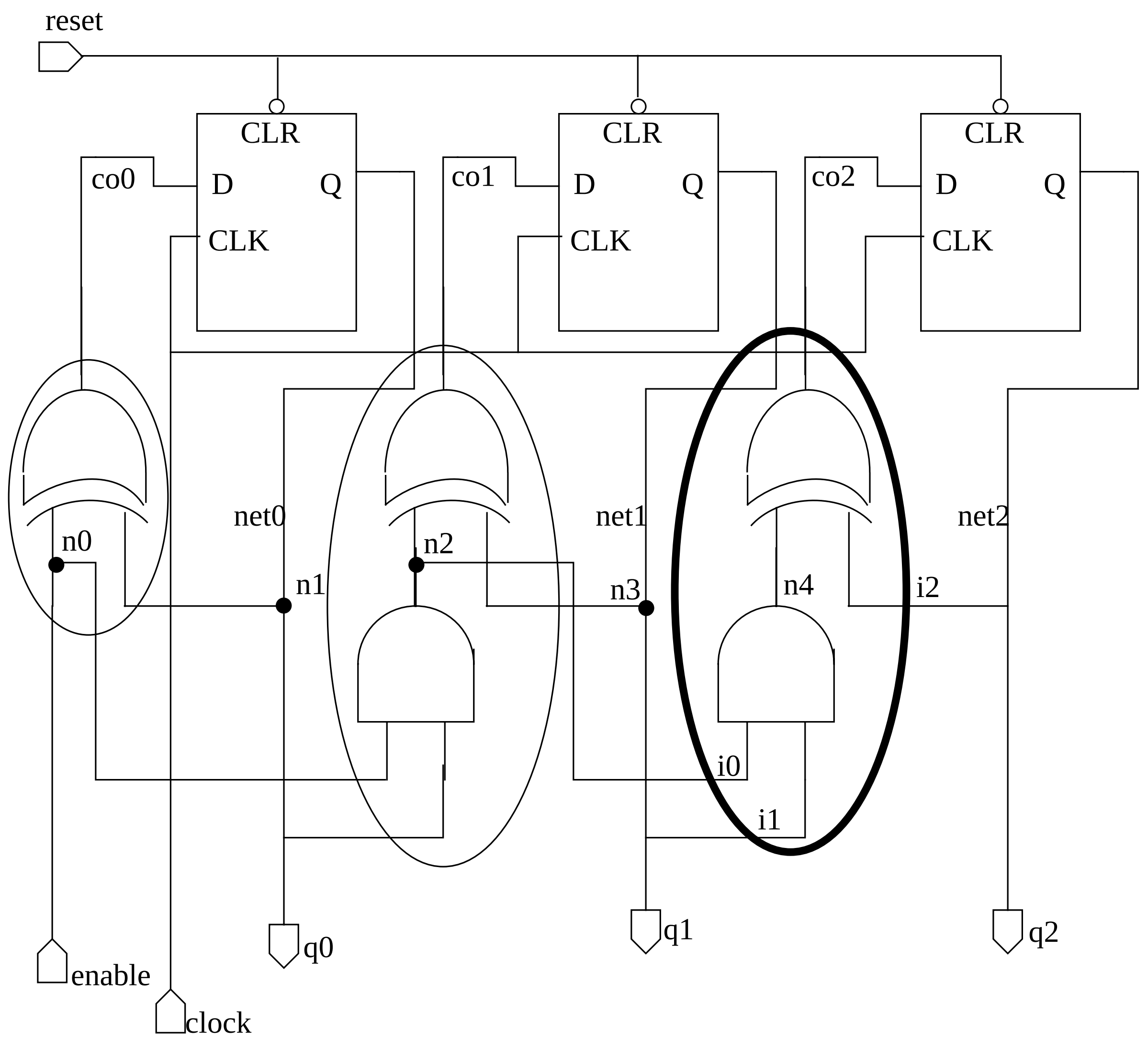


Figure 2 (a): A basic 3 bit counter

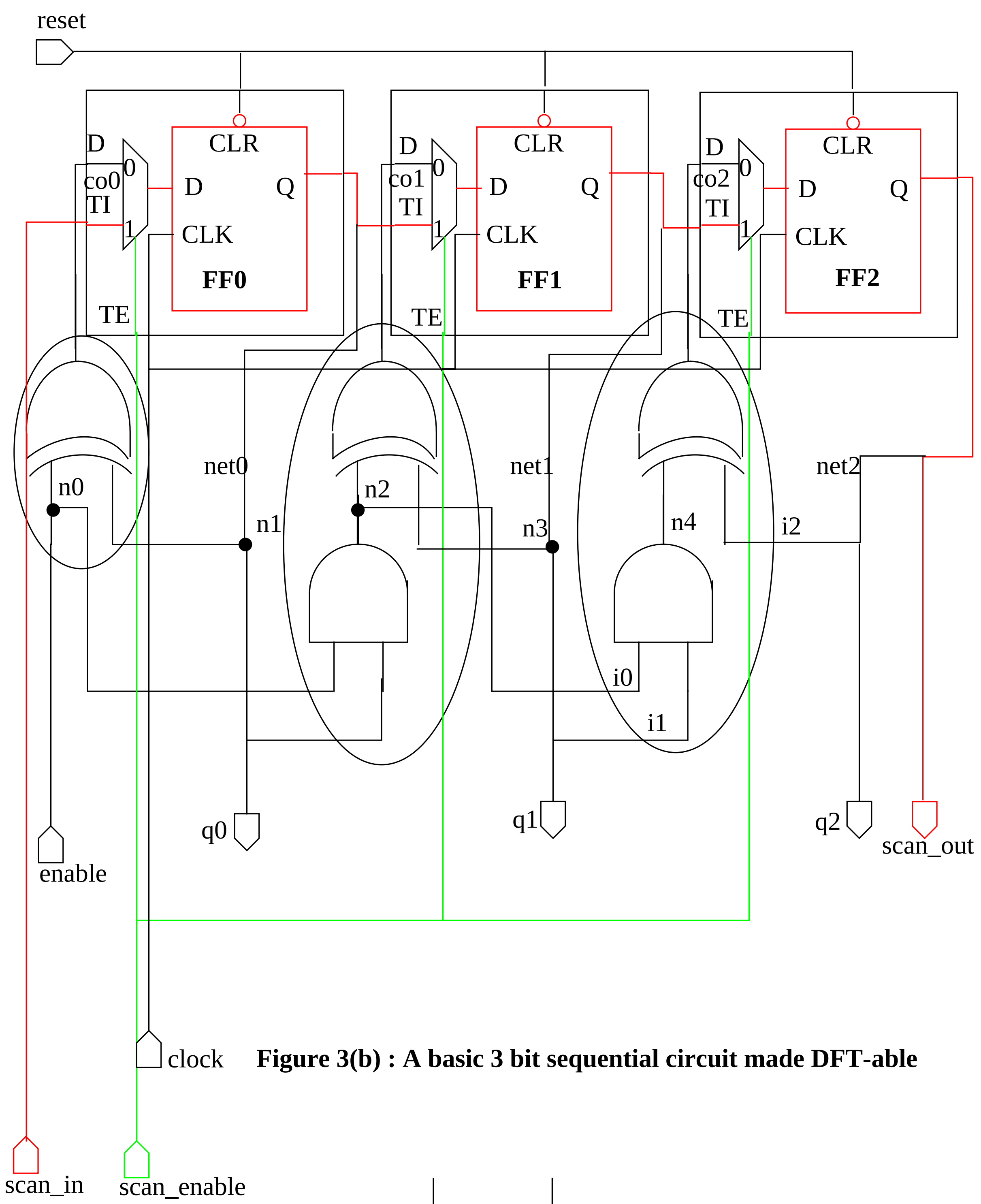


Figure 3(b) : A basic 3 bit sequential circuit made DFT-able

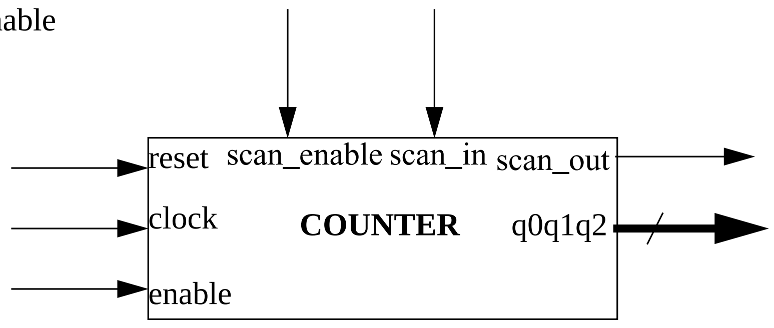


Figure 3(b) : A basic 3 bit DFT-able sequential circuit box

Fig 4. gives a diagram, where we have attempted to generate PFT, which can detect a stuck at ‘0’ fault at the node ‘n4’ as shown in Fig 2(a) and also Fig 3(a). The output q0, which is also the ‘scan_out’ from the design, is drawn in green, and red color. Green means, no fault was reported, as it will be in pre-production fault simulation, or in post-production testing, if there was no stuck at ‘0’ fault detected in the physical device. Red shows an error, and is only possible in post-production test.

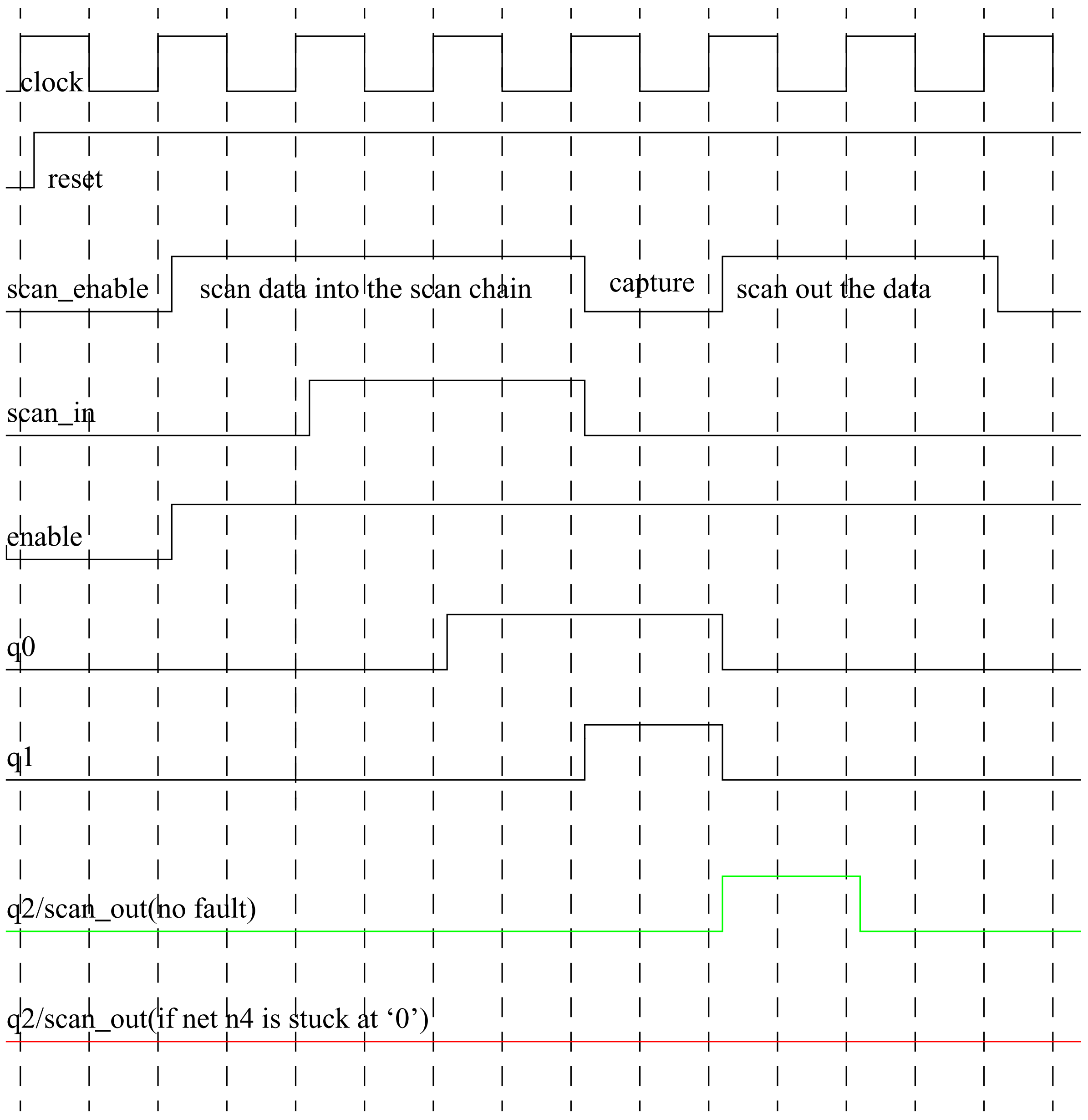


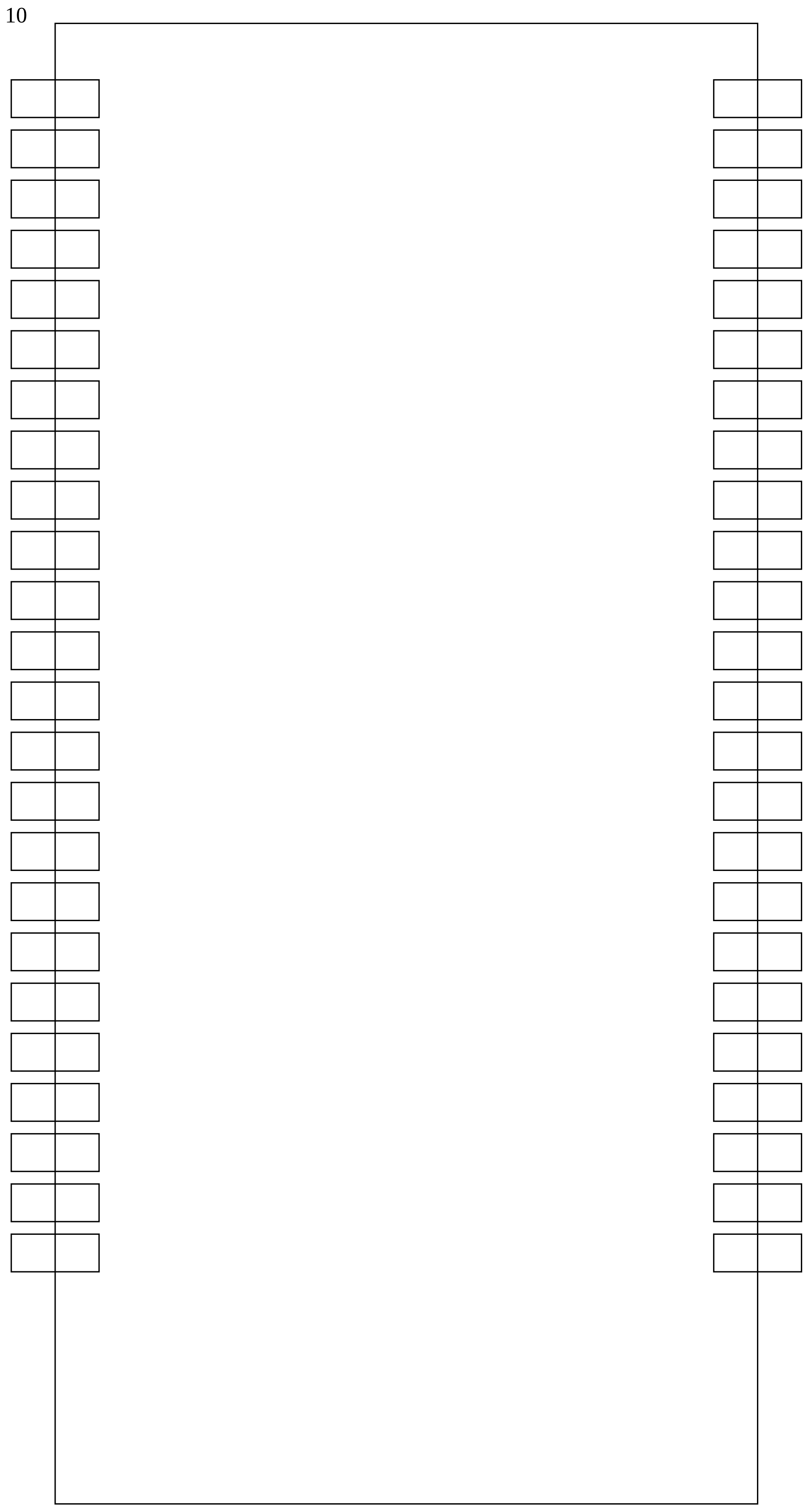
Fig 4: Fault Simulation to evaluate stuck at ‘0’ fault at net ‘n4’ in Fig 2(a)

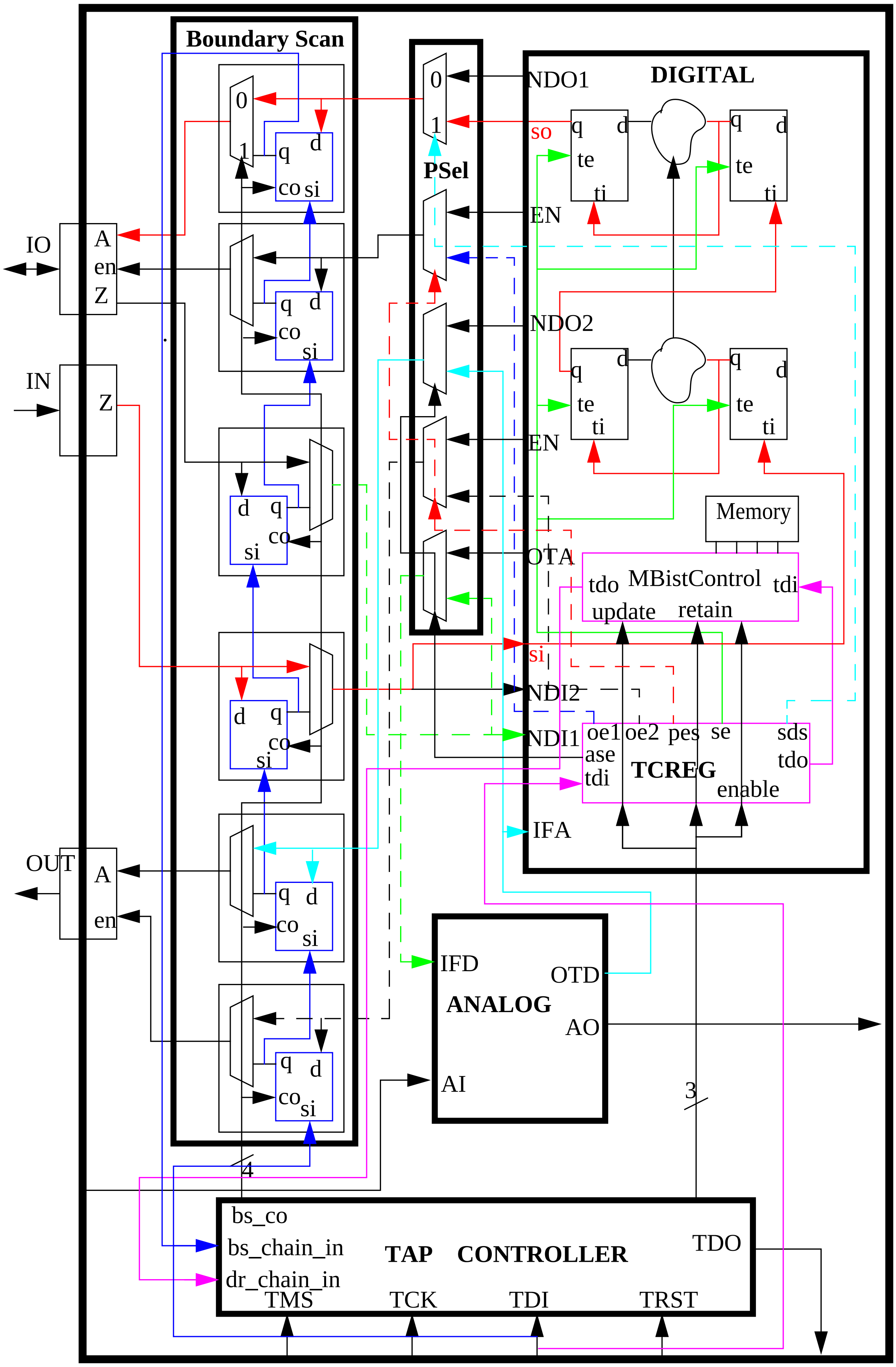
Notes:
- With the introduction of muxes in the filp-flops, we have in fact increased the propagation delay of the flipflop.
- We have also added extra pins in the circuit
- Since q2 was already an output, and ‘scan_out’ = ‘q2’ one may argue that, what is the need to add extra pin on q2. Yes in principles we don’t need to, but just for the sake of understanding it, we did it. And more appreciation will come as we move to complex designs, and a full System on Chip (SoC)
- We have just tried to detect a single stuck at fault in our example, in fact PFTs are done for every possible node/net in the design. Where PFTs are not able to check every possible stuck at fault in the design, it is said that the fault-coverage is not 100%.
- It is the task of the Tools such as ATPG (automatic test pattern generator) to write PFTs. A DFT architect’s work is to modify the circuit, so that ATPG is able to do PFTs and those PFTs can then be used to perform fault simulation as well as post production tests. The aim of this document is also how to do DFT rather than PFT, and the rest of the document will focus entirely on DFT.
- The circuit shown above is in fact a 3 bit binary counter. The end user/customer will always use it as a counter, keeping ‘scan_enable’ permanently tied to ‘0’. So the DFT circuit is redundant for the end user/customer.
- This is a very small design, having just 3 filp flops, in reality, there can be thousands of flip-flops in a design, consequently there can be many scan chains in a single design. We will see more about it in our next example.

Summary:
- The circuit given to us in Fig 2(a) was modified in order to make the internal nodes in the deign ‘controllable’ and ‘observable’, so that we can now perform fault-simulation, before production of the device, then save those sets of PFT to be used by a Tester, on physical device after production. We can now say that we have added DFT circuit to our original design, to make it DFT-able. The resulting DFT-able of Fig2(a) is Fig 3(a).
- This method of DFT will only be able to detect stuck at faults in the design.
- Any sequential design may be made DFT-able to detect the stuck-at faults in the design, by the method shown above.
- The process of DFT is independent of the functional specification of the circuit

Now that we are familiar with DFT terms, and basic principles, we move on. The next example we take will be a complex Digital Chip.

Example 3:
Fig 5 shows a Chip, with a digital block,Make this Chip DFT-able.


10


Boundary Scan
PSel
DIGITAL
NDO1
so
EN
NDO2
EN
OTA
si
NDI2
NDI1
IFA
Memory
tdo MBistControl tdi
update retain
oe1 oe2 pes se sds
ase
tdo
tdi
TCREG
enable
IO
A
en
Z
IN
Z
OUT
A
en
IFD
OTD
ANALOG
AO
AI
3
4
bs_co
bs_chain_in
dr_chain_in
TAP CONTROLLER
TDO
TMS
TCK
TDI
TRST

TMS = Test Mode Select
TCK = Test Clock
TRST = Test Reset
TDI = Test Data In
TDO = Test Data Out

IFD = In From Digital
OTD = Output To Digital
AI = Analog In
AO = Analog Out

OTA = Out To Analog
IFA = In From Analog
NDO = Normal Data Out
NDI = Normal Data In
TReg = Test Register

oe1 = Output Enable1
oe2 = Output Enable2
pes = Pad Enable Select
ase = Analog Select Enable
se = Scan Enable
sds = Scan Data Select
si = Scna Input
so = Scan Output

te = Test Input
ti = Test Enable

si = Scna Input
co = Contorl
bs_co = Boundary Scan Control
dr_chain_out = Data Reg chain out

Psel = Path Selector.