# FPGA 1 – The Clock

Version 1.0

**Heinrich du Toit**

**31 March 2011**

## Contents

## Details

| | |
|---|---|
| Title | FPGA 1 - The clock |
| Release date | 31 March 2011 |
| Author | Heinrich du Toit |
| Current version | 1.0 |

## 1 Introduction

This is my first article about FPGA and DSP design and implementation. In my few years working as an electronics engineer the majority of my time has been writing VHDL code for FPGA designs. Therefore I that feel I'm in the position to write some comments on the topic.

FPGA design is hardly something new in the industry and yet it is a much discussed topic that few engineers truly grasp (and most ignore).

There are many topics around FPGA development I like to discuss but the first article will be about the simple concept of the clock.

## 2 The clock

There is in essence nothing difficult to understand about the clock. Like almost every digital design an FPGA uses a clock, pretty much the same as a microprocessor.

And yet the clock signal is probably the most important signal in the FPGA. It is the signal that drives the processing inside the FPGA as such. Without it nothing will happen.

Firstly it is very important to note that the influence of the clock signal in an FPGA is vastly more complex and intricate than that of a microprocessor. In a microprocessor the hardware "logic" as such has already been fixed and everything has been tested and approved. It is already known what the recommended and maximum clock frequencies of the microprocessor are. After that the microprocessor simply computes the instructions contained in the code as such. The clock is almost trivial to the process.

Inside an FPGA that is not the case. The hardware configuration as such is the code and the clock therefore directly drives the code as such. The complexities and possible problems are far more intricate and therefore a better understanding of how this works is required.

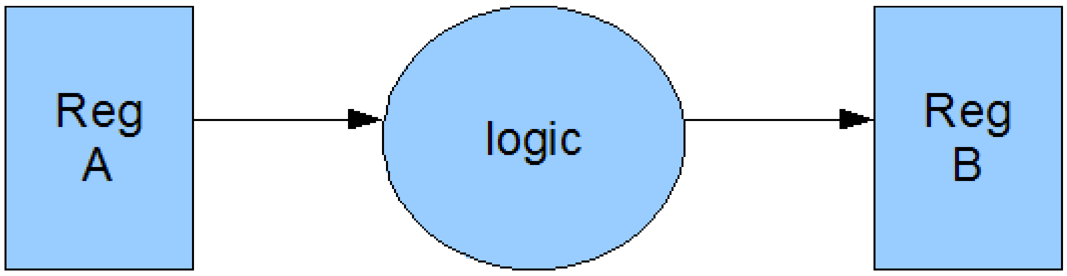
Secondly it must be noted that an FPGA (as we know them at present) is designed to implement synchronous circuits. No other logic/circuit design is really supported or recommended. The FPGA works best with this type of logic and therefore it is perhaps the only recommended logic implementation to be used outside of academic study.

## 3 Synchronous circuit

Defining a synchronous circuit:

- Data inside the system is synchronized to edges of the clock, therefore data (or registers) only updates on clock edges.
- Logic computation happens between these clock edges, while registers values are static
- A single clock drives the whole circuit or domain.

In the following diagram can be seen a simplified view of what a simple synchronous circuit looks like:



The first component of importance is the registers inside an FPGA. Without going into detail of exact implementation, the registers is in its simplest form a memory value that updates its output value to the input value on the rising edge of the clock signal.

Therefore as seen in the diagram the design of synchronous circuits consists of logic sequences between registers that are updated on the clock edges. A complete FPGA design consists of many of these circuits. Each time the clock rising edge occurs the registers will update their output values.

Note that the A and B registers can in fact be the same registers which would then be a feedback design as such. This is very common in for example counters or timers.

There are various things to discuss about this design. For example most of the complexities of FPGA design are about what happens inside the logic section.

### 3.1 Timing and propagation of data

This is perhaps the first and most important thing to understand about the FPGA. Nothing really happens instantaneous.

When the clock edge is triggered the register output is not immediately made the same as the input, this takes a certain amount of time.

And even more importantly the logic computation between the registers is not instantaneous. Depending on various factors it will take a certain amount of time before the wanted value is available at the input to register B in our diagram. Remember that internally digital hardware is actually analogue electronic circuits, therefore voltages takes time to rise and fall and this result in various timing delays in the computation.

And here in lies one of the most important factors of FPGA designs, that of timing closure and efficiency.

If the clock edge happens to occur before the logic section has finished computing the answer then the input to register B would be undefined at best and corrupt at worse. This will off course not have the wanted effect and therefore the FPGA implementation will be broken as such.

Therefore it is of paramount importance that the logic section be designed such that the correct answer is always computed before the next clock edge, and this single requirement can many times be the most difficult part of FPGA development.

## 4 A simplified look into FPGA logic

There are various components inside modern FPGAs and I do not want to go into a detailed discussion about this in the current document.

There are invariable 2 key components to the logic inside an FPGA.

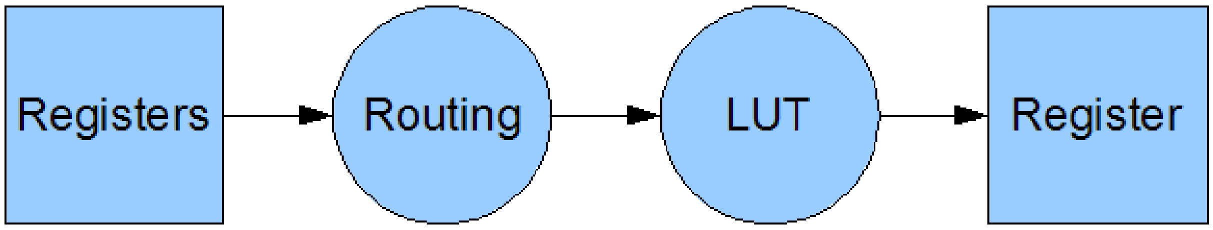
- Look-up table (LUTs)
- Routing logic (The interconnect)

LUTs are simply small memory functions that normally have 4 or 6 bit inputs and 1 output. A 4-bit LUT will therefore have 16-bits of memory and a 6-bit LUT will have 64-bits of memory.

The routing logic is the stuff in the FPGA which is used to connect the different logic parts together. The exact internal layout of an FPGA is somewhat more complex than this and is different depending on the make and model of the FPGA used, but for our discussion this is not important.

### 4.1 A simple design

So the simplest logic setup we can in theory construct inside an FPGA will look as follows:



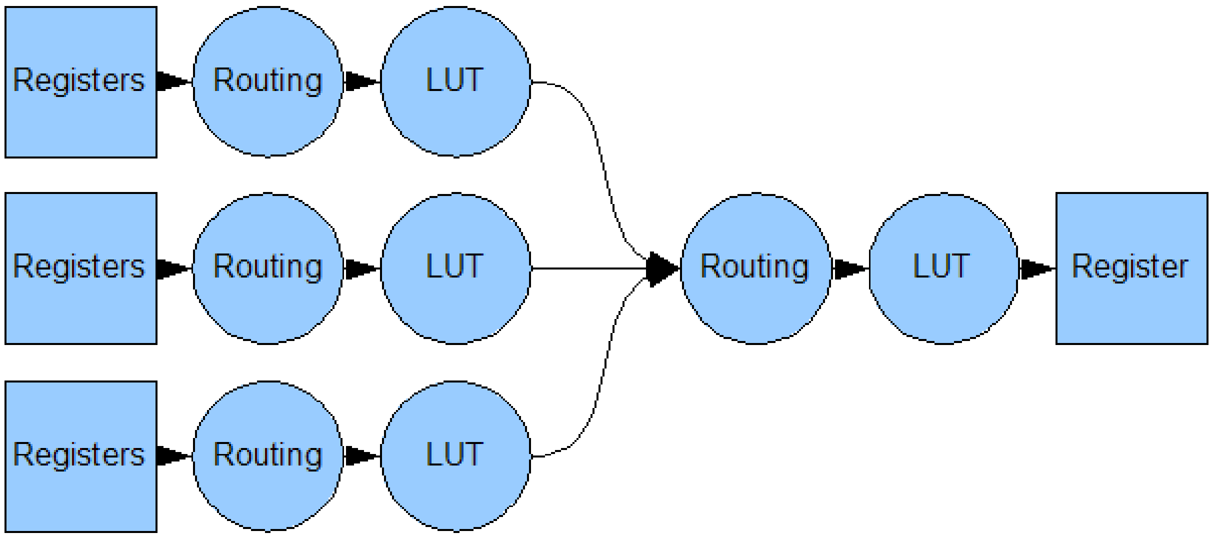
This is a very simple logic setup. The values from the registers is routed to the inputs of a LUT which then computes a new result and this result is once again stored inside one or more registers. (Note that a LUT has only 1 bit output but more than one LUT can be used in parallel)

This is the ideal setup for an FPGA in terms of performance. The distance between the registers is at an absolute minimum and therefore a very high clock speed can be used. But in terms of complexity and functionality this design is extremely limiting.

The reason this is limiting is that the complexity of the calculation is limited by the number of inputs available on a single LUT. Which on lower end FPGA is normally 4 and on higher end FPGAs are 6. One can easily see that more complex functions will simple not be feasible.

For more complex designed more than 1 LUT needs to be used. Or more importantly we need to have LUTs in series.

## 4.2 More complex design



Here a much more complex logical setup has been created putting multiple LUTs in a tree structure to compute a result. The benefit of this setup is that it can take a much larger amount of inputs. Therefore this will enable the setup of much more complex functions.

But one must notice that the data must now propagate through twice the amount of logic and therefore will effectively take twice as long to compute. The effect of this is that the maximum clock speed this circuit can sustain is only about half that of the previous design.

## 4.3 Complexity versus speed

As clearly seen above we have 2 factors which for the most part are at odds with each other. On the one side we have computation speed and on the other complexity of the logic. It is the task of the FPGA programmer to balance this and produce a working solution.

It is clearly seen therefore that choosing an appropriate clock speed for a design is therefore an important design decision. If one chooses a clock speed to high it might become extremely difficult to program the FPGA and obtain timing closure. (Might even require a more expensive FPGA) On the other side if the clock speed is too low it might become very difficult to compute all the necessary calculations fast enough.

All modern FPGAs have clock circuitry inside that enables the FPGA to setup its own clock which is in fact different from the clock signal generated by the hardware outside the FPGA. This way the clock can in fact be changed later in the design without changing the hardware. But in many designs especially those of a DSP nature this can cause difficulties and therefore not something desired later in the design.

## 4.4 Choosing a clock speed

Looking at the datasheet of any FPGA one will see quite impressive clock speed claims.

For example even the lowest speed grade cyclone III FPGA from Altera will indicate a core performance ability of about 400Mhz which is quite amazing if one understands the possibilities inside an FPGA.

But does this mean the FPGA can effectively implement a design at this clock speed? In theory at least the answer is yes.

But in order for a design to actually operate at this potential max speed of the device a number of constraints must be met.

1. All logic must be of the simple nature inside the FPGA. Putting quite a lot of strain on the programmer's ability to optimize things.
2. Mapping and routing of the logic into the FPGA must be quite close to optimal. This puts considerable strain on the software to optimize and compute the actual configuration of the FPGA. It would suffice to say that in more complex designs where the FPGA becomes used to a reasonable amount this timing closure might even be impossible to obtain.

Therefore in my opinion it is probably not a good idea to try and implement a design near the maximum possible operating frequency of the FPGA. My personal recommendation would be to at the very least half the maximum clock frequency of the FPGA unless there is time to put considerable design time into optimization of the FPGA design.

In practice I found that on this particular FPGA a clock speed of around 80MHz actually works great and provides sufficient room for complex logic implementation without producing terrible headaches in optimization.

In order to actually establish a feasible clock speed for any design some experimentation with the design tools is necessary. Using the timing tools available it is possible to obtain a very good idea of what clock speeds might be feasible for the design in the early stages of development.

## 5 Optimization

Unless a very low clock speed is chosen (in which case the choice of using an FPGA can be disputed) at some stage the code inside the FPGA will possibly need some optimization. A good FPGA programmer can and should anticipate possible timing problems and use an optimized design well before problems occur. In any case it is vital to any FPGA programmer to understand a certain amount of basics about implementing optimized designs for an FPGA.

Optimization in an FPGA has to do with 2 things in reality.

The first goal of optimization is in order to achieve timing closure, the stuff primarily discussed in this article.

The second goal is to use as little logic as possible inside the FPGA. This is outside the scope of this article but this is in many designs a very important factor.

A certain part of obtaining closure is to some extent outside the control of the programmer. This is the part the design software will take care of:

- Converting the code into the logic form that can be implemented inside the FPGA
- Mapping and fitting the logic into the FPGA
- Routing the signals between the components inside the FPGA

Some design software allows some control over the fitting and routing processes but this influence will be limited at best.

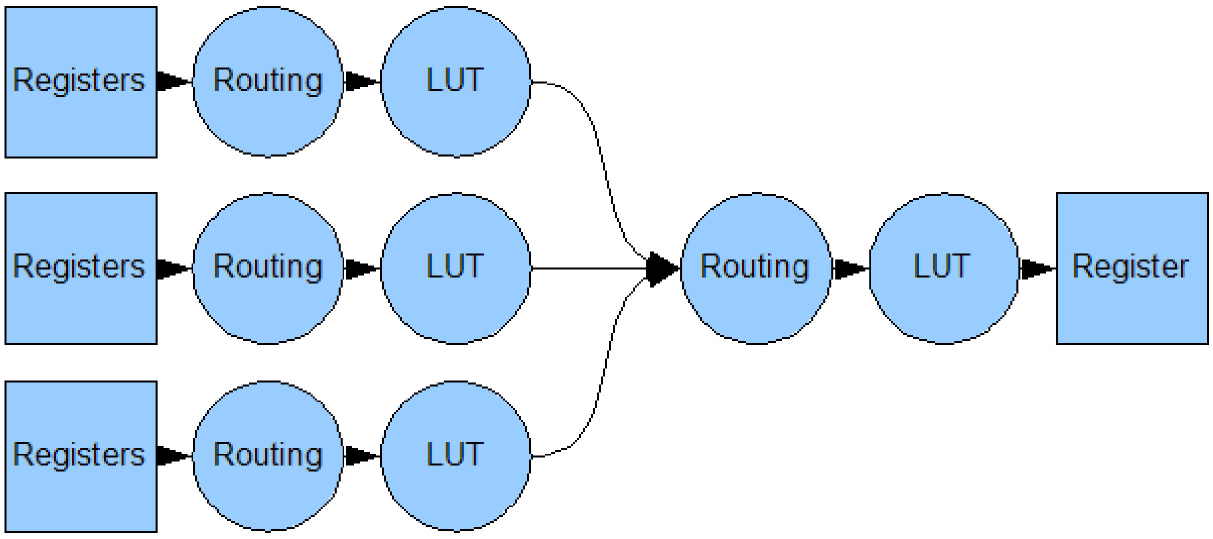
By having a good understanding of how the compiler works the programmer can indeed greatly influence how the software convert the code into logic, but this is a discussion for another time.

## 5.1 Serializing the design

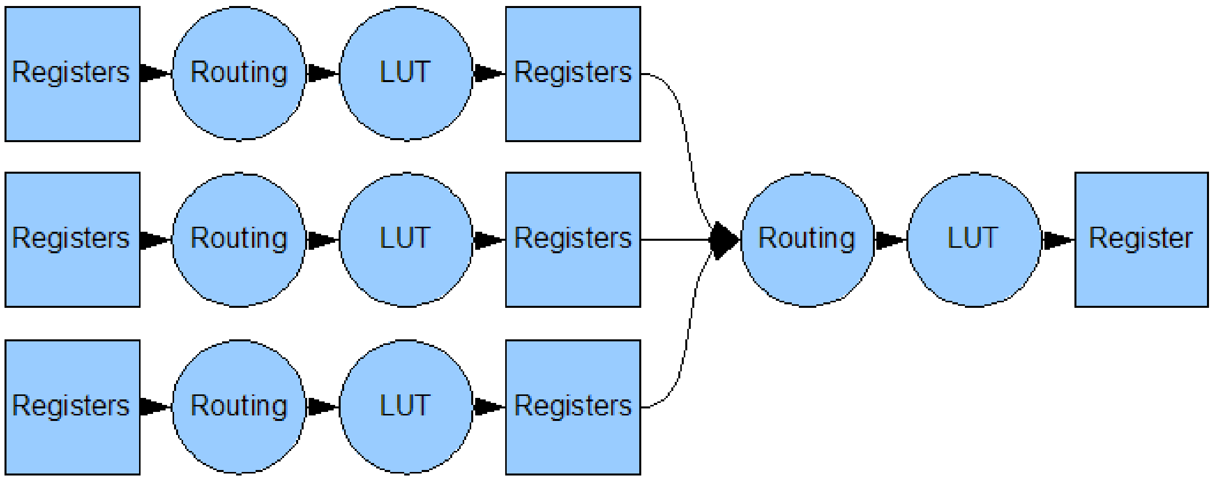
The primary way of optimizing the design for speed is to serialize it.

This basically means that the complex computations is broken up into smaller bits that can be computed one after each other in separate clock cycles.

If we look at the complex design diagram in section 4.2



This can be serialized into 2 stages by inserting registers after the first stage of LUTs.



The result is that it takes 2 clock cycles before the final result is available, but the clock speed can be much higher.

In theory this seems like a rather simple optimization to accomplish but in reality this is not always the case. Although development software provides various tools to help the programmer in this process it can still take a considerable amount of time to find where the optimization is required and then actually modifying the code as needed.

The one possible downside is that this in theory uses more registers but in practice most FPGAs have about an equal amount of LUTs and registers and therefore this is clearly the intended way of programming them.

## 6 Metastability

Our discussion of clocks inside an  FPGA would by fairly incomplete without some discussion about meta stability. There is already a substantial amount of information available on the internet about this topic so this article will not go into too much detail.

For the most part of the FPGA design there are only one clock signal. All registers are updated at the same time and if timing closure has been obtained the input value of the registers will always be valid when the register is clocked.

But in some cases we need to move between boundaries of clocks, or even from unsynchronized data outside into the FPGA. A simple example would be if we sampled a standard serial line. (UART) Or perhaps need to communicate with a microprocessor outside the FPGA.

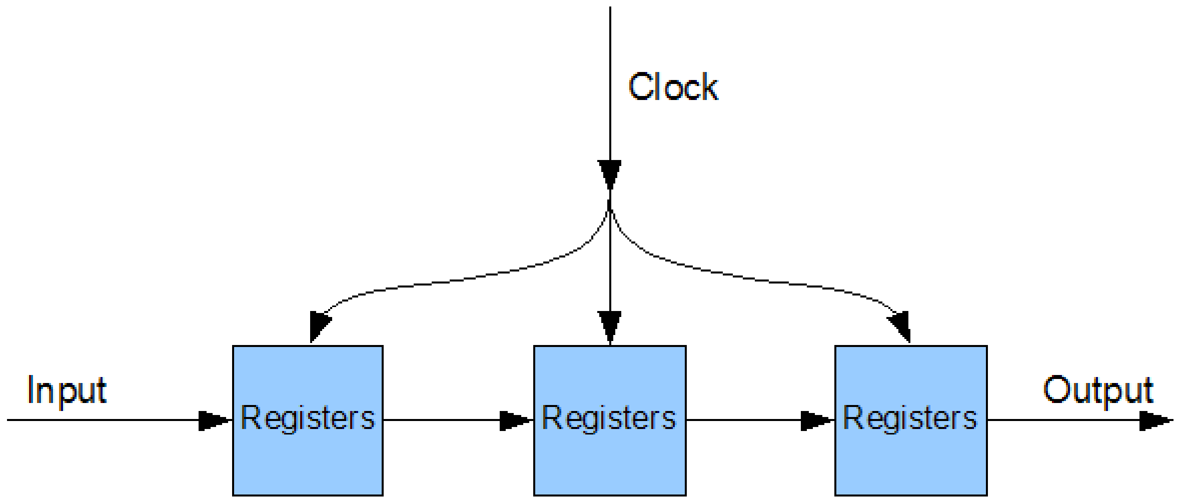
What this means is at the time we clock the data into the register we are not certain of the condition of the input value to the register. In most cases it would be either 1 or 0 and that would not cause any problems. (Note that registers are implemented as D-type flip-flops)

But remember that digital signals are actually analogue signals. So for example a “0” would be 0 volts and a “1” would be 3.3 volts. But what happens if the voltage is busy transitioning and is currently at 1.5 volts?

And here in lies the problem of metastability, which is actually an instability problem. It turns out that if the input voltage to the register is somewhere between the predefined regions for a digital “0” or “1” the output of the register becomes undefined and at worse becomes unstable. This means the output of the register can get stuck between a clear defined “0” or “1” for even more than a clock cycle or even worse start to oscillate. This can cause havoc in the rest of the circuit and produce quite undesired results.

## 6.1 The solution

The solution turns out to be fairly simple.



The solution to metastability is a circuit consisting of a sequence of registers in series that are all driven by the same clock. This is the same clock that will be used to drive the logic following this circuit.

In practice at least 3 registers are recommended. 2 would probably work in most cases and in extreme cases even 4 or 5 can be used.

The statistical probability that an unwanted effect will propagate through all 3 registers is so small that it can practically be ignored and therefore the output of the last register in the sequence can be considered a stable “0” or “1”.

Whenever data is transferred from a different or unknown clock domain this circuit is needed because the possibility exists of sampling the data during the transition between a digital “0” and “1”. It is therefore very important to solve this problem before it occurs as debugging problems related to metastability issues is extremely difficult.

## 7 Conclusion

From our discussions in this article it should be clear that the clock signal plays an important role in the functioning of the FPGA.

It is therefore important that the FPGA programmer have a clear understanding of the complexities involved in this in order to be effective at producing functioning FPGA designs.

This subject is off course in no way exhausted and a lot more can be said about this topic. But that is for another day. It would suffice to say that the understanding of timing closure and optimization inside an FPGA is anything but a triviality.

# 8 Appendices

## 8.1 Sources

Most of this document was written from experience.

- Cyclone 3 handbook – Altera
- Wikipedia articles on metastability