Technical Report

UCAM-CL-TR-115
ISSN 1476-2986

Number 115

Formal validation of an integrated circuit design style

I.S. Dhingra

August 1987

ISSN 1476-2986

Contents

1 Introduction

A specific design style is only ever used if it meets the required needs of the task in hand. The task in hand now is that of generating large, complex, application specific systems on silicon in a fairly short space of time with the confidence that they will perform to the required specification. In the past the development of a large circuit might have been done using a team of engineers over a period of few years, *e.g.* the development of the 68000 microprocessor. This method of circuit development is not acceptable in the present day due to the time and manpower spent in iterating to get the design correct. What is needed is a design technique which is easy to follow and gives very high degree of confidence in the first time correct implementation of the circuit.

Before such a design technique can be developed we need to look at the sort of mistakes that are made which slow down the development of the circuit. The sort of errors that are generally made can be classified into two categories—*timing errors* and *logical errors*. Designers use simulators to model the behaviour of circuits before their implementation. The critical point to note here is that the degree of confidence one has in the design after it has been simulated is no more than the confidence one has in the model of the primitives used in the simulation. The sort of errors that are not easy to catch at the simulation stage are the complex timing errors. This is because the process of manufacturing is not ideal and so there are slight differences from one batch of devices to the next.

So, the sort of design technique needed is a synchronous design scheme. Such a design scheme dictates that all storage elements be updated by the use of a global clock. This reduces the timing problems to the areas between the clocked elements and the problems of identifying difficult timing paths and race hazards is simplified. It also frees the designer to think more about the algorithm and the logical design of the system rather than the detailed timing requirements of the logic.

The price paid for this simplification of the design style is an increase in the size of the chip. This price is probably worth paying since the adoption of a difficult asynchronous design would probably lead to the introduction of obscure timing errors which may be difficult to model and may not be discovered until the chip has been fabricated.

The technology we are interested in is CMOS, and associated with it are hazards not encountered with the simple NMOS technology. For example in CMOS all latches are clocked with a pair of clock lines which are inverses of each other. The constraints on the clock lines are that the overlap between the clocks and the rise and fall times of the clock edges should be less than the delay across a gate (*i.e.* less than a few nano-seconds). This is a stringent requirement for a VLSI technology and may be very difficult to meet as the device geometries get smaller and the capacitive and resistive effects due to the clock lines become more significant. This not only introduces clock stagger across the chip but also degrades the clock rise and fall times—both of which are hazardous and may cause the latches to become transparent for a short time and hence corrupt data. To over come the problem of clock stagger on a large chip requires fine tuning of the clock delays across the chip which would be difficult to model at the design stage and still provide no guarantee as to the probability of it working. To overcome the poor clock rise and fall times requires one to use excessively large clock drivers and clock buffers at regular intervals which leads to rather complex clock structures. Unfortunately it is also desirable not to have very fast clock rise and fall times, since this generates electromagnetic interference and hence causes neighbouring lines to get corrupted, a highly undesirable feature!

The solution to these problems is to do with the design style, not the deployment of clever distributed clock drivers. Presented here is a technique for formally analysing such design styles. Our work is based on a design style known as CLIC [8] which is tolerant to considerable clock overlap and slow clock rise and fall times. It is an extension of the work done by Goncalves and De Man [4] where they present a design style they call NORA that generalises yet another design style known as DOMINO [7].

Common to all of these design styles is the use of dynamic logic rather than static logic. The reason for this comes from the fact that in static CMOS, all logic functions are duplicated—once using the p-type transistors and then again using the n-type transistors. This has the advantage that the static power consumption of the circuit is very low, but it also means that almost twice the amount of silicon area is being used than necessary. However in the context of a synchronous system we have a continuously running global clock which could be used for other purposes: *i.e.* instead of using static logic blocks we use dynamic logic blocks which are clocked by the global clock. This does increase the power consumption a little but it is still less than the power that may be used by a similar NMOS circuit.

This dynamic design feature was used in the DOMINO design style which only used the n-type gates and provided only non-inverting logic. The NORA design style on the other hand used both n-type and p-type devices and so provided the full freedom of inverting and non inverting logic. However a requirement with the NORA design technique was that the clock rise and fall times should be kept fairly small to avoid data corruption by the latchs becoming transparent for a short time. This problem begins to dominate as one approaches VLSI as described above. The CLIC design style overcomes this by use of a two phase clocking scheme.

After a short overview of the CLIC design style and its design rules, we give formal means of developing *correct* CLIC gates. This is based on a simple transistor model with charge storage capability and a simple four value model of the signals on wires. The CLIC design style is fully described in [8]. The notation used there is quite different from ours and the clock labeling scheme is completely different which does not easily lend itself to formal analysis. So a summary of this design technique is first given.

2 Overview of the CLIC Design Style

2.1 Introduction

In this section a brief overview is given of synchronous design methodology and dynamic logic, together with how these two ideas are married together to form the basis of the CLIC design style.

2.1.1 Synchronous Design Methodology

The basic principle behind any synchronous design philosophy is that the system is separated into blocks of purely combinatorial logic with no data storage facility, interleaved by register latches which hold the data between clock pulses. There is a global system clock which is used to clock the register latches, and, the period of this is such as to allow all combinatorial logic blocks to finish evaluation of results. So, on the tick of the clock, new data appears on the inputs of the combinatorial logic blocks, and the old results are passed as inputs to the next stage by use of the register latches. By definition there is no feedback within the purely combinatorial blocks. This principle is illustrated in figure 1.

```
        .-------.       .-------.       .-------.       .-------.
-->--+--| Logic |--->---| Latch |--->---| Logic |--->---| Latch |---+--->
     |  '-------'       '-------'       '-------'       '-------'   |
     |                                                              |
     '--------------------------------------------------------------'
```

Figure 1: Synchronous Logic Concept

2.1.2 Dynamic Logic

A dynamic logic gate has two phases of operation, precharge and evaluate, which is controlled by a system clock. For example a simple pseudo-NMOS dynamic gate is shown in figure 2. The output is precharged high while the `clk` input is low. Then as `clk` goes high the path to `Vdd` is turned off and the path to `Gnd` is turned on. Now the output will either remain floating high or will be discharged low depending on the inputs.

```
              ------+------ Vdd
                    |
                   -'
        .---------0||
        |          -.
        |           |
        |         .-+-+------- output
        |         |   |
        |        -'   '-
        |     ---||   ||---
        |  ip1   -.   .-  ip2
        |         |   |
        |         '-+-'
        |           |
        |          -'
clk ----+----------||
                   -.
                    |
              ------+------ Gnd
```

Figure 2: A simple dynamic logic gate

Many design styles use such dynamic gates. In particular the DOMINO logic design style [7] uses this sort of structures with each output terminated by a static inverter. The advantage of this is that the output of the inverter is low while the gate is in its precharged state. So this can be fed as input to other such gates resulting in a chain of dynamic gates separated by static inverters. Now when the clock goes high, all the gates change to the evaluation phase. Since all inputs are initially low no output of a gate will change state unless the gate at the start of the chain changes. Effectively what we have here is a chain of evaluation going from the start of the chain to the end, much as the fall of one domino causes the next to fall which in turn causes the next to fall and so on.

2.1.3 The CLIC Design Style

This design style uses both dynamic logic features and the principles of synchronous design. It comes with a set of rules which guarantee that there will be no timing hazards. The basic principle behind this is very similar to that of the NORA design style as described in [4]. The major difference is that instead of using the simple clock and its inverse, a two phase non-overlapping clocking scheme is used which results in having four clock lines— ϕ_1, $\overline{\phi}_1$, ϕ_2 and $\overline{\phi}_2$. The remainder of this section tries to give a brief overview of this design style. A more detailed electrical analysis of this is given in [8].

2.2 Clock Definition and Generation for CLIC

The two phase clock for the CLIC design style is generated from a single square wave clock input. On every rising edge of the external clock input an internal narrow pulse is generated on the ϕ_1 clock line. Similarly on the falling edge of the external clock another pulse is generated on the ϕ_2 clock line. These two internal clock lines are then inverted to form the remaining two internal clock lines, namely $\overline{\phi}_1$ and $\overline{\phi}_2$ respectively. These are shown in figure 3 together with the external clock.

```
External             --------          --------          --------
Clock       --------|        |--------|        |--------|        |-----
Input

                     -                 -                 -
Phi1        --------| |---------------| |---------------| |-----------
            --------   ---------------   ---------------   -----------
Phi1'               |_|               |_|               |_|

                              -                 -                 -
Phi2        -----------------| |---------------| |---------------| |---
            -----------------   ---------------   ---------------   ---
Phi2'                        |_|               |_|               |_|
```

Figure 3: The external and the internal clock relationships

For the purposes of analysing the clock scheme we can divide the clock cycle into eight distinct intervals as shown in figure 4. The shaded regions represent uncertainty in the value on the clock lines, *i.e.* the value could be Hi, Lo or something in between. The essential requirements of the clock scheme are that the duration of the clock pulses, *i.e.* the intervals t_3 and t_7, should be long enough so that the internal gates of the chip have enough time to precharge their outputs.

2.3 CLIC Primitive Gates

All the logic gates used in the CLIC environment, except for the static inverter, are dynamic logic gates which are driven by one of the four clock lines. The primitives used in realising a logic function are a combination of these gates. There are four basic building blocks used in CLIC circuits—the N_Shell, the P_Shell, the Latch, and the Stat_Inv. These are illustrated in figure 5. Both the N_Shell and the P_Shell devices need extra components, namely transistors, to be "wired" into the them

```
          /------ One Cycle -----\
          |                      |
           t1 t2 t3 t4 t5 t6 t7 t8
           |  |  |  |  |  |  |  |
           ^  ^  ^  ^  ^  ^  ^  ^
             ---------               ---------
Phi1         |||   |||               |||   |||
          ------   ---------------------   -----------------

          ------   ---------------------   -----------------
Phi1'        |||   |||               |||   |||
             ---------               ---------

                         ---------               ---------
phi2                     |||   |||               |||   |||
          ------------------   ---------------------   ------

          ------------------   ---------------------   ------
phi2'                    |||   |||               |||   |||
                         ---------               ---------
```

Figure 4: Full definition of the internal clock

to make n-type and p-type gates respectively. It is these n-type and p-type gates which form the primitive gates as used in the CLIC design scheme.

In the remainder of this section we give a brief account of how each of these devices work. The working of the static inverter however are trivial and will not be discussed here.

2.3.1 The Latch

The Latch as used in the CLIC environment is a dynamic device. It needs to be updated at regular intervals otherwise the value held on its output may deteriorate. There are two types of latches in the CLIC system, one driven by ϕ_1 and $\overline{\phi}_1$, and the other driven by ϕ_2 and $\overline{\phi}_2$. The differences between the two are only in the clocks they are driven by and none other.

A typical latch is illustrated in figure 6 together with the clocks which drive it. The input is required to be stable during the times when there is a positive pulse on the ϕ line and a negative pulse on the $\overline{\phi}$ line. By stable is meant that the the input should either remain Hi or Lo, but should not be in the process of changing. This constraint on the input is necessary to stop the latch from latching onto a false value.

If a latch is correctly clocked then it can be in one of two modes—Transparent mode or Latched mode.

```
      --------+------
              |
             -'
        .--O||
        |    -.
        |     |
        |     +----- op
        |     |
        |  /-----\
ip  ======>|  N  |
        |  \-----/
        |     |
        |    -'
phi ----+---||
             -.
              |
      --------+------ (a) N_Shell
```

```
       --------+------
               |
              -'
         .--O||
         |    -.
         |     |
         |  /-----\
 ip  ======>|  P  |
         |  \-----/
         |     |
         |     +----- op
         |     |
         |    -'
phi' ----+---||
              -.
               |
       --------+------ (b) P_Shell
```

```
          --------+------
                  |
                 -'
       .-------O||
       |         -.
       |          |
       |         -'
       |    ---O||
       |  phi'   -.
       |          |
ip ----+          +---- op
       |          |
       |         -'
       |    ----||
       |  phi    -.
       |          |
       |         -'
       '--------||
                 -.
                  |
          --------+------    (c) Latch
```

```
         phi'
          |
        |\|
        | \
ip -----|L >O---- op
        | /
        |/|
          |
         phi
```

```
     --------+------
             |
            -'
       .--O||
       |    -.
       |     |
ip ----+     +---- op
       |     |
       |    -'
       '---||
            -.
             |
     --------+------    (d) Static Inverter
```

```
        |\
ip -----| >O---- op
        |/
```

Figure 5: The primitive building blocks of CLIC

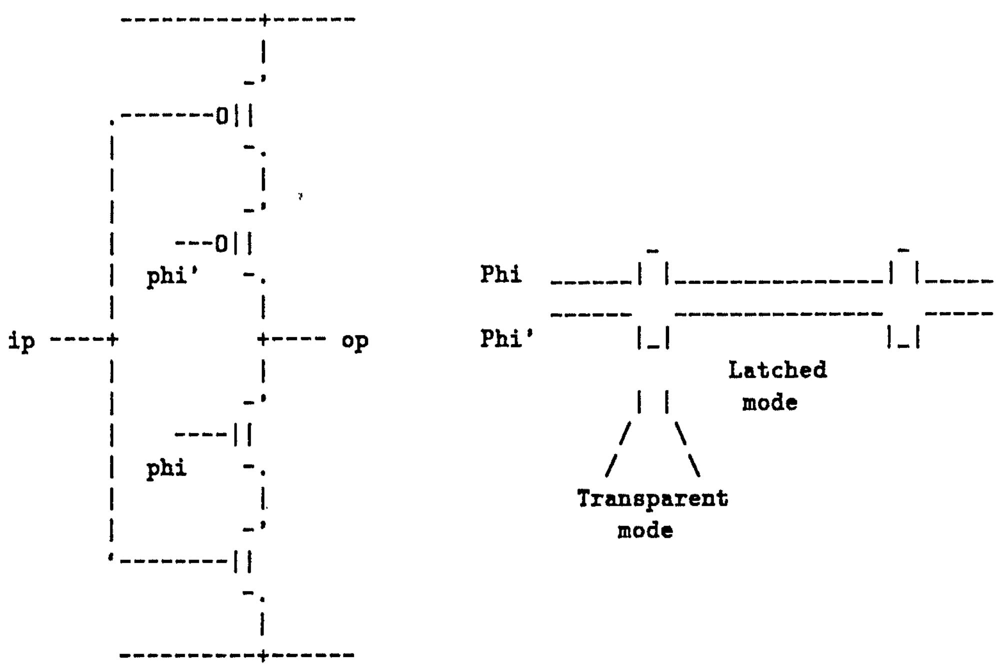

Figure 6: The Latch

Transparent mode ϕ = Hi, and $\overline{\phi}$ = Lo

The two transistors driven by the clock lines get switched on, so the value on the output of the latch is charged to the inverse of the input. It is also important to note that during this time the latch behaves as a static inverter, *i.e.* if there were any changes on the input then they would be reflected by a change on the output.

Latched mode ϕ = Lo, and $\overline{\phi}$ = Hi

The two transistors driven by the clock lines get switched off, so the output node of the latch gets isolated from the power rails and so retains the previously charged value. During this time any changes on the input have no effect on the output. The output node is designed to hold the value long enough until the latch is refreshed again by going into the transparent mode for a short time.

The latch only needs to be in the transparent mode for a short time, long enough for the output node to be charged to the correct value. Even if there is considerable clock overlap and clock edges are poor, the latch will still lock onto the correct value provided the input is stable during the interval when there is a positive pulse on the ϕ line and a negative pulse on the $\overline{\phi}$ line. This restriction ensures that the latch behaves correctly as regards locking onto the input value.

2.3.2 P-type and N-type Logic Gates

The name of p-type and n-type logic gates arises from the fact that these gates use only p-type or n-type transistors respectively except for the precharging and enabling transistors. All p-type gates are driven by either ϕ_1 or ϕ_2 and all n-type gates are driven by either $\overline{\phi}_1$ or $\overline{\phi}_2$. Perhaps the best way to understand the working of these gates is by example.

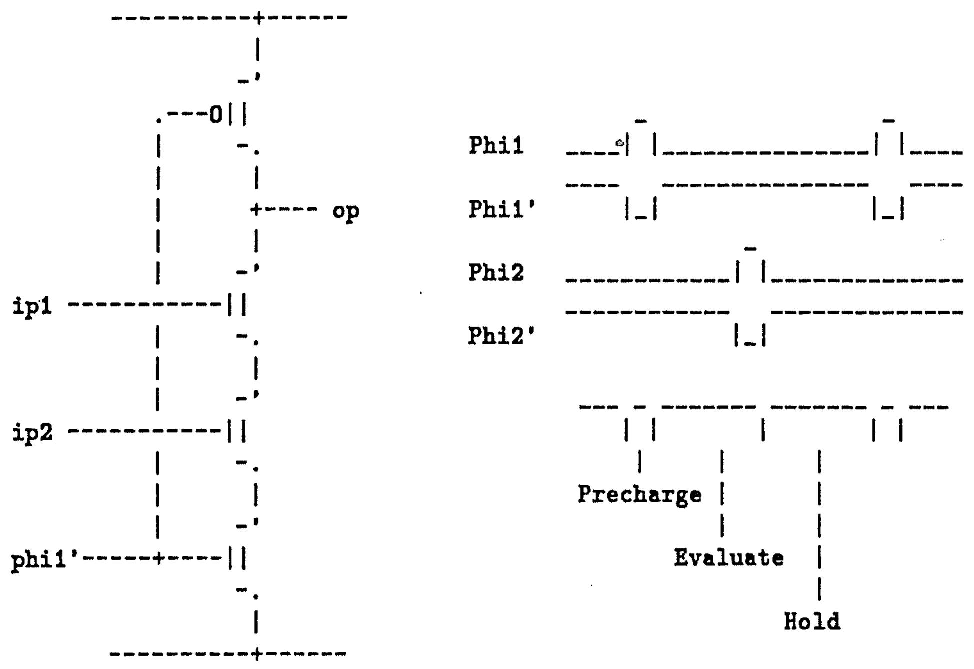

Figure 7: Two input N-type Nand gate

Consider a simple two input n-type Nand gate as shown in figure 7. The working of this gate has two distinct phases—the Precharge period and the Evaluation period.

Precharge $\overline{\phi} =$ Lo

During this period the output of the gate is pulled Hi by the enabled p-transistor. Any changes on the inputs during this time have no effect on the output since the bottom n-transistor is off and so the path to Gnd is effectively cut, *i.e.* the output cannot be pulled Lo.

Evaluation $\overline{\phi} =$ Hi

When $\overline{\phi}$ goes Hi the top p-transistor goes off and the bottom n-transistor comes on. If both the inputs now go Hi then the output will be pulled down to logic level Lo, otherwise it will remain floating at Hi.

The correct answer is generated on the output of the gate at the end of the evaluation period and is held static until the next precharge period.

Note that the output may hold the wrong answer if the inputs are allowed to go Hi and then go Lo during the evaluation period. Since there is no pull-up during the evaluation period, if the output goes Lo then it will remain so until the next precharge period. So for example if the inputs are initially Hi and then go Lo, then at the end of the evaluation period the output and the inputs will all be Lo which is the wrong answer for a Nand gate. To overcome this problem a restriction is imposed which states that "there should be no Hi to Lo transitions on the inputs of n-type gates during the evaluation period."

Similarly the working of p-type gates can be understood by considering a two input p-type Nor gate. The structure of this is shown in figure 8. As in the case of n-type gates, this too has the precharge period and the evaluation period.

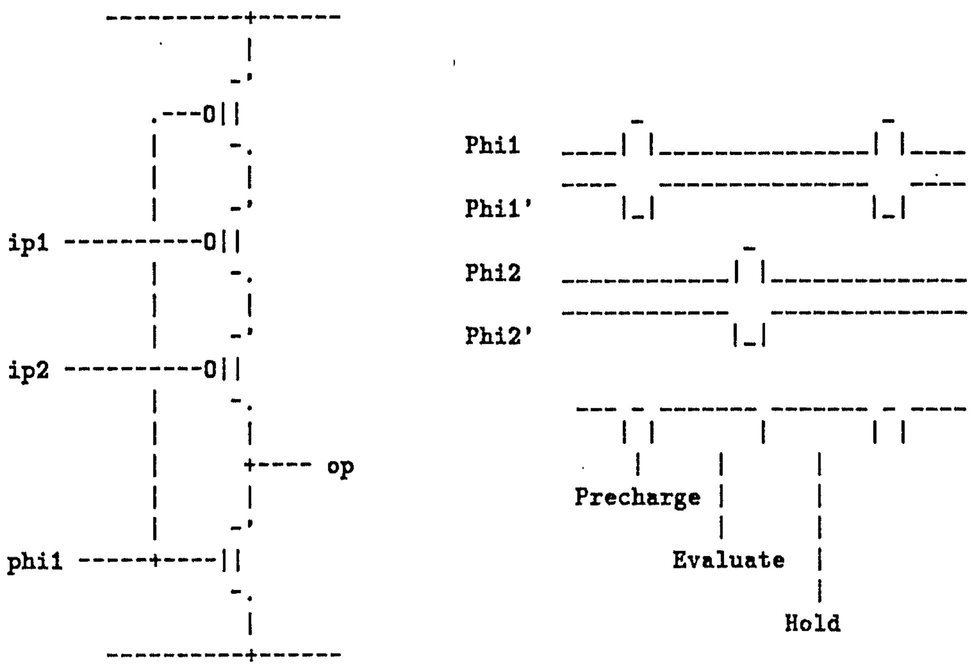

Figure 8: Two input P-type Nor gate

Precharge $\phi =$ Hi

During this period the output of the gate is pulled Lo by the enabled n-transistor. Any changes on the inputs during this time have no effect on the output since the top p-transistor is off and so the path to Vdd is effectively cut, *i.e.* the output cannot be pulled Hi.

Evaluation $\phi = \mathsf{Lo}$

When ϕ goes Lo the bottom n-transistor goes off and the top p-transistor comes on. If both the inputs now go Lo then the output will be pulled up to logic level Hi, otherwise it will remain floating at Lo.

Just as before the correct answer is generated on the output of the gate at the end of the evaluation period and is held static until the next precharge period.

For similar reasons to those of the n-type gates, we need to impose the restriction that "there should be no Lo to Hi transitions on the inputs of p-type gates during the evaluation period."

2.4 Composition Rules for CLIC

From the previous section's work we note that the outputs of the n-type gates cannot have Lo to Hi transitions during the evaluation period which is exactly the requirements on the inputs of the p-type gates. Similarly the outputs of the p-type gates cannot have Hi to Lo transitions during the evaluation period which also meets the exact requirements for the inputs of the n-type gates. This is not by accident but is designed into the style so that we can compose these gates together. Rules like this and others which are necessary to make sure that the circuit designed with CLIC dynamic gates does not contain any timing hazards are presented in this section.

Perhaps the best method of presenting the CLIC composition rules is simply by listing them. Before giving these rules we introduce a few shorthands for the various types of gates and the clocks by which they may be driven.

P_Gate(ϕ) A p-type gate driven by ϕ where ϕ is either ϕ_1 or ϕ_2.

N_Gate($\overline{\phi}$) A n-type gate driven by $\overline{\phi}$ where $\overline{\phi}$ is either $\overline{\phi}_1$ or $\overline{\phi}_2$.

Latch(ϕ,$\overline{\phi}$) A latch driven by ϕ and $\overline{\phi}$ where these clock pairs are either ϕ_1 and $\overline{\phi}_1$ or ϕ_2 and $\overline{\phi}_2$.

If a clock phase ϕ for example is used in the statement of a rule then it is intended that the rule be interpreted as being in two parts—part one with all instances of ϕ and $\overline{\phi}$ replaced by ϕ_1 and $\overline{\phi}_1$ and part two with all instances of ϕ and $\overline{\phi}$ replaced by ϕ_2 and $\overline{\phi}_2$. Having got these we can now give the CLIC composition rules as follows:

Rule 1. An N_Gate($\overline{\phi}$) may be driven by:

(a) P_Gate(ϕ)

(b) N_Gate($\overline{\phi}$) buffered by a static inverter

(c) Latch(ϕ,$\overline{\phi}$)

(d) Latch(ϕ,$\overline{\phi}$) buffered by a static inverter

Rule 2. A P_Gate(ϕ) may be driven by:

(a) N_Gate($\overline{\phi}$)

(b) P_Gate(ϕ) buffered by a static inverter

(c) Latch(ϕ,$\overline{\phi}$)

(d) Latch(ϕ,$\overline{\phi}$) buffered by a static inverter

Rule 3. A Latch(ϕ_1,$\overline{\phi}_1$) may be driven by:

(a) N_Gate($\overline{\phi}_2$)

(b) N_Gate($\overline{\phi}_2$) buffered by a static inverter

(c) P_Gate(ϕ_2)

(d) P_Gate(ϕ_2) buffered by a static inverter

(e) Latch(ϕ_2,$\overline{\phi}_2$)

(f) Latch(ϕ_2,$\overline{\phi}_2$) buffered by a static inverter

Rule 4. A Latch(ϕ_2,$\overline{\phi}_2$) may be driven by:

(a) N_Gate($\overline{\phi}_1$)

(b) N_Gate($\overline{\phi}_1$) buffered by a static inverter

(c) P_Gate(ϕ_1)

(d) P_Gate(ϕ_1) buffered by a static inverter

(e) Latch(ϕ_1,$\overline{\phi}_1$)

(f) Latch(ϕ_1,$\overline{\phi}_1$) buffered by a static inverter

3 Formalising the Clic Design Style

The objective here is to use the various formal techniques available to capture the major concepts which go to make a useful design style. There are a number of level at which we could view the development of the circuit design, from the physics of the semiconductor devices to the top level specifications of a system given in vague terms using English like specification languages. A designer cannot hope to view all his circuit at all of these levels at once. He neither has the capacity nor the expertise in all of these areas, so the best he can do is to work with simple models of the lower level implementations of the various devices. The formalisation of the CLIC design style presented here will thus reflect the sort of devices a logic designer might reasonably be expected to work from. So we begin this section with a very brief introduction of the formalism used.

3.1 Introduction to Higher-Order Logic

The formalism used is that of typed higher-order logic developed by Mike Gordon at the University of Cambridge [5]. This logic uses standard predicate calculus notation. So for example "$\mathsf{P}(x)$" is interpreted as "x has property P." It has the usual logical operators $\sim$, $\wedge$, $\vee$, $\supset$ and $=$ denoting negation, conjunction, disjunction, implication and equivalence respectively. Also provided are the two quantifiers $\forall$ and $\exists$ which express the concepts of *all* and *some*, *e.g.* "$\forall x.\ \mathsf{P}(x)$" means that P holds for every value of x and "$\exists x.\ \mathsf{P}(x)$" means that P holds for some value of x. Finally conditionals of the form "if b then t_1 else t_2" are expressed as "$(b \rightarrow t_1 \mid t_2)$."

What makes this logic higher-order is that quantification is allowed over functions and predicates. So for example the induction axiom for natural number can be expressed as follows:

$$\forall \mathsf{P}.\ \ \mathsf{P}(0) \wedge (\forall n.\, \mathsf{P}(n) \supset \mathsf{P}(n{+}1)) \ \supset\ \forall n.\, \mathsf{P}(n)$$

The use of higher-order logic (HOL) as a hardware description language is explained in [6]. Essentially devices can be expressed as predicates with the labels of external ports of the device being synonymous with the arguments to the predicate. So for example a simple inverter with the input node labeled *ip* and the output node labeled *op* and delay δ would be defined as follows:

$$\mathsf{Invert}(ip, op) \ \ =_{def}\ \ \forall t.\ op(t{+}\delta) = {\sim}ip(t)$$

Where $=_{def}$ is simply a means of saying that the l.h.s. is defined to be equal to the r.h.s.. Joining together of devices in HOL is done by the conjunction of the predicates with the common ports having the same labels. Hiding of internal nodes is done be means of existentially quantifying them. This is explained in more detail in [2].

3.2 Overview

To see how to formalise the CLIC design style we first need to look at the forms of the correctness statements at the top level. For any given device, the correctness statement simply states that "the implementation together with some input-conditions on the inputs implies the specification together with some output-conditions on the outputs." This can be formally stated as follows:

$$\begin{pmatrix} \mathsf{Dev\_Imp}(ip_1, \ldots\ ip_n, op_1, \ldots\ op_m)\ \wedge \\ \mathsf{Ip\_Cond}\ ip_1\ \wedge \cdots \wedge\ \mathsf{Ip\_Cond}\ ip_n \end{pmatrix} \supset \begin{pmatrix} \mathsf{Dev\_Spec}(ip_1, \ldots\ ip_n, op_1, \ldots\ op_m)\ \wedge \\ \mathsf{Op\_Cond}\ op_1\ \wedge \cdots \wedge\ \mathsf{Op\_Cond}\ op_m \end{pmatrix} \tag{1}$$

So the derivation of the correctness statement of a device which is composed of two lower level devices simply requires that the input-conditions and the output-conditions match for all those lines which are to be connected between them. Then by simple logical manipulation we can show that the top level correctness statement can be derived purely from the lower level correctness statements.

This then is the overview of our methodology for the formalisation of a design style. Naturally the Ip_Cond and Op_Cond predicates will have to be defined to reflect the design rules for the particular design style. Also other parameters may need to be added to these predicates to formalise any peculiarities specific to the design style.

However before going on to giving the correctness statements for the various CLIC primitive gates we need a few preliminary axioms and definitions to get us off the ground. Namely we need to clarify the model of transistors we plan to use, the sort of values we plan to propagate around the circuit and reason about and the definitions of the clock.

3.3 Formal Definitions of CMOS Primitives

In formalising the CMOS primitives we need to declare the sort of values nodes in a circuit can have. These values will depend on the models we use for the primitive devices, namely the transistors. After studying the CLIC design style it seems that a unidirectional model for the transistor is adequate for describing the various gates *etc.* So the values we use reflect this choice of the transistor model.

Any node in the circuit can only have one of the following four values:

```
        Er
       /  \
     Hi    Lo
       \  /
        Zz
```

These four values are derived by using Bryants work [1] with only two strengths. Note that they form a complete lattice.

With this we introduce a simple operator $\sqcup$ which states what happens at a node when two signals meet. $\sqcup$ is simply defined to be the least upper bound on the above lattice of values. Now we are ready to define the basic primitives of the CMOS technology within the constraints of the above four valued algebra.

$$\mathsf{Vdd}(x) \quad =_{def} \quad \forall t.\ x(t) = \mathsf{Hi} \tag{2}$$

$$\mathsf{Gnd}(x) \quad =_{def} \quad \forall t.\ x(t) = \mathsf{Lo} \tag{3}$$

$$\mathsf{N\_Tran}(g, i, o) \quad =_{def} \quad \forall t.\ o(t) = ((g\,(t) = \mathsf{Lo}) \rightarrow \mathsf{Zz} \mid i\,(t)) \tag{4}$$

$$\mathsf{P\_Tran}(g, i, o) \quad =_{def} \quad \forall t.\ o(t) = ((g\,(t) = \mathsf{Hi}) \rightarrow \mathsf{Zz} \mid i\,(t)) \tag{5}$$

$$\mathsf{Join}(i_1, i_2, o) \quad =_{def} \quad \forall t.\ o(t) = i_1(t) \sqcup i_2(t) \tag{6}$$

$$\mathsf{Cap}_1(i, o) \quad =_{def} \quad \forall t.\ o(t) = (\sim (i\,(t) = \mathsf{Zz}) \rightarrow i\,(t) \mid i\,(t-1)) \tag{7}$$

Note that all further devices described here will be built out of these six primitive building blocks.

3.4 Formal Definition of Clock

The Clock as described earlier is derived from a single square wave input. The formal definition used is not derived from such a single input but it is simply defined to be that which such a circuit might generate. This is because we do not model the various gates to have delay. It would not be too difficult to derive the given definition as an abstraction of what might be generated by the circuit if we were to use a different model for the gates.

```
Phi1     - - Lo  Hi  Lo  Lo  Lo  Hi  Lo  Lo  Lo  Hi  Lo  Lo - -
                  _               _               _
Phi1     ________| |_____________| |_____________| |___________
         ________   _____________   _____________   ___________
Phi1'            |_|             |_|             |_|
                          _               _               _
Phi2     ________________| |_____________| |_____________| |___
         ________________   _____________   _____________   ___
Phi2'                    |_|             |_|             |_|
```

Figure 9: Graphical representation of the four clock lines

However we are now going to define a predicate Clock with four arguments which are the four lines which might be generated by the clock generator circuit. The way we do this is to define how one of the four clock lines behaves, and then relate the behaviour of the other three lines to it. Before giving the formal definition, here is a graphical representation with an arbitrary starting point.

Note that the clock is cyclic over four units of time and the uncertainty states have been eliminated. This is done to help convey the basic principle of how the bulk of the work regarding the formalisation was done, rather than present even more unnecessary detail than already present. A full treatment to this is given in [3] where the correctness of the design style is shown at the finer grain of time and then related to the coarser grain of time at which this paper deals. Here then is the formal definition of the above graphical representation:

$$\mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) \;=_{def}\; \begin{array}{l} \mathsf{Cycle}\ \phi_1 \quad \wedge \\ \mathsf{Shift}\ \ \phi_1\ \phi_2\ \wedge \\ \mathsf{Invert}\ \phi_1\ \overline{\phi}_1\ \wedge \\ \mathsf{Invert}\ \phi_2\ \overline{\phi}_2 \end{array} \tag{8}$$

Now definitions for the various predicates used to define Clock can be given. The simplest two, namely Shift and Invert are as follows:

$$\mathsf{Shift}\ \phi_1\ \phi_2 \;=_{def}\; \forall t.\ \phi_2(t) = \phi_1(t{+}2) \tag{9}$$

$$\mathsf{Invert}\ \phi\ \overline{\phi} \;=_{def}\; \forall t.\ \overline{\phi}\ t = \mathsf{NOT}(\phi\ t) \tag{10}$$

Where NOT is the negation function over the values Hi, Lo and Er.

So far we have defined the way in which all the clock lines are related to ϕ_1 but have not given a formal definition for Cycle. Before doing this we will state informally the behaviour of ϕ_1.

- ϕ_1 is cyclic over four units of time
- During *any* four consecutive units of time the value on ϕ_1 is Hi exactly once and Lo for the other three units.
- ϕ_1 can start in any of its four possible states.

Now we can formalise each of these three informal statements into logic to give the following definition for clock:

$$\begin{aligned}\mathsf{Cycle}\ \phi \ =_{def}\ & (\forall t.\ \phi(t) = \phi(t{+}4)) \ \wedge \\ & (\ \mathsf{Cycle\_1}\ \phi\ 0 \ \vee \\ & \ \ \mathsf{Cycle\_1}\ \phi\ 1 \ \vee \\ & \ \ \mathsf{Cycle\_1}\ \phi\ 2 \ \vee \\ & \ \ \mathsf{Cycle\_1}\ \phi\ 3\ )\end{aligned} \tag{11}$$

$$\begin{aligned}\mathsf{Cycle\_1}\ \phi\ t_0 \ =_{def}\ & (\phi(t_0) \ \ \ \ \ = \mathsf{Hi}) \ \wedge \\ & (\phi(t_0{+}1) = \mathsf{Lo}) \ \wedge \\ & (\phi(t_0{+}2) = \mathsf{Lo}) \ \wedge \\ & (\phi(t_0{+}3) = \mathsf{Lo})\end{aligned} \tag{12}$$

This seems like a lengthy definition for Clock and it could have been shorter but for two reasons. Firstly that it closely mimics the way the designer has informally described the signals on the clock lines, and secondly it is of the form which allows some of the latter lemmas to be more easily derived.

However here are a few of the more elegant definitions I came up with at the time of thinking how to define the cyclic property of clock. These are all provably equivalent to the definition given above.

$$\begin{aligned}\mathsf{Cycle}\ \phi \ =_{def}\ & (\exists t.\ \phi(t) \ \ \ \ \ = \mathsf{Hi} \ \wedge \\ & \qquad \phi(t{+}1) = \mathsf{Lo} \ \wedge \\ & \qquad \phi(t{+}2) = \mathsf{Lo} \ \wedge \\ & \qquad \phi(t{+}3) = \mathsf{Lo} \ \ ) \wedge \\ & (\forall t.\ \phi(t{+}4) = \phi(t)\ )\end{aligned} \tag{13}$$

$$\begin{aligned}\mathsf{Cycle}\ \phi \ =_{def}\ & \exists n.\ (0 \leq n \leq 3) \ \wedge \\ & \qquad \forall t.\ \phi(t) = ((\mathsf{MOD4}\ t \ = \ n) \rightarrow \mathsf{Hi} \mid \mathsf{Lo})\end{aligned} \tag{14}$$

Where MOD4 is the remainder of dividing its argument by 4.

Note how the various properties of Clock are separated into different predicates. This is done deliberately so that we can follow the formalisation more easily and also that it simply reads better.

3.5 Formal Composition Rules for CLIC

The rules governing the interconnection of CLIC gates have been described earlier in a rather lengthy and informal way. If we are to be able to formalise them then understanding the reason behind them is necessary. Consider for example an n-type gate driving a p-type gate. During the precharge period the output of the n-type gate is precharged Hi, which means that the inputs of the p-type gate are at the correct level, namely that the transistors are off and the output node of the p-type gate would be isolated if it went into the evaluation phase now. So when the clock changes and puts both of these gates into the evaluation phase, the output of the p-type gate does not change unless and until its inputs change. However if we had the other situation where the input of the p-type gate was held Lo then as soon as the gate went into its evaluation phase the output would change to Hi. Now no matter what happens to the inputs, the output cannot be changed to Lo until the next precharge period. So effectively the gate has erroneously changed its output value.

In summarising this we can say that the inputs of a p-type gate *must* not have Lo to Hi transitions during its evaluation phase and also the output of an n-type gate *does* not have a Lo to Hi transition during its evaluation phase. So to capture this sort of behaviour we need a single predicate which captures the output behaviour of the n-type gate and the constraints on the inputs of the p-type gate.

Here is the formal definition of such a predicate WB as used in our system. It relies on the fact that the gates are clocked and that the clock is correctly behaved.

$$\mathsf{WB}\ x\ \phi \ =_{def}\ \forall t.\ \begin{pmatrix} \phi(t{+}1) & = \phi(t)\ \wedge \\ x(t) & = \phi(t) \end{pmatrix} \ \supset\ x(t{+}1) = x(t) \qquad (15)$$

This says that the node x is defined to be "Well Behaved" with respect to ϕ where ϕ is one of the four clock line as defined by Clock.

So for example the output of an n-type gate driven by $\overline{\phi}_1$ satisfies "WB *op* ϕ_1," and this is exactly the required input conditions for a p-type gate driven by ϕ_1. In this context what "WB *op* ϕ_1" means is that while ϕ_1 is Lo, *i.e.* the n-type gate is in its evaluation phase, then the *op* cannot have a rising edge on it. This is exactly right since once the output of an n-type gate has been discharged then it cannot go to Hi again until the clock rises and precharges the gate as discussed above.

With this one predicate we have now condensed all of the rules which were listed in a rather informal way. However this predicate relies on the formal definition of Clock and must always be used in conjunction with it. This is not a restriction since CLIC is a dynamic design style and so all CLIC gates will require the existence of Clock for their correct behaviour.

3.6 Formal Derivations of CLIC Primitive Gates

Having got the preliminaries out of the way we can now begin the derivations of the correctness statements for the various CLIC gates. There are four types of gates in the CLIC design methodology, namely the n-type gate, p-type gate, the latch and the static inverter. Statements of correctness can be individually derived for the latch and the static inverter, but it would be foolish to simply derive a statement of correctness for each of the various n-type and p-type gates separately. Rather than doing this it is far better to derive some general theorems which will then be useful for generating the statement of correctness for the individual n-type and p-type gates.

3.6.1 N-type and P-type Logic Gates

For any general theorems to be proved of n-type or p-type gates we first need to extract out what is common to the various gates. A simple split would be to separate the set of components which perform the logic specific function into one bag and the remainder into an other. We call the remainder of an n-type gate the N_Shell, since it has a hole in it into which other components need to be inserted before it can function as an n-type CLIC gate.

```
      ----------+----------
                |
               -'
         .---0||
         |     -.
         |      |
         |      \    .------.
         |       >---| CAP3 |--- op
         |      /    '------'
         |      |
         |      ip
         |
         |
         |      o1
         |      |
         |     -'
phi -----+----||
               -.
                |
      ----------+----------
```

Figure 10: N_Shell as used in CLIC

This is illustrated in figure 10 and can be formally stated as follow:

$$\mathsf{N\_Shell}(\phi, o_1, ip, op) \quad =_{def} \quad \begin{array}{ll} \exists p_0\ p_1\ p_2\ p_3. & \\ \quad \mathsf{Gnd}(p_0) & \wedge \\ \quad \mathsf{Vdd}(p_1) & \wedge \\ \quad \mathsf{N\_Tran}(\phi, p_0, o_1) & \wedge \\ \quad \mathsf{P\_Tran}(\phi, p_1, p_2) & \wedge \\ \quad \mathsf{Join}(p_2, ip, p_3) & \wedge \\ \quad \mathsf{Cap}_3(p_3, op) & \end{array} \tag{16}$$

Cap_3 in the above definition is simply a capacitor with a "memory" of three units of time just as Cap_1 has a "memory" of one unit of time. Note that Cap_3 is derived by composing three Cap_1 devices together.

Before we can progress further we need to define the property which is held true of all those cluster of devices which may be inserted into this N_Shell. By studying the mechanism of an n-type gate we note that the cluster of devices which get inserted in the N_Shell perform one of two functions—they either maintain a link between the *ip* and the o_1 nodes of the N_Shell, or they don't. We call this property Opt_Link and it can be formally stated as follows:

$$\mathsf{Opt\_Link}(ip, op) \quad =_{def} \quad \forall t.\ (op\ t = ip\ t) \vee (op\ t = \mathsf{Zz}) \tag{17}$$

Here it is worth noting that the true property of two nodes being linked or not linked is not actually captured because we are using a directional flow of information model. The best we can do under the circumstances as stated, is say that the values on the two node are equal or that the input node to the N_Shell has a floating value on it. This is still not quite enough but we'll leave the extra conditions until later when they are needed. However there is enough for the following two properties of the N_Shell to be derived.

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) \wedge \\ \mathsf{N\_Shell}(\overline{\phi}_1, a, b, op) \ \wedge \\ \mathsf{Opt\_Link}(a, b) \end{pmatrix} \supset \ \mathsf{WB}\ op\ \phi_1 \tag{18}$$

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) & \wedge \\ \mathsf{N\_Shell}(\overline{\phi}_1, a, b, op) & \wedge \\ \mathsf{Opt\_Link}(a, b) & \wedge \\ \phi_1(t) = \mathsf{Hi} & \end{pmatrix} \supset \begin{pmatrix} \mathsf{Def}\ op\ \ t & \wedge \\ \mathsf{Def}\ op\ (t{+}1) & \wedge \\ \mathsf{Def}\ op\ (t{+}2) & \wedge \\ \mathsf{Def}\ op\ (t{+}3) & \end{pmatrix} \tag{19}$$

Where $\quad \mathsf{Def}\ a\ t \quad =_{def} \quad (a(t) = \mathsf{Hi}) \vee (a(t) = \mathsf{Lo})$

The first theorem can be interpreted as saying that *if* the N_Shell is implemented correctly *and* it is correctly driven by clock *and* the cluster of devices placed in it are correctly behaved in that they have the property of Opt_Link *then* the output will be "Well Behaved," *i.e.* the output will not have Lo to Hi transitions during the evaluation phase. The second theorem simply says that given the same assumptions and assuming that at some time t the clock phase ϕ_1 goes Hi then the output will be "Well Defined" for the times t to $t+4$, *i.e.* the output will be either Hi or Lo.

Now that we have these general theorems we must ensure that this Opt_Link property is derivable for all the various sorts of cluster of elements that may be inserted in the N_Shell. For this to be truly general it will require us to talk of the structure of an arbitrary cluster of devices.

Any logic function which is implementable can be simplified into a network of transistors which only includes transistors in series and/or transistors in parallel with the outputs of the transistors joined together by the Join device. On the basis of this the following three theorems together allow us to show that any cluster containing only parallel and/or series transistor networks, can be shown to maintain the Opt_Link property across the two terminals by which the cluster is connected to the N_Shell.

$$\mathsf{N\_Tran}(g,i,o) \supset \mathsf{Opt\_Link}(i,o) \tag{20}$$

$$\begin{pmatrix} \mathsf{Opt\_Link}(a,b) \wedge \\ \mathsf{Opt\_Link}(b,c) \end{pmatrix} \supset \mathsf{Opt\_Link}(a,c) \tag{21}$$

$$\begin{pmatrix} \mathsf{Opt\_Link}(a,b) \wedge \\ \mathsf{Opt\_Link}(a,c) \wedge \\ \mathsf{Join}(b,c,d) \end{pmatrix} \supset \mathsf{Opt\_Link}(a,d) \tag{22}$$

To illustrate this let's look at a simple example namely the two input Nand gate of figure 7. The structure of this can be formally defined as follows:

$$\begin{array}{lll} \mathsf{N\_Nand\_Imp}(\phi, ip_1, ip_2, op) & =_{def} & \exists p_1\ p_2\ p_3. \\ & & \quad \mathsf{N\_Shell}(\phi, p_1, p_3, op)\ \wedge \\ & & \quad \mathsf{N\_Tran}(ip_1, p_1, p_2)\ \ \wedge \\ & & \quad \mathsf{N\_Tran}(ip_2, p_2, p_3) \end{array} \tag{23}$$

Now by using theorems 18, 19, 20, 21 and 22 we can derive the following two properties. This states that the output of a two input Nand gate is "well behaved" and that the output is also "well defined" over a certain interval of time with reference to the point when ϕ_1 is Hi.

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) \qquad \wedge \\ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1, ip_1, ip_2, op) \end{pmatrix} \supset \mathsf{WB}\ op\ \phi_1 \tag{24}$$

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) & \wedge \\ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1, ip_1, ip_2, op) & \wedge \\ \phi_1(t) = \mathsf{Hi} & \end{pmatrix} \supset \begin{pmatrix} \mathsf{Def}\ op\ t & \wedge \\ \mathsf{Def}\ op\ (t{+}1) & \wedge \\ \mathsf{Def}\ op\ (t{+}2) & \wedge \\ \mathsf{Def}\ op\ (t{+}3) & \end{pmatrix} \tag{25}$$

So far we have only demonstrated that the output of n-type gates are "well behaved" and "well defined," but nothing has been said about the derivation of the logical behaviour of these gates. For this we need theorems considerably more complex than those given for Opt_Link. These properties include Link, No_Link and WB_Link which state under what circumstances a "link" exists across the two nodes of the cluster of devices inserted in the N_Shell. The line of thought regarding work on these is very similar to that followed for Opt_Link, so we shall not deal with them here. However the other two theorems for the two input nand gate giving its logical behaviour are presented below. They use an abstraction function Val_Abs which maps the values Hi and Lo to true and false respectively.

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) & \wedge \\ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1, ip_1, ip_2, op) & \wedge \\ \phi_1(t) = \mathsf{Hi} & \wedge \\ \mathsf{Def}\ ip_1\ (t{+}1) & \wedge \\ \mathsf{Def}\ ip_2\ (t{+}1) & \end{pmatrix} \supset \tag{26}$$
$$\Big(\mathsf{Val\_Abs}\ op\ (t{+}1) = \sim(\mathsf{Val\_Abs}\ ip_1\ (t{+}1) \ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}1))\Big)$$

$$\begin{pmatrix} \mathsf{Clock}(\phi_1, \overline{\phi}_1, \phi_2, \overline{\phi}_2) & \wedge \\ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1, ip_1, ip_2, op) & \wedge \\ \mathsf{WB}\ ip_1\ \overline{\phi}_1 & \wedge \\ \mathsf{WB}\ ip_2\ \overline{\phi}_1 & \wedge \\ \phi_1(t) = \mathsf{Hi} & \wedge \\ \mathsf{Def}\ ip_1\ (t{+}1) & \wedge \\ \mathsf{Def}\ ip_2\ (t{+}1) & \wedge \\ \mathsf{Def}\ ip_1\ (t{+}2) & \wedge \\ \mathsf{Def}\ ip_2\ (t{+}2) & \end{pmatrix} \supset \tag{27}$$
$$\Big(\mathsf{Val\_Abs}\ op\ (t{+}2) = \sim(\mathsf{Val\_Abs}\ ip_1\ (t{+}2) \ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}2))\Big)$$

The treatment for p-type logic gates follows exactly the same line of argument, even to the point where considerable number of the intermediate results are common to both.

3.6.2 The Latch

This is also known as the C^{2}MOS latch and its structure is shown in figure 11. This is Formally captured as follows:

```
       ----------+---------
                 |
                -'
      .--------o||
      |         -.
      |          |
      |         -'
      |     ---o||
      |  phi'   -.
      |          |
      |           \    .------.
ip ---+            >---| CAP3 |--- op
      |           /    '------'
      |          |
      |         -'
      |     ----||
      |  phi    -.
      |          |
      |         -'
      '---------||
                -.
                 |
       ----------+---------
```

Figure 11: The Latch as used in CLIC

$$
\begin{array}{lll}
\mathsf{Latch\_Imp}(\phi,\overline{\phi},ip,op) & =_{def} & \exists p_0\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6. \\
 & & \mathsf{Gnd}(p_0) \quad \wedge \\
 & & \mathsf{Vdd}(p_1) \quad \wedge \\
 & & \mathsf{N\_Tran}(ip,p_0,p_2) \wedge \\
 & & \mathsf{N\_Tran}(\phi,p_2,p_4) \wedge \\
 & & \mathsf{P\_Tran}(\overline{\phi},p_3,p_5) \wedge \\
 & & \mathsf{P\_Tran}(ip,p_1,p_3) \wedge \\
 & & \mathsf{Join}(p_4,p_5,p_6) \quad \wedge \\
 & & \mathsf{Cap_3}(p_6,op)
\end{array}
\tag{28}
$$

Since this is simply a one off result, the derivation is not important but what is important is the result so only that is presented. The full behaviour of the Latch device is summarised in the following three theorems.

$$
\begin{pmatrix}
\mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge \\
\mathsf{Latch\_Imp}(\phi_1,\overline{\phi}_1,ip,op) &
\end{pmatrix}
\supset
\begin{pmatrix}
\mathsf{WB}\ op\ \phi_1 & \wedge \\
\mathsf{WB}\ op\ \overline{\phi}_1 &
\end{pmatrix}
\tag{29}
$$

$$
\begin{pmatrix}
\mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge \\
\mathsf{Latch\_Imp}(\phi_1,\overline{\phi}_1,ip,op) & \wedge \\
\phi_1(t) = \mathsf{Hi} & \wedge \\
\mathsf{Def}\ ip\ t &
\end{pmatrix}
\supset
\begin{pmatrix}
\mathsf{Def}\ op\ t & \wedge \\
\mathsf{Def}\ op\ (t{+}1) & \wedge \\
\mathsf{Def}\ op\ (t{+}2) & \wedge \\
\mathsf{Def}\ op\ (t{+}3) &
\end{pmatrix}
\tag{30}
$$

$$\begin{pmatrix} \mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge \\ \mathsf{Latch\_Imp}(\phi_1,\overline{\phi}_1,ip,op) & \wedge \\ \phi_1(t) = \mathsf{Hi} & \wedge \\ \mathsf{Def}\ ip\ t & \end{pmatrix} \supset \begin{pmatrix} \mathsf{Val\_Abs}\ op\ t & = & \sim\mathsf{Val\_Abs}\ ip\ t & \wedge \\ \mathsf{Val\_Abs}\ op\ (t+1) & = & \sim\mathsf{Val\_Abs}\ ip\ t & \wedge \\ \mathsf{Val\_Abs}\ op\ (t+2) & = & \sim\mathsf{Val\_Abs}\ ip\ t & \wedge \\ \mathsf{Val\_Abs}\ op\ (t+3) & = & \sim\mathsf{Val\_Abs}\ ip\ t & \end{pmatrix} \tag{31}$$

The first of these captures the fact that the output may drive any of p-type or n-type gates, even both at the same time. The second theorem states that the output is "well defined" so the results can be abstracted into the boolean domain by use of the Val_Abs abstraction function. Finally the third gives the logical behaviour between the input and the output at the abstract level, *i.e.* on the clock tick the input is inverted and passed to the output where it is held static until the next clock tick.

3.6.3 The Static Inverter

This is the only device in the entire CLIC design style which does not need one of the clock lines for it to function correctly. It has only two external ports namely the input (ip) and output (op) ports and its behaviour could perfectly be defined without the use of the Clock predicate. However, to enable it to be used in conjunction with other dynamic CLIC devices, its correctness statement has to be given in the same *form*. So we begin by giving the formal definition for the structure of the gate as follows:

$$\begin{array}{lll} \mathsf{Stat\_Inv\_Imp}(ip,op) & =_{def} & \exists p_0\ p_1\ p_2\ p_3. \\ & & \quad \mathsf{Gnd}(p_0) \qquad \wedge \\ & & \quad \mathsf{Vdd}(p_1) \qquad \wedge \\ & & \quad \mathsf{N\_Tran}(ip,p_0,p_2) \ \wedge \\ & & \quad \mathsf{P\_Tran}(ip,p_1,p_3) \ \wedge \\ & & \quad \mathsf{Join}(p_2,p_3,op) \end{array} \tag{32}$$

Now the usual three properties can be derived for this gate. The first one being that it's output is "well behaved."

$$\begin{pmatrix} \mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) \ \wedge \\ \mathsf{Stat\_Inv\_Imp}(ip,op) \end{pmatrix} \supset \begin{pmatrix} (\mathsf{WB}\ ip\ \overline{\phi}_1 \supset \mathsf{WB}\ op\ \phi_1) & \wedge \\ (\mathsf{WB}\ ip\ \phi_1 \supset \mathsf{WB}\ op\ \overline{\phi}_1) & \wedge \\ (\mathsf{WB}\ ip\ \overline{\phi}_2 \supset \mathsf{WB}\ op\ \phi_2) & \wedge \\ (\mathsf{WB}\ ip\ \phi_2 \supset \mathsf{WB}\ op\ \overline{\phi}_2) & \end{pmatrix} \tag{33}$$

Inspecting this theorem reveals that it is in a different *form* than the others so far. In fact it is not so different as to not allow logical inferences to be made using the same techniques. However if it were to be put in the same form as the ones so far, we would have four different theorems giving rise to the four different clauses. Remember that the inverter is used to invert the *polarity* of a gate so that a gate may drive its own sort, *e.g.* a p-type gate may drive another p-type gate only if it is buffered by an inverter. Since there are two different sorts of gates, n-type and p-type, and two clock phases, the need arises for four very similar theorems, or one containing all four clauses.

The remaining two theorems for this device are fairly standard. In fact they are even simplified a little to take advantage of the fact that this device is not clocked. The next theorem for instance simply states that "if the input is defined then so is the output." The last one gives the logical behaviour of the gate appropriately abstracted to the boolean level using the Val_Abs predicate.

$$\begin{pmatrix} \mathsf{Stat\_Inv\_Imp}(ip, op) \wedge \\ \mathsf{Def}\ ip\ t \end{pmatrix} \supset \mathsf{Def}\ op\ t \tag{34}$$

$$\begin{pmatrix} \mathsf{Stat\_Inv\_Imp}(ip, op) \wedge \\ \mathsf{Def}\ ip\ t \end{pmatrix} \supset (\mathsf{Val\_Abs}\ op\ t\ =\ \sim\mathsf{Val\_Abs}\ ip\ t) \tag{35}$$

3.7 Formalising the CLIC Circuit Design Methodology

So far we have outlined a method for deriving the correctness statement of any logic gate designed in the CLIC design style. If we are to design real circuits with these correctness statements, rather than just admire their elegance and still use the old rules of thumb, then we must provide formal means of doing so. *i.e.* a formal method of combining the correctness statements of an arbitrary number of gates resulting in a new correctness statement for the new circuit.

What the designer is interested in is simply obtaining the logical behaviour of the system so that he may satisfy himself that the system does what he intended it to do. So a technique is needed which allows the designer to compose the logical behaviour component of the specifications and leave the rest of the "checking" to the system. In our system simple logical inferences would be used to check the validity of connecting together the output of one gate to the input of an other. In fact the rule involved is the resolution of the predicates and their arguments governing the constraints on the inputs of devices, against the predicates and their arguments governing the properties of the outputs. Work is in hand at present to automate this process.

To illustrate the use of this consider the the implementation of a simple logical AND function with delay. Firstly, in order to get delay we will have to use a latch since this is the only device in the CLIC design style which allows behaviours between input and output to be mapped across time giving a controlled unit delay with respect to the clock. So the solution we shall choose to implement is to drive the output of an n-type nand gate, such as the one already used in an earlier example, by a latch. This is illustrated in figure 12.

```
                                 phi2'
           |-----\                |\
           |      \               | \
ip ----| N       >O--------|L >O----- op
           |      /               | /
           |-----/                |/
            phi1'                phi2
```

Figure 12: A simple CLIC Circuit

The theorems needed for these two devices have already been derived earlier in this paper, namely theorems 27 and 31, for the nand gate and the latch respectively. The correctness statement for the nand gate is exactly as needed but that of the latch needs to be messaged into a form which allows these two to be combined. Given below are the two theorems for these two devices in their correct form just before they are to be combined. There are a number of important steps involved before we get to this state involving a lemma about clock but these are not covered here. Further details regarding these intermediate steps can be found in [3].

$$\begin{pmatrix} \mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge \\ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1, ip_1, ip_2, op) & \wedge \\ \mathsf{WB}\ ip_1\ \overline{\phi}_1 & \wedge \\ \mathsf{WB}\ ip_2\ \overline{\phi}_1 & \wedge \\ \phi_1(t) = \mathsf{Hi} & \wedge \\ \mathsf{Def}\ ip_1\ (t+1) & \wedge \\ \mathsf{Def}\ ip_2\ (t+1) & \wedge \\ \mathsf{Def}\ ip_1\ (t+2) & \wedge \\ \mathsf{Def}\ ip_2\ (t+2) & \end{pmatrix} \supset \tag{36}$$
$$\big(\mathsf{Val\_Abs}\ op\ (t+2) = \sim(\mathsf{Val\_Abs}\ ip_1\ (t+2)\ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t+2))\big)$$

$$\left(\begin{array}{ll}\mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge\\ \mathsf{Latch\_Imp}(\phi_2,\overline{\phi}_2,ip,op) & \wedge\\ \phi_1(t) = \mathsf{Hi} & \wedge\\ \mathsf{Def}\ ip\ t{+}2 & \end{array}\right) \supset$$
$$\left(\begin{array}{lcll}\mathsf{Val\_Abs}\ op\ (t{+}2) & = & \sim\mathsf{Val\_Abs}\ ip\ (t{+}2) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}3) & = & \sim\mathsf{Val\_Abs}\ ip\ (t{+}2) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}4) & = & \sim\mathsf{Val\_Abs}\ ip\ (t{+}2) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}5) & = & \sim\mathsf{Val\_Abs}\ ip\ (t{+}2) & \end{array}\right) \tag{37}$$

Now we can combine these two by using Modus Ponens and Conjunction rules together with theorem 25 which satisfies the input constraint for the latch. The final result together with hiding the internal line using the existential quantifier looks like the following:

$$\left(\begin{array}{ll}\mathsf{Clock}(\phi_1,\overline{\phi}_1,\phi_2,\overline{\phi}_2) & \wedge\\ \left(\begin{array}{l}\exists x.\ \ \mathsf{N\_Nand\_Imp}(\overline{\phi}_1,ip_1,ip_2,x)\ \wedge\\ \qquad \mathsf{Latch\_Imp}(\phi_2,\overline{\phi}_2,x,op)\end{array}\right) & \wedge\\ \mathsf{WB}\ ip_1\ \overline{\phi}_1 & \wedge\\ \mathsf{WB}\ ip_2\ \overline{\phi}_1 & \wedge\\ \phi_1(t) = \mathsf{Hi} & \wedge\\ \mathsf{Def}\ ip_1\ (t{+}1) & \wedge\\ \mathsf{Def}\ ip_2\ (t{+}1) & \wedge\\ \mathsf{Def}\ ip_1\ (t{+}2) & \wedge\\ \mathsf{Def}\ ip_2\ (t{+}2) & \end{array}\right) \supset \tag{38}$$
$$\left(\begin{array}{lcll}\mathsf{Val\_Abs}\ op\ (t{+}2) & = & (\mathsf{Val\_Abs}\ ip_1\ (t{+}2)\ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}2)) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}3) & = & (\mathsf{Val\_Abs}\ ip_1\ (t{+}2)\ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}2)) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}4) & = & (\mathsf{Val\_Abs}\ ip_1\ (t{+}2)\ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}2)) & \wedge\\ \mathsf{Val\_Abs}\ op\ (t{+}5) & = & (\mathsf{Val\_Abs}\ ip_1\ (t{+}2)\ \wedge\ \mathsf{Val\_Abs}\ ip_2\ (t{+}2)) & \wedge\end{array}\right)$$

Note that this theorem which gives the combined behaviour of the n-type nand gate and the latch is in the same form as the correctness statements for the two devices from which it is built. This particular feature helps our formal approach of combining CLIC gates together to be expanded to cover circuits of arbitrary complexity. The correctness statements will increase in size in combining large and complex circuits but will not change in their inherent structure.

4 Discussion and Future Work

A full formal presentation has been given for the CLIC design style which we believe to be suitable for VLSI. In particular a *form* for the correctness statement of CLIC gates has been developed which maintains uniformity of specifications across the many levels of hierarchy of circuit design—from the very simple logic gates to fairly complex structures using many macro blocks. The use of these formal techniques have been demonstrated on a simple example which uses many CLIC gates, both simple and complex. A major case study based on the design of a digital phase-locked loop is in progress which demonstrates the use of these techniques on large systems.

Naturally the work presented is only as good as the axioms on which it is based. Current models used for the primitive devices of integrated circuits are simplistic, with the view to making the proofs of correctness easier. These models are not inaccurate, but merely incomplete. They have only the features which are relevant to the design style; other properties are not modelled. Too simplistic a model of these devices, however, may allow a failure mode to pass unobserved. So proofs based on such models become void. More realistic models are needed for these primitives, together with means of showing that the simple models suffice in controlled environments. It is hoped that research in this area will support most of the work done to date using simpler models, by formally showing that the simpler models are adequate in the environment in which they are used. Some results in this area are already available [9], where the simulation model used in [1] is embedded in logic. However this is not yet developed to the point where it is usable for the dynamic behaviour of circuits.

Acknowledgements

I would like to thank all members of the Cambridge Hardware Verification Group. In particular Tom Melham and Jeff Joyce who made valuable comments on earlier drafts of this paper. Special thanks are due to Mike Gordon who provided the necessary stimulus for this work and who was a continuous source of encouragement. I would also like to thank Dave Orton of Racal Research and Glynn Winskel of Cambridge University for providing many stimulating discussions in hardware and formal methods respectively.

References

[1] Randal Everitt Bryant
"A Switch-Level Simulation Model for Integrated Logic Circuits," PhD. Thesis, also available as a Technical Report MIT/LCS/TR-259, Laboratory for Computer Science, MIT, Massachusetts, March 1981.

[2] A. Camilleri, M. Gordon and T. Melham
"Hardware Verification using Higher-Order Logic," In: *Proceedings of the IFIP International Conference: From H.D.L. Descriptions to Guaranteed Correct Circuit Designs*, Grenoble, September 9–11, 1986.

[3] I. S. Dhingra
Ph.D. Thesis. Computer Laboratory, University of Cambridge. 1987

[4] Nelson F. Goncalves and Hugo J. De Man
"NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures," *IEEE Journal of Solid-State Circuits*, SC–18 (3), June 1983 pp. 261–266.

[5] M. J. C. Gordon
"HOL: A Machine Oriented Formulation of Higher-Order Logic," Technical Report 68, Computer Laboratory, University of Cambridge, 1985.

[6] M. J. C. Gordon
"Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware," In: *Formal Aspects of VLSI Design*, edited by G. Milne and P. A. Subrahmanyam, North-Holland, 1986.

[7] R. H. Krambeck, Charles M. Lee and Hung-Fai Stephen Law
"High-Speed Compact Circuits with CMOS," *IEEE Journal of Solid-State Circuits*, SC–17 (3), June 1982 pp. 614–619.

[8] D. W. R. Orton
"Clocked Dynamic Logic for CMOS," Racal Research Internal Memo of 10*th* January 1984, Worton Drive, Worton Grange Industrial Est., Reading RG2 OSB, England.

[9] Glynn Winskel
"Models and Logic of MOS Circuits," in *VLSI Specification, Verification and Synthesis*, Ed. G. Birtwistle and P.A. Subrahmanyam, Kluwer 1987. (These proceedings)