Asynchronous Nano-electronics: Preliminary Investigation

Alain J. Martin & Piyush Prakash
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125, USA

Abstract

This paper is a preliminary investigation in implementing asynchronous QDI logic in molecular nano-electronics, taking into account the restricted geometry, the lack of control on transistor strengths, the high timing variations. We show that the main building blocks of QDI logic can be successfully implemented; we illustrate the approach with the layout of an adder stage. The proposed techniques to improve the reliability of QDI apply to nano-CMOS as well.

1. Introduction

Currently, molecular nano-electronics is considered a plausible successor to CMOS. Although immense fabrication difficulties still lie ahead, at least experimental devices are feasible today. Molecular nano-electronics is a mostly non-lithographic bottom-up fabrication technology whose main advantage over CMOS is to break the limit of lithography in terms of feature size and device density. (Densities in the range of 10^{10} to 10^{12} devices per cm^2 are mentioned in the literature [2, 3].)

All technologies at the nanoscale level, including nano-CMOS, will face great fabrication challenges that will translate into important parameter variations and decreased reliability. Timing will be difficult to control and predict. Wires will be short, by necessity and by choice: the technology does not allow to grow long wires reliably and wire delay is quadratic in the length. Consequently, it will be impossible to build a useful global clocking network with those technologies. For those two main reasons—difficulty in controlling and predicting timing and impossibility of building a clock network—asynchronous logic seems an ideal and perhaps unavoidable choice for digital circuits in this technology. QDI, which among all types of asynchronous methods, relies on the weakest timing assumption (isochronic fork), seems particularly well suited.

This paper is a preliminary investigation in implementing asynchronous (QDI) logic in molecular nanotechnology. It is first and foremost an assessment, and further improvement, of QDI's robustness to extreme parameter variations and restricted geometry. The paper is organized as follows. First, we briefly describe the main features of this new technology. Among several possible alternatives we choose a somewhat idealized one based on complementary FETs. We define its main layout rules. Next, we show how to design basic combinational gates. State-holding elements like C-elements and precharge function-blocks require some attention because the traditional "staticizer" implementation must be avoided. We give an implementation for those cells that avoid staticizers. We show how the logic family of the Caltech Asynchronous Microprocessor(CAM), and the logic family of the MiniMIPS can be both implemented without use of staticizers. We illustrate the design style with the implementation of a ripple-carry adder. Finally, the implementation of isochronic forks with its implied timing assumption is analyzed.

2. Molecular Nanotechnology

Our target technology is a somewhat advanced version of the devices produced in the Heath Lab at Caltech, the Lieber Group at Harvard and the HP group, [5, 6, 1]. Chemists are able to grow silicon nanowires (NW) with diameters around 5nm and up to ten microns in length. These wires are aligned in one direction, either by a flow process or by nano-imprint. Because of the difficulty in arranging the wires in a regular pattern, the feasible geometry is restricted to a crossbar: a collection of parallel wires in one direction superimposed with a collection of parallel wires in the orthogonal direction. However, the grid is not perfectly regular and several wires may be broken.

Interesting things may be made to happen at the intersection of two orthogonal wires. Special molecules (e.g. rotaxane) can be placed at the crosspoint between two wires to connect the two wires. Such a junction is originally of high enough resistance that the two wires are not electrically connected. But, if a high voltage is applied between the two wires, the molecule will begin conducting and continue to do so when the voltage is lowered. In other words, by applying voltage at selected junctions, the junctions can be programmed to conduct. The junctions can also be made to rectify, i.e., we can program both resistors and diodes.

Resistors and diodes are sufficient to build a complete logic family and have been used to build PLAs and memories, but they have no gain (amplification) and therefore signal degradation is unavoidable, leading to failure in any computation containing a significant number of transitions in series. In order to implement general computations, transistors are needed to provide amplification, which in turn requires to be able to make semi-conductors out of silicon NWs. NWs can be doped during the growth in order to control their conductivity. Heavily doped regions of a wire are conducting; some regions can be kept lightly doped so as to control their conductivity via an electrical field created by applying a voltage on the crossing metal wire. If a dielectric can be placed at the intersection of (and between) the two wires, one has effectively created a FET at the junction. (So far no technology offers programmable FETs, unlike resistive junctions and diode junctions.)

Technologies with one type of transistors (usually pFET), diodes, and resistive pullups or pulldowns have been announced and demonstrated. The transistors have enough gain to build restoring logic. Recently, all three groups mentioned have announced that they will soon (or already do) fabricate FETs of both n- and p-types, with appropriate threshold voltages and acceptable gain, making it possible to design complementary logic circuits.

Although the characteristics of the transistors are still shrouded in mystery, we take as working hypothesis that such devices will be built. We also assume the availability of low-resistance metal wires for the grid direction that provides the gates of transistors. (Such metal wires are obtained either by coating silicon wires with metal during crystal growth, or by imprinting pure metal wires using the nano-imprint technique.)

Let us summarize the components of our (slightly precursory) target technology. As already mentioned, we believe that most properties of this technology will apply to nano-CMOS. The basic building block is the tile. A tile is a crossbar array of nanowires: the (top) horizontal wires are metal conducting wires that are used as gates of transistors; the (bottom) vertical wires are semiconducting silicon nanowires (NW) used as channels of transistors. Wire length is strictly limited, say around 10μm, with a wire-to-wire pitch of approximately 10nm. (Metal wires can be packed more densely and can be longer as they are less resistant. But we will ignore those differences.) We assume that we can implement tiles of 100×100 wires.

A tile may be a routing tile or a compute tile. A routing tile contains only programmable junctions and is used to connect two compute tiles. A compute tile consists of an *n-plane*, a *p-plane*, and a *routing plane*. An n-plane contains n-transistors only; a p-plane contains p-transistors only; a routing plane contains programmable connections only.

An important advantage of this technology is that the different active crosspoints (transistors, resistors, diodes) fit exactly under the area of the crosspoint and therefore do not increase the area of the array, hence contributing to the density advantage of the technology.

A few important restrictions and properties must be mentioned. (1) It is not possible to mix the different components (n-transistor, p-transistor, resistor) inside the same plane. Each plane is homogeneous. (2) It is not possible to connect wires end-to-end in one direction. Connections are made through orthogonal wires. (3) Connection resistance is high (100KΩ) and highly variable. (4) The drive of transistors is not well characterized. We assume that the limit to the number of transistors in series is similar to CMOS. We have chosen 3 as a hard limit. (5) Up to 10% of the wires and connections may be broken or stuck open. (6) Finally, the collection of nano-tiles is placed on top of a standard silicon layer. Power distribution and input/output are implemented in the silicon layer. All vertical nanowires in a p-plane are connected to a micro-wire distributing Vdd, and all vertical nanowires in an n-plane are connected to a micro-wire distributing GND.

3. Implementing QDI Logic

Among all asynchronous logic families, QDI is the most robust to parameter variations because it relies on the weakest timing assumption: the *isochronic fork*. The main question addressed in this paper is the following. In the presence of extreme PVT (process, voltage, temperature) parameter variations, how will a QDI circuit (implemented in molecular nano) fail, and what can be done to avoid the failure or at least significantly improve the circuit's robustness?

3.1. Combinational Cells

The implementation of combinational cells does not present any particular problem beside the usual restriction on the length of transistor chains, but it gives us the opportunity to fix a general layout scheme. Figure 1 shows two alternative implementations for the inverter and the nand2. (The nor2 gate is derived form the nand2 by exchanging Vdd and GND, and p- and n-transistors.) A dot at the intersection of two wires indicates a connection, a diagonal bar (/) indicates an n-transistor. A diagonal bar (\) indicates a p-transistor. In the first type of layout, the n- and p-planes are vertically aligned w.r.t. the metal wires; in the second type, they are horizontally aligned w.r.t. the metal wires. It

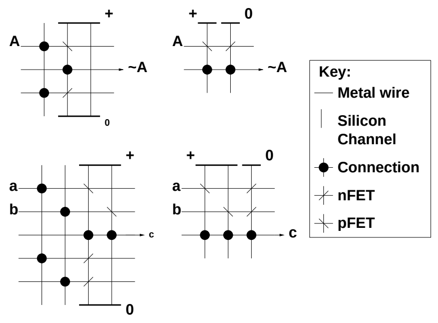

Figure 1. Two possible layouts for the inverter and nand2

is clear that the second choice of layout presents an important area advantage: the area of the nand-gate is 6x3 in the first case and 2x3 in the second case. As shown in Figure 2, the area penalty of the first layout for an n-input combinational gate is $2n^2$. But there is another, more important, advantage to the second layout choice. If an input is shared by several transistor gates, then in the second layout choice, the input signals to the different gates are carried by the same metal wire, resulting in very similar RC characteristics for the different paths. In the first layout choice, the paths from one input to several transistor gates are very dissimilar, with the paths to one type of transistors going through two resistive contacts and the paths to the other type of transistors being a single metal wire. As we shall see, this difference is important for the implementation of isochronic forks.

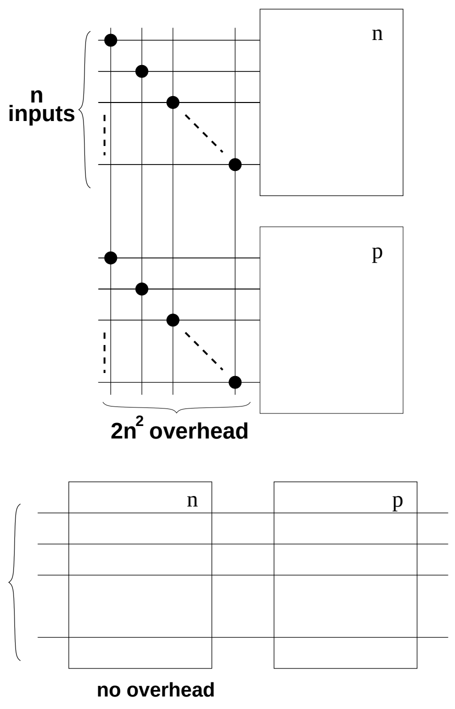

Figure 2. Two possible layouts for an n-input cell

3.2. General Layout Scheme

The general scheme for a compute tile is shown in Figure 3. The two transistor planes are side by side with the input signals running horizontally on metal wires. Each vertical semiconductor wire of the transistor planes can be used (inside a transistor plane) as a pullup or pulldown transistor chain if transistors are placed at some of the cross-points. Because of the restricted geometry, parallel chains cannot share transistors, and therefore all boolean expressions are implemented in the disjunctive normal form: The expression $a \wedge (b \vee c)$ can only be implemented as $a \wedge b \vee a \wedge c$. The general shape of the routing plane is that of an inverted T as shown on the figure. The part of the routing plane between the two transistor planes is used only for the feedback connections needed in the implementation of state-holding gates. (We will give examples shortly.) The bottom part of the routing plane connects the pullup and pulldown chains to the output(s) of the logic gate. (A similar scheme has been proposed in [1].) A single compute tile contains several cells.

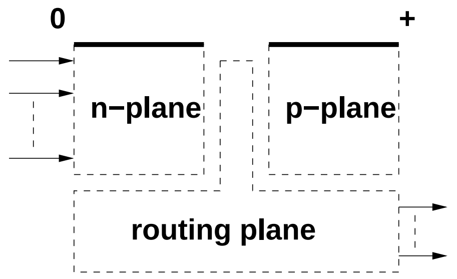

Figure 3. General layout scheme for a compute tile

3.3. State-Holding Operators

The standard CMOS implementation of state-holding operators in this technology does present a serious problem: the weak inverter of the staticizer. Consider the operator defined by the two *production rules*:

$$\begin{array}{lcl} A & \rightarrow & x\uparrow \\ B & \rightarrow & x\downarrow \end{array}$$

Without loss of generality, let's assume that it is implemented as

$$\begin{array}{lcl} A & \rightarrow & x\_\downarrow \\ B & \rightarrow & x\_\uparrow \end{array} \qquad \begin{array}{lcl} x\_ & \rightarrow & x\downarrow \\ \neg x\_ & \rightarrow & x\uparrow \end{array}$$

By definition of a state-holding element, there exist states in the computation where $\neg A \wedge \neg B$ holds. In those "floating" states, the path to Vdd through the pullup implementing B, and the path to GND through the pulldown A are both cut. The voltage value of the physical node implementing $x\_$ has to be maintained in other ways. The general approach is to find another path to Vdd and another path to GND to maintain the current value of $x\_$. The simplest and crudest solution is to add a "staticizer" (or "keeper"). The staticizer implementation consists of *always* maintaining the current value of $x\_$ by adding the pullup $\neg x \rightarrow x\_\uparrow$ and the pulldown $x \rightarrow x\_\downarrow$, giving the gate:

$$\begin{array}{lcl} A \vee x & \rightarrow & x\_\downarrow \\ B \vee \neg x & \rightarrow & x\_\uparrow \end{array}$$

The added circuitry is a feedback inverter with input x and output $x\_$. The difficulty with this solution is that when the value of $x\_$ has to be changed, for instance from true to false, there is a conflict (a "fight") between $A \rightarrow x\_\downarrow$ and $\neg x \rightarrow x\_\uparrow$. Symmetrically, when the value of $x\_$ has to be changed from false to true, there is a "fight" between $B \rightarrow x\_\uparrow$ and $x \rightarrow x\_\downarrow$. There is no logical resolution to those conflicts; they can only be resolved by physical means by making sure that the current through the pullup implementing B and through the pulldown implementing A is stronger than the current through the transistors of the feedback inverter. This is usually achieved by implementing the two transistors of the feedback inverter as highly resistive ("weak"), making the currents through the pullup and the pulldown of the feedback much smaller than the currents through the pullup B and pulldown A. But the weak currents cannot be too weak since they have to maintain the voltage on node $x\_$ in the presence of possibly important leakage.

Hence, the correct behavior of the staticizer relies on a two-sided inequality requirement on the feedback inverter's current. In a technology where the physical parameters of the design are very hard to control and the transistors cannot be sized, it would be very risky to rely on the relative strengths of conducting paths. *We conclude that we will not be able to use staticizers in nano-technology.*

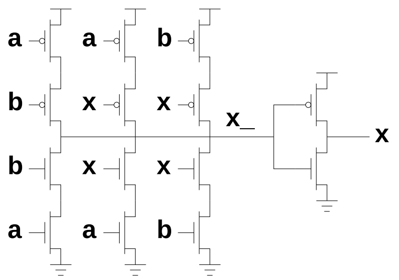

Figure 4. A two-input C-element implemented as a majority gate.

We have to use another solution to maintain the value of the output nodes in the floating states. The general method consists in identifying the floating states in which the value of the output has to be maintained and using the feedback inverter *only in those states.* Without additional information, the floating states are characterized by $\neg A \wedge \neg B$, leading to the general solution:

$$\begin{array}{lcl} A \vee \neg B \wedge x & \rightarrow & x\_\downarrow \\ B \vee \neg A \wedge \neg x & \rightarrow & x\_\uparrow \end{array}$$

3.4. C-element

The Muller C-element is an essential building block of asynchronous logic. The 2-input C-element with inputs a and b and output x is defined as:

$$\begin{array}{lcl} a \wedge b & \rightarrow & x\uparrow \\ \neg a \wedge \neg b & \rightarrow & x\downarrow \end{array}$$

The above transformation applied to this pair of PRs leads to the well-known majority-gate implementation (see Figure 4):

$$\begin{array}{lcl} a \wedge b \vee a \wedge x \vee b \wedge x & \rightarrow & x\_\downarrow \\ \neg a \wedge \neg b \vee \neg a \wedge \neg x \vee \neg b \wedge \neg x & \rightarrow & x\_\uparrow \\ x\_ & \rightarrow & x\downarrow \\ \neg x\_ & \rightarrow & x\uparrow \end{array}$$

For the three-input C-element, the transformation gives:

$$a \wedge b \wedge c \vee a \wedge x \vee b \wedge x \vee c \wedge x \rightarrow x\_\downarrow$$
$$\neg a \wedge \neg b \wedge \neg c \vee \neg a \wedge \neg x \vee \neg b \wedge \neg x \vee \neg c \wedge \neg x \rightarrow x\_\uparrow$$

The layout of the C-elements in Figure 5 shows the use of the midsection of the routing plane for feedback connections (using the output x as gate of some transistors).

4. Read/Write Register

The read/write register is used in a standard pipeline stage of the Caltech Asynchronous Microprocessor (CAM).

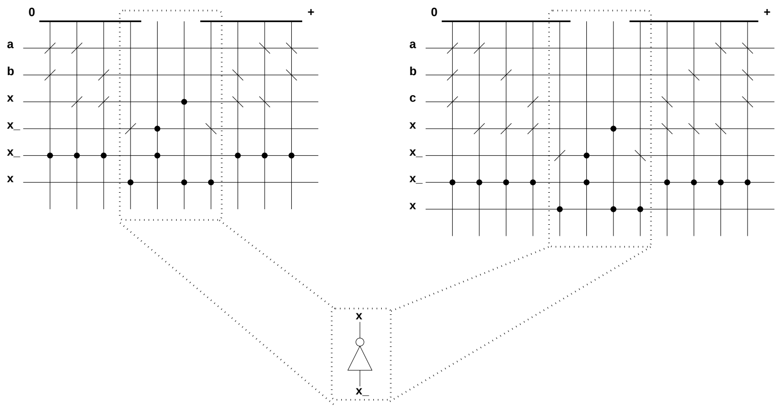

Figure 5. Schematics for a 2-input and a 3-input C-elements. The central part is used for feedback paths.

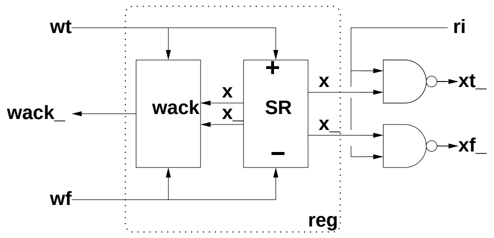

Figure 6. Dual-rail single-bit register with read and write interfaces

It is also used in a slightly different form to implement general-purpose registers in the MiniMIPS and Lutonium.

As shown in Figure 6, in its simplest form, the register consists of three parts: the set/reset latch maintaining the current value of the register bit $(x, x\_)$, the two nand-gates that constitute the read part, and the write-acknowledge (*wack*) cell that generates the acknowledge signal for the write handshake. The implementation of the read part presents no difficulty. The set/reset latch is implemented as cross-coupled nor-gates which do not need relative sizing unlike the implementation with cross-coupled inverters (see [7]). Only the *wack* needs some attention. Its specification is

$$\begin{array}{lcl} wt \wedge x \vee wf \wedge x\_ & \rightarrow & wack\_\downarrow \\ \neg wt \wedge \neg wf & \rightarrow & wack\_\uparrow \end{array}$$

The floating states are $\neg wt \wedge wf \wedge \neg x\_$ and $wt \wedge \neg wf \wedge \neg x$. In both states, the write interface is in the process of changing the values of x and $x\_$ and therefore the output $wack\_$ should be kept high in the floating states. The transformation then gives

$$\neg wt \wedge \neg wf \vee \neg wt \wedge \neg x\_ \vee \neg wf \wedge \neg x \;\rightarrow\; wack\_\uparrow$$

Now the two guards are complementary. *The write-acknowledge is a combinational gate.* Altogether the above design leads to a compact nano layout for the register. No feedback inverter is needed, the only feedback being that of the cross-coupled nor-gates, and there are at most two transistors in series on all paths. See Figure 7.

4.1. Precharge Function

We know that we can implement all computations that don't require arbitration with just standard combinational gates and the 2-input C-element. But performance will suffer, and it takes just a few additional cells to allow efficient circuit implementations. For instance, the C-element, the set-reset latch and the precharge function are enough to implement the circuits of the Caltech MiniMIPS as well as the circuits of the CAM even though the two design styles are different. Again, the set-reset latch can be implemented with cross-coupled nor-gates. The precharge function is a state-holding cell with two sets of inputs: the input *en* is a control signal used to precharge the output node z high in the neutral state ($\neg en$). The other set X of inputs is used to compute the boolean function F when *en* is high. Function F is necessarily small enough to fit in a single pulldown, and therefore the number of inputs of X is also limited—rarely more than four. The PRS for the precharge function cell is:

$$\begin{array}{lclclcl} \neg en & \rightarrow & z\_\uparrow & \quad & z\_ & \rightarrow & z\downarrow \\ en \wedge F & \rightarrow & z\_\downarrow & \quad & \neg z\_ & \rightarrow & z\uparrow \end{array}$$

The floating state is $en \wedge \neg F$. Applying the transformation and after simplification, we get:

$$\begin{array}{lcl} \neg en \vee \neg F \wedge \neg z & \rightarrow & z\_\uparrow \\ en \wedge F \vee en \wedge z & \rightarrow & z\_\downarrow \end{array}$$

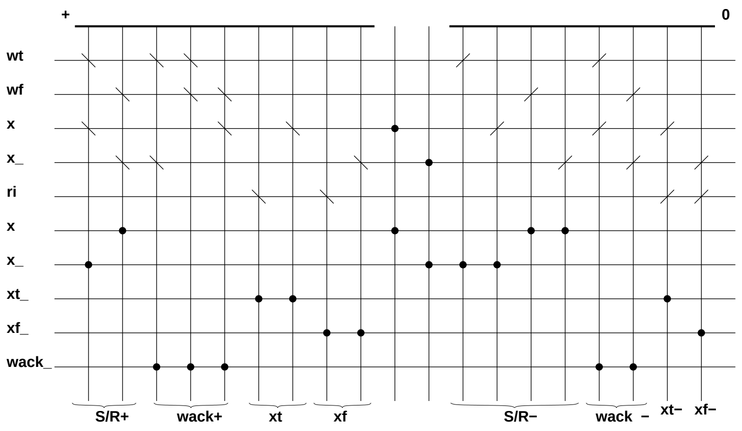

Figure 7. Schematic for the dual-rail read/write register

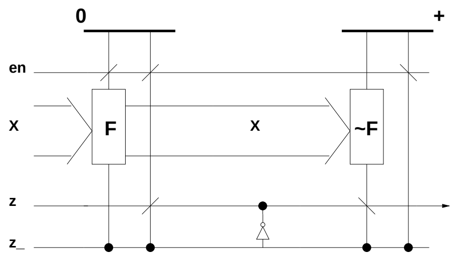

Figure 8. Function block with single output. Both boolean expressions F and $\neg F$ are used to make the gate combinational.

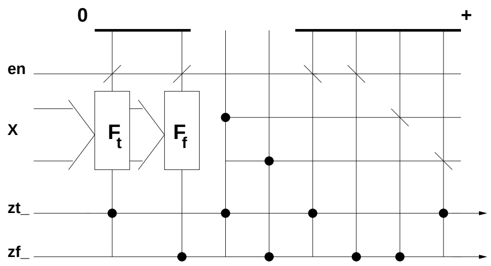

Figure 9. Schematic for a dual-rail function block

A symbolic layout is shown in Figure 8 . For example, if F is the dual-rail equality of a and b, i.e. $f = a0 \wedge b0 \vee a1 \wedge b1$, the complement of F has to be implemented as $\neg a0 \wedge \neg a1 \vee \neg b0 \wedge \neg a1 \vee \neg b1 \wedge \neg a0 \vee \neg b0 \wedge \neg b1$, since unfortunately we can only implement parallel or-terms. This leads to 5 parallel pullups. The added complexity may be significant. But it should be observed that the added pullups and pulldowns are used only to maintain the value of the output in the floating state(s). Therefore, their possible complexity, which translates into a weak drive (I_{ds} current), might be acceptable since those extra terms are not used to switch the output but only to maintain the current value of the output. However, the following simplification significantly improves the circuit. It applies whenever it is required to compute the dual-rail version of the function, i.e.:

$$\begin{array}{lll} \neg en & \rightarrow & zt\_\uparrow \\ en \wedge Ft & \rightarrow & zt\_\downarrow \end{array} \qquad \begin{array}{lll} \neg en & \rightarrow & zf\_\uparrow \\ en \wedge Ff & \rightarrow & zf\_\downarrow \end{array}$$

If the only floating states are $en \wedge Ft \wedge \neg Ff$ for the Ff block, and $en \wedge Ff \wedge \neg Ft$ for the Ft block, then we can staticize the PRS as:

$$\begin{array}{lll} \neg en \vee \neg zf\_ & \rightarrow & zt\_\uparrow \\ \neg en \vee \neg zt\_ & \rightarrow & zf\_\uparrow \end{array}$$

with the pulldowns unchanged. This simplification requires that $\neg en \vee Ft \vee Ff$ be an invariant of the system. In other words, the condition for applying the transformation is that *the function inputs are not reset when en is set.* A symbolic nano-layout for this solution is shown in Figure 9. Certain QDI templates already satisfy this condition. It is the case for the control/data decomposition scheme of the pipeline stage used in the CAM. (It relies on an easily satisfiable isochronicity assumption.) Other QDI templates can easily be transformed to satisfy the invariant. It is the case for the PCHB ("precharge half-buffer") of the MiniMIPS design style. Since the validity (usually called $v(L)$) of the data input is always computed, it suffices to replace the enable signal en with en' defined as $v(L)\underline{C}en$ (i.e. en' is the output of the C-element with en and $v(L)$ as inputs).

5. Implementing an Adder

Next, we describe the implementation of an n-bit adder stage in nano. The adder is designed in the same style as

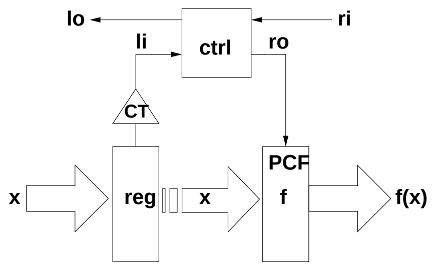

Figure 10. Control/data decomposition of a pipeline stage

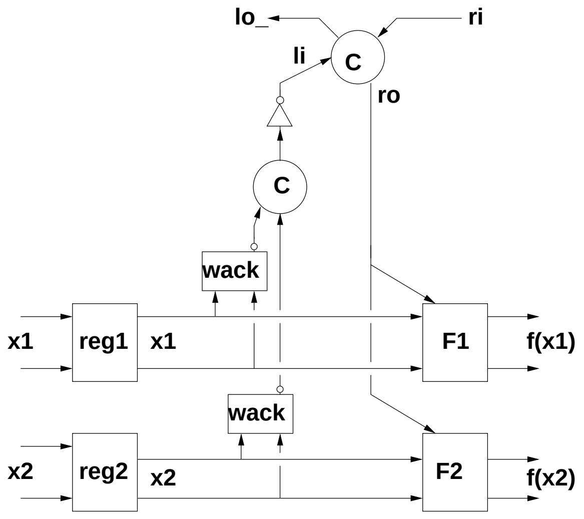

Figure 11. A control/data pipeline stage with half-buffer control

the one used in the CAM. The basic QDI pipeline stage $*[L?x; R!F(x)]$ can be implemented as in Figure 10 using the control/data decomposition approach of the CAM. (See, for instance, [7].) As shown in Figure 11, the control part can be implemented as a simple half-buffer (which can be as simple as a C-element in this case); the datapath consists of a dual-rail register per bit of input with the two nand-gates of the read part replaced by the precharge functions computing the carry and sum for every pair of input bits.

In this design, the condition for removing staticizers is satisfied provided a mild isochronicity assumption is fulfilled, and therefore the function blocks for sum and carry can use the derived transformation. The isochronicity assumption is that the delay for the valid inputs to propagate from the outputs of the registers to the input of the function blocks should be less than the delay of the adversary path consisting of the *wacks* (in parallel), the completion tree, and the C-element in the control.

Furthermore, in order to avoid propagating the control signal en (which is also ro in this case) to all bits of the datapath, we can replace en with $ct \vee cf$ in all cells but the least significant one, where ct and cf are the pair of bits implementing the carry-in. The nano-layouts for the carry and sum functions are shown in Figure 12 and Figure 13. The production rules describing the pullups and pulldowns of the sum and carry cells are as follows. For the sum:

$$\begin{array}{lcl}
ct \wedge (a = b) \vee cf \wedge (a \neq b) & \rightarrow & st\_\downarrow \\
cf \wedge (a = b) \vee ct \wedge (a \neq b) & \rightarrow & sf\_\downarrow \\
\neg ct \wedge \neg cf \vee \neg sf\_ & \rightarrow & st\_\uparrow \\
\neg ct \wedge \neg cf \vee \neg st\_ & \rightarrow & sf\_\uparrow
\end{array}$$

For the carry-out:

$$\begin{array}{lcl}
cf \wedge at \wedge bt \vee ct \wedge at \vee ct \wedge bt & \rightarrow & dt\_\downarrow \\
ct \wedge af \wedge bf \vee cf \wedge af \vee cf \wedge bf & \rightarrow & df\_\downarrow \\
\neg ct \wedge \neg cf \vee \neg df\_ & \rightarrow & dt\_\uparrow \\
\neg cf \wedge \neg ct \vee \neg dt\_ & \rightarrow & df\_\uparrow
\end{array}$$

6. Isochronic Fork

Once all QDI building blocks have been defined, the next step is to compose them into a working system. We leave aside the issue of assembling 1 into a layout and concentrate on the critical issue of isochronic forks: Given the wide timing variations to be expected in this technology and in nano-CMOS, can we satisfy the timing requirements of isochronic forks? Although the Async reader should be familiar with the concept of isochronic fork, there still exists enough misunderstanding about the nature of the isochronicity requirement that a discussion is in order. In the past, a simple-to-explain timing condition has often

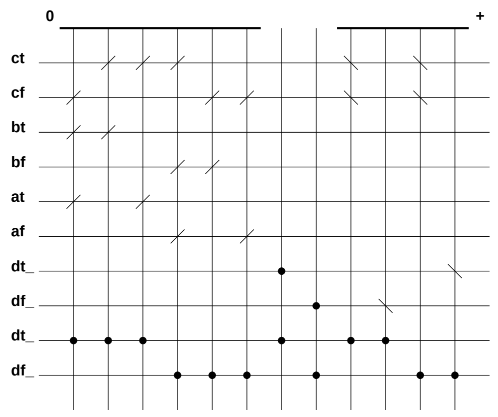

Figure 12. Carry-out computation

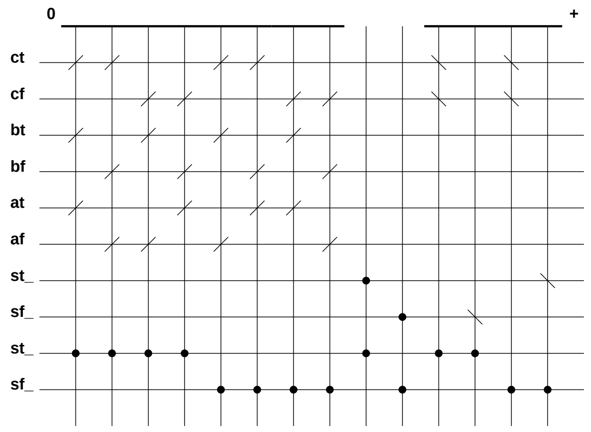

Figure 13. Sum computation

been used to implement the isochronic fork; but, in the presence of important timing variations, this sufficient timing condition is difficult to implement as it relies on a two-sided timing inequality. The designer should switch to a more complex but easier to satisfy necessary and sufficient condition. This condition relies on the notion of *adversary path*. It is a one-sided inequality stating that a single transition should be shorter in time than a sequence of multiple transitions.

Let us first review how the need for isochronic forks arises. A computation implements a partial order of transitions. In the absence of timing assumptions, this partial order is based on a causality relation. For example, transition $x\uparrow$ causes transition $y\downarrow$ in state S if and only if $x\uparrow$ makes guard By of $y\downarrow$ true in S. Transition $y\downarrow$ is said to *acknowledge* transition $x\uparrow$. We do not have to be more specific about the precise ordering in time of transitions $x\uparrow$ and $y\downarrow$. The acknowledgment relation is enough to introduce the desired partial order among transitions, and to conclude that $x\uparrow$ precedes $y\downarrow$. In an implementation of the circuit, gate Gx with output x is directly connected to gate Gy with output y, i.e., x is an input of Gy.

Hence, a necessary condition for an asynchronous circuit to be delay-insensitive is that all transitions are acknowledged.

Unfortunately, the class of computations in which all transitions are acknowledged is very limited, as we proved in [8]. Consider the following example in which the specification of the computation requires an ordering of transitions on variables x, y, and z defined by the sequence:

$$\ldots\ x\uparrow;\ \ y\uparrow;\ \ \ldots; x\uparrow;\ \ z\uparrow\ldots$$

Implementing this sequence of transitions requires introducing at least one control variable c to distinguish the states in which $x\uparrow$ causes $y\uparrow$ from the states in which $x\uparrow$ causes $z\uparrow$, leading to PRs of the form:

$$\begin{array}{ll} c \wedge x & \rightarrow\ y\uparrow \\ \neg c \wedge x & \rightarrow\ z\uparrow \end{array}$$

As shown in Figure 14, x as the output of gate Gx is forked to $x1$, an input of gate Gy with output y, and to $x2$, an input of gate Gz with output z. A transition $x\uparrow$ when c holds is followed by a transition $y\uparrow$, but not by a transition $z\uparrow$, i.e. transition $x1\uparrow$ is acknowledged but transition $x2\uparrow$ is not, and vice versa when $\neg c$ holds. Hence, in either case, a transition on one output of the fork is not acknowledged. In order to guarantee that the unacknowledged transition completes without violating the specified order, a timing assumption called the *isochronicity assumption* has to be introduced, and the forks that require that assumption are called *isochronic forks*[8]. The branch of the fork with the non-acknowledged transition is called an *isochronic branch.* (Not all forks in a QDI circuit are isochronic.)

6.1. The Isochronicity Assumption

First, it is important to realize that the weakest timing assumption associated with an isochronic fork is not a relation between the delays on the different branches of the fork—which would be a very tight assumption indeed. For example, in the case of a fork with two branches, if $\delta(t1)$ and $\delta(t2)$ are the delays of transition $t1$ on one branch and transition $t2$ on the other branch, the timing assumption is NOT that $|\delta(t1) - \delta(t2)| \leq \epsilon$. As already mentioned, such a timing requirement has been used because of its simplicity. It is sufficient but not necessary, and is very difficult to fulfill in the presence of large parameter variations, in particular threshold voltages.

In order to characterize a weaker timing condition for an isochronic fork to behave properly, let us examine how a circuit will fail when a transition on an isochronic branch fails to complete. In our previous example, assume that $x1\uparrow$

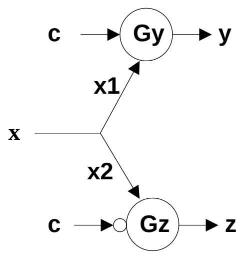

Figure 14. The fork (x,x1,x2) is isochronic: a transition on x1 causes a transition on y only when c is true, and a transition on x2 causes a transition on z only when c is false. Hence, certain transitions on x1 and on x2 are not acknowledged, and therefore a timing assumption must be used to guarantee the proper completion of those unacknowledged transitions.

causes $y\downarrow$, but $x2\uparrow$ does not complete. Since $x2\uparrow$ does not change the value of z, initially the failure of $x2\uparrow$ to complete does not cause a failure of gate Gz. But $y\downarrow$ causes a sequence of transitions that will eventually change another input of Gz, say transition t since we do not know the sense of this transition. At this point, transition $x2\uparrow$ not having completed will cause a misfiring of Gz, since in the partial-order specification of the circuit, $x2\uparrow$ should have completed.

Hence, the isochronicity assumption is that transition $x2\uparrow$ completes before the sequence of transitions starting at $x1\uparrow$ and ending at t. This sequence of transitions defines a path in the circuit called the *adversary path* of isochronic branch $x2$. The isochronicity assumption states that the delay $\delta(x2\uparrow)$ for unacknowledged transition $x2\uparrow$ to complete be always smaller than the sum of the delays of the transitions on the adversary path (including acknowledged transition $x1\uparrow$).

This is a one-sided inequality that can always be satisfied by making the adversary path longer.

6.2. Transition Delays

Estimating transition delays can be tricky. Transition delay has essentially two components: the diffusion delay of a voltage level propagating along a wire, and the switching delay which is the time for a voltage change to cut or tie a transistor chain connecting the output node of an operator to Vdd or ground. (In our implementation of QDI, an input of a logical operator is always the gate of one or several transistors.) The switching delay is going to be different depending whether the transition is a "cut" or "tie": To cut an n-type transistor chain requires a voltage change from Vdd

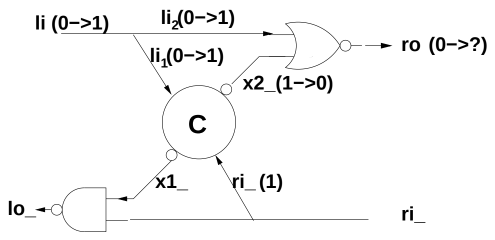

Figure 15. The fork (li,li1,li2) is isochronic, with li2 as isochronic branch for the transition 0 to 1 on li

down to below the threshold voltage Vtn, almost a full voltage swing. To tie an n-type transistor chain requires a voltage change from zero to just Vtn. And similarly for p-type transistors. Obviously, in the presence of long slew rates (the "slope" of the voltage change) a cut may take much longer than a tie. The isochronicity assumption can be of two types: "tie-before-tie" or "cut-before-tie," the first one being easier to satisfy than the second one.

For instance, the isochronic fork in the control/data pipeline of Figure 11 is of the type "tie-before-tie." It is required that the valid data $x1$ and $x2$ tie the transistors in $F1$ and $F2$ before the adversary paths tie the enable signal through ro. On the other hand, the isochronic fork in the PCHB (precharge half-buffer) which is the main template in the MiniMIPS (see [7, 9]), is of the type "cut-before-tie." The data inputs to the function block have to return to neutral (low voltage) hence cutting the pulldown chain of the function block, before the neutrality is detected and the enable signal is raised (a tie).

6.3 A Worst-case Example

In the proposed layout for molecular nano, the restricted wire layout and highly resistive connections make the implementation of the isochronicity assumption somewhat easier. Let us look at a worst-case example, where the adversary path contains just one gate. The example is the Caltech Q-element or the active-active buffer (see for instance [7]) shown in Figure 15. We analyze the isochronic fork $(li, li1, li2)$. Initially, $x2\_$ is true and ro is false. In order to satisfy the handshake specification, transition $li\uparrow$ should not change the value of ro. But since the inverted output of the C-element is the input $x2\_$ of the nor-gate the adversary path ($li1$, C-element, $x2\_$) causes transition $x2\_\downarrow$, which could cause transition $ro\uparrow$, unless the isochronic-branch transition $li2\uparrow$ terminates before $x2\_\downarrow$. This is one of the tightest isochronicity requirements since the adversary path contains only one gate, the C-element. (Since the C-element is used with an inverted output in this cell, in CMOS the output could be taken before the inverter if the

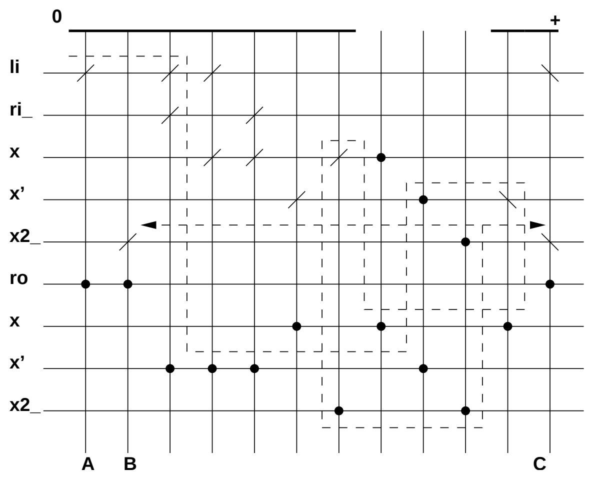

Figure 16. Partial layout showing the adversary path in dotted line

C-element is implemented with a staticizer. However, in an implementation of the C-element as a majority gate, the inverted output has to be taken after two inverters, making the adversary path safer.)

The nano-layout sketch of Figure 16 shows the implementation of the C-element with the two inverters, and the nor-gate. (The nand-gate is omitted.) The isochronicity requirement translates as follows. Initially the voltage on $x2_{-}$ is high and there is a path from GND to ro along the pulldown B. When the voltage on li rises, the pulldown A from GND to ro must be tied by the n-transistor with gate li and the pullup C must be cut by the p-transistor with gate li before the adversary path (marked by a dotted line on the figure) cuts pulldown B by the n-transistor with gate $x2_{-}$ and ties pullup C by the p-transistor with gate $x2_{-}$.

Keeping the isochronic-branch delay short requires that the layout discipline does not add extra delay through long and/or resistive wires or by connection resistances. The layout rule that places the n-plane and p-plane side by side allows the same metal wire to be used for all branches of the same isochronic fork provided it all fits within the length constraint of a metal nano-wire. This layout rule eliminates all extra delays on the isochronic branch besides the transition delay. Furthermore, the transformation that eliminates the staticizer helps make all transition delays of similar length by pushing the transition voltage towards $Vdd/2$.

With this precaution, the timing requirement for this example is that a single transition delay (a tie) on a single metal wire of the isochronic branch must be shorter than 3 transition delays plus 9 RC delays of contacts and wires for the adversary path. This timing requirement should not be difficult to fulfill.

7 Oscillating Rings

An important point concerning the implementability of QDI is to make sure that the rings of restoring gates composing a QDI system keep oscillating, since a QDI system is nothing but an interlocking of oscillating rings.

Since hardware computations are non-terminating, each transition $z\uparrow$ is followed, after a number of other transitions, by transition $z\downarrow$, and vice versa. Since the guards Bu and Bd of those transitions are mutually exclusive, the chain of transitions between $z\uparrow$ and $z\downarrow$ must contain a transition that invalidates Bu. Hence, transition $z\uparrow$ invalidates itself through a sequence of intermediate transitions. Can we still say that Bu is stable? Is it possible that the effect of $z\uparrow$ propagates through the cycle of gates fast enough to invalidate itself? At the electrical level, could the voltage $V(z)$ stabilize at an intermediate value close to $Vdd/2$?

Arguing that such a ring of operators is not self-invalidating is equivalent to arguing that the ring oscillates. This is an electrical property of the circuit that relates the slew rates of transitions, the gain of the operators, and the number of operators on the ring. We will have to require that any cycle of operators be implemented with a number of stages at least equal to a chosen minimum to guarantee that the cycle is not self-invalidating. In CMOS, three restoring operators with good gain are usually sufficient, although we usually require five to be safe. In molecular nano-technology, the minimal number of operators on each ring will have to be determined based on the above-mentioned electrical properties of the technology.

8 Conclusion

The main purpose of this paper was to establish the existence of an efficient QDI logic family given the strict design rules and large parameter variations of our target nano-technology. A contribution of the research has been to show that there exist efficient static implementations of state-holding elements without weak feedback and with no more than two transistors in series in the pullups. Such an approach will be of interest for “extreme nano-CMOS” as well. Also, the isochronicity assumption has been further refined.

Several issues have not been addressed: (1) As up to 10% of the devices will be unusable (broken wire, stuck open contact, etc) defect tolerance is an integral part of the design. We have not presented any solution for defect tolerance, although preliminary investigation indicates that satisfactory strategies exist. (2) For the usual density reasons, memory requires specialized design. (Small memories have already been demonstrated.) (3) Input and output, as well as power supply, will be done at a microlevel, e.g. CMOS.

How to connect nanowires to the microlevel CMOS connections without increasing the nano-pitch is a separate issue. Several solutions exist. (See [4, 1].)

Acknowledgments

Acknowledgment is due to my colleagues Jim Heath and André DeHon for sharing some of their knowledge about molecular nanotechnology, and to my students Sean Keller and Chris Moore for their excellent comments and criticisms. The research described in this paper was supported by a grant from the National Science Foundation.

References

[1] Greg Snider, Philip Kuekes and R Stanley Williams. CMOS-like logic in defective, nanoscale crossbars. *Nanotechnology*, **15** 881-891, 2004.

[2] M.R Stan *et al.* Molecular Electronics: From Devices and Interconnect to Circuits and Architecture. *Proceedings of the IEEE*, 91, 11, 1940-1957, 2003.

[3] Andre DeHon. Nanowire-Based Programmable Architectures. *J. of Emerging Technologies in Computer Systems*, 2005.

[4] Andre DeHon and Helia Naeimi. Seven strategies for tolerating highly defective fabrication. *IEEE Design and Test of Computers*, 22(4), 2005.

[5] C.M.Lieber *et al.* Design and hierarchical assembly of nanowire-based moletronics. DARPA Moletronics PI Meeting. 2002.

[6] N.A.Melosh *et al.* Ultrahigh-density nanowire lattices and circuits. *Science*, **300**, 112-115, 2003.

[7] A.J. Martin, M. Nyström. Asynchronous Techniques for System-on-Chip Design. *Proceedings of the IEEE*, Special Issue on Systems-on-Chip, 94, 6, 1089-1120. 2006.

[8] Alain J. Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. *Sixth MIT Conf. on ARVLSI, ed. W.J. Dally*, 263-278, MIT Press, 1990.

[9] Alain J. Martin *et al.* The Design of an Asynchronous MIPS R3000 Microprocessor. *Proc. 17th Conf. on ARVLSI*. Los Alamitos, Calif.: IEEE Computer Society Press, pp. 164–181, 1997.