Derivation of an Asynchronous Counter

with 105ps/bit load time and early completion in 90nm CMOS

Adam Megacz

July 17, 2009

Abstract

This draft memo describes the process by which I methodically derived a design for a 4/2 GasP Kessels counter. Once settled, the counter can decrement at full GasP speed. The time required to settle is data-dependent, but in the worst case is no more than 104ps/bit with 200λ of wire capacitance on each state wire.

Since settling is performed using the reverse GasP path, the counter should – in theory – settle at around 50ps/bit. It is unclear why this performance has not been achieved.

Schematics are complete and borrowing (but not decrementing) has been verified. Layout and decrement testing will be completed soon.

1 The Counter

A counter consists of a sequence of *bits*. Each bit has four possible *states*: `Zero`, `One`, `Two`, and `Done`. The state of the i^{th} bit will be denoted by s_i, where the least significant bit is the 0^{th} bit. Bits without physical storage are considered to be in the `Done` state.

The *value* of the i^{th} bit is denoted by v_i and defined to be:

$$v_i = \begin{cases} 0 \text{ if } s_i = \texttt{Done} \\ 0 \text{ if } s_i = \texttt{Zero} \\ 1 \text{ if } s_i = \texttt{One} \\ 2 \text{ if } s_i = \texttt{Two} \end{cases}$$

The *value* of the entire counter is denoted by V and defined to be:

$$V = \sum_{i=0}^{\omega} 2^i v_i$$

Two bits are adjacent if their indices differ by 1. A *GasP module* appears between each pair of adjacent bits. Two GasP modules are adjacent if they share a bit.

A GasP module is *enabled* when its *enabling condition* is met. The enabling condition for the counter's GasP modules is:

(E1) The module's *less significant neighbor* (LSN) is in the `Zero` state and its *more significant neighbor* (MSN) is *not* in the `Zero` state.

A finite but unbounded amount of time after becoming enabled a GasP module will *fire*. When a GasP module fires, it modifies the states of the adjacent bits according to its *firing actions*. The firing actions of each GasP module in the counter depend on the state of its MSN:

(A1) If MSN=`Done`, set LSN`:=Done`

(A2) If MSN=`One`, set LSN`:=Two` and set MSN`:=Zero`

(A3) If MSN=`Two`, set LSN`:=Two` and set MSN`:=One`

1.1 Requirements

The key to understanding the counter algorithm is observing that the enabling condition and firing actions satisfy the following five requirements:

(R1) There is no state in which two adjacent GasP modules are both enabled. [1]

(R2) Once enabled, a GasP module's enabling condition will remain true until it fires, and any states upon which its firing action depends will not change.

(R3) The firing actions of a GasP module always cause its enabling condition to become false.

(R4) The firing actions of a GasP module modify the states of the adjacent bits, but those modifications do not result in a net change to the value V of the entire counter.

(R5) For any given k, if $(\forall i{>}k)s_i{=}$`Done` before firing, that property will also hold after firing. [2]

Requirements (R1)-(R3) apply to all GasP circuits. Requirements (R4) and (R5) are specific to the counter.

Observe that requirement (R1) holds: if a given GasP module is enabled then its LSN is `Zero`, so the GasP module which shares that bit sees that its MSN is `Zero` and is therefore not enabled. Moreover, the enabled module's MSN is not `Zero`, so the GasP module which shares that bit sees its LSN is not `Zero` and is therefore not enabled.

Requirement (R2) follows from the fact that the state of the bits adjacent to a given GasP module change only when the GasP modules adjacent to it fire, which happens only when those modules are enabled, which by (R1) cannot be the case when the original module was enabled.

Requirement (R3) holds because in all three cases the firing actions set the LSN to a non-`Zero` state.

Requirement (R4) holds for each of the three possible firing actions:

(A1): $(2^{i+1})2 + (2^i)0 = 2^{i+2} = 2(2^{i+1}) = (2^{i+1}) + (2^{i+1}) = (2^{i+1})1 + (2^i)2$

(A2): $(2^{i+1})1 + (2^i)0 = 2^{i+1} = (2^{i+1})0 + (2^i)2$

(A3): $(2^{i+1})0 + (2^i)0 = 0 = (2^{i+1})0 + (2^i)0$

Requirement (R5) holds because whenever the MSN is `Done` before firing it remains `Done` after firing (proof: induction on the bit number).

[1] Requirement (R1) can be weakened slightly to the following: "if there is a state in which two GasP modules are both enabled, none of the firing actions taken by either module invalidates the enabling condition of the other, nor does it change any element of the state on which the firing action taken by the other depends." In practice this weakening is seldom helpful.

[2] If we assume that physical storage is allocated only to some initial segment of the bits, this requirement ensures that the amount of physical storage will never increase.

2 Encodings

We now consider the problem of how to encode the four bit states using GasP *state wires*. Each state wire has two states, *empty* and *full*. The verbs *to fill* and *to drain* are used for the transitions between these wire states.

The encoding of a single bit state as a bundle of wire states is subject to both *correctness* and *performance* considerations.

2.1 Correctness Considerations

Informally, whenever a firing action changes a bit from one state to another, it is important than these states are encoded such that any state which the bit "passes through" on the way does not cause any other GasP module to become enabled. An exception to this rule is allowed if the module would be enabled in the final bit state as well and the action it takes is the same in both bit states.

More formally, if we have bit states `X`, `Y`, and `Z` such that the Hamming distances $H(\texttt{X},\texttt{Y})+H(\texttt{Y},\texttt{Z})=H(\texttt{X},\texttt{Z})$, then a firing action may change from bit state `X` to bit state `Z` only if any adjacent module which is enabled in bit state `Y` takes the same action in that state as in bit state `Z`.

2.2 Performance Considerations

GasP is fastest in the "reverse" direction; that is, emptiness propagates faster than fullness. Indeed, this is the only distinction between the two wire conditions; the actual voltage levels used for full and empty may be changed independently of encoding considerations[3].

Therefore, the time between two firings is minimized by choosing encodings such that the second firing is enabled in some bit state `Y`, and the first event causes that firing by changes from bit `X`, where every wire which is full in bit state `Y` is also full in bit state `X` (but not vice versa). Less formally, transitions are fast when they involve only *draining* and no *filling*.

2.3 State Encoding for the Counter

The four bit states will be encoded using two state wires, which is the minimum number that suffice. Although there are 24 possibilities for the encoding, the correctness conditions require consideration of only the Hamming distance between states, which reduces the number of cases to consider to three.

The correctness conditions require that states `Zero` and `Two` not be separated by a Hamming distance of 2; if this were the case, then the firing action which changes the LSN from `Zero` to `Two` would cause it to pass through the state `One` "on the way," which could enable an adjacent GasP module. Moreover, the adjacent GasP

[3] and indeed are usually chosen based on NMOS/PMOS asymmetry.

module takes a different action on a `One` than a `Two`: (A2) vs (A3). The same argument applies to states `Zero` and `Done`. Therefore, `Zero` must be Hamming-adjacent to both `Two` and `Done`. This requirement completely determines the Hamming distances of the state encodings.

The most time-critical operation in the counter is the "borrow" that occurs immediately after loading the counter; all other operations require no more than a single GasP cycle, and so are at least as fast as the environment which is performing decrement requests. Therefore, we will choose the state encoding such that the operation which changes the LSN from a `Zero` state to a non-`Zero` state is as fast as possible.

There are two such transitions: `Zero`-to-`Done` caused by (A1) and `Zero`-to-`Two` caused by (A2) or (A3). Therefore we can encode `Zero` as both state wires full, and each of `Done` and `Two` as having one wire empty (a different wire for each state). Quite fortunately, this encoding satisfies the correctness conditions, since `Zero` is Hamming-adjacent to both `Two` and `Done`.

Together, these two conclusions completely determine how the four states are encoded as GasP state wires, up to renaming of the two wires:

State	Wire A	Wire B
`Done`	empty	full
`Zero`	full	full
`Two`	full	empty
`One`	empty	empty

Adjacent rows in the table correspond to states with Hamming-adjacent codes; additionally the top and bottom rows have Hamming-adjacent codes. We will call the two state wires "`ZeroOrTwo`" and "`ZeroOrDone`" after the states in which each wire is full.

3 Circuit Design

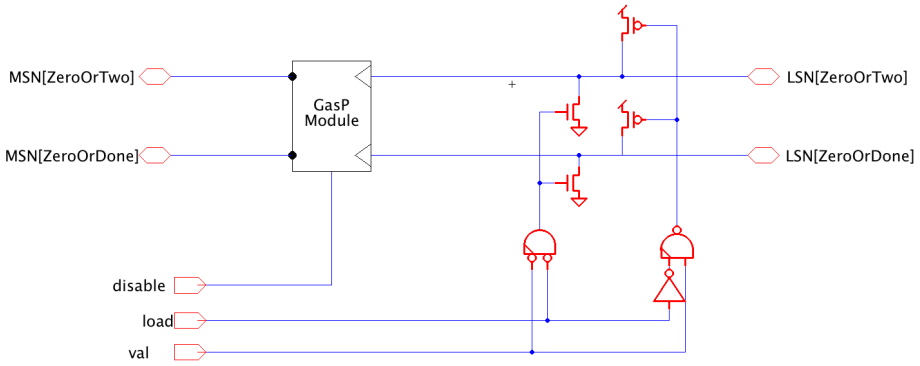
Having chosen the enabling condition, firing actions, and state encodings, we are now ready to start designing circuits.

3.1 One Bit

The figure below shows the top level cell for a single bit of the counter. This includes two state wire connections to the more significant neighbor (MSN) labeled `MSN[ZeroOrTwo]` and `MSN[ZeroOrDone]` and two state wire connections to the less significant neighbor, labeled `LSN[ZeroOrTwo]` and `LSN[ZeroOrDone]`. Three connections are provided to the environment:

- `val` carries the value to load into this bit of the counter.
- `load` causes the value presented on `val` to be loaded into this bit of the counter.

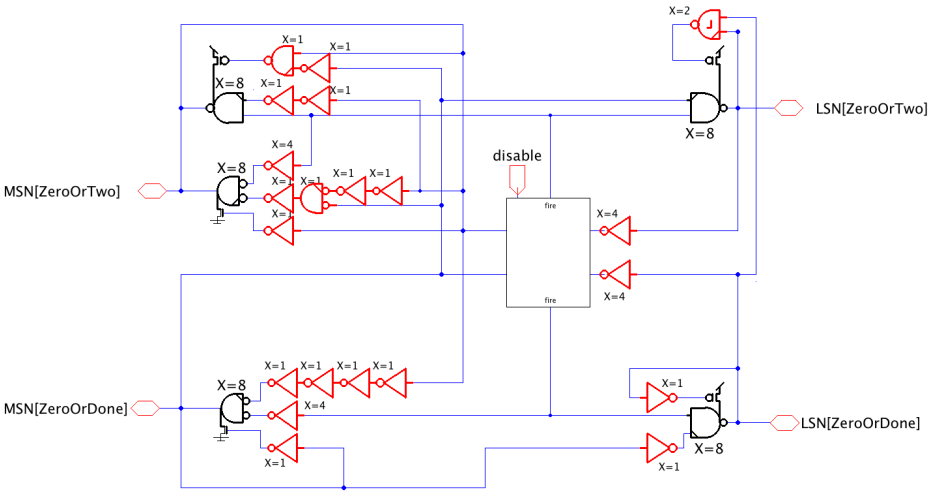
- `disable` causes the firing of the GasP module to be disabled. This signal must be asserted two gate delays before `load` and must remain asserted until two gate delays after `load` is de-asserted.



3.2 The GasP Module

The diagram below shows the circuitry for a single GasP module; the large cell in the center is the firing circuit described in the final part of this section.

Note that this diagram includes many redundant gates in order to improve readability; for example, in an actual layout the two inverters whose inputs are driven by `LSN[ZeroOrDone]` would be combined into a single gate.



The five gates in this diagram which drive the state wires are ordinary four-transistor NAND/NOR gates with a fifth transistor which can disconnect the ap-

propriate half of the gate from the power rail.

The "active" behavior each of these five state-manipulating gates can be described as follows, and are a direct implementation of the firing actions (A1)-(A3):

- The `LSN[ZeroOrTwo]` state wire is emptied (pulled low) when the GasP module fires and the `MSN[ZeroOrDone]` wire is full at firing time.
- The `LSN[ZeroOrDone]` state wire is emptied (pulled low) when the GasP module fires and the `MSN[ZeroOrDone]` wire is empty at firing time.
- The `MSN[ZeroOrDone]` state wire is filled (pulled high) when the GasP module fires and the `MSN[ZeroOrTwo]` wire is empty at firing time.
- The `MSN[ZeroOrTwo]` state wire is filled (pulled high) when the GasP module fires and both `MSN` wires are full at firing time.
- The `MSN[ZeroOrTwo]` state wire is emptied (pulled low) when the GasP module fires and the `MSN[ZeroOrTwo]` wire is full at firing time.

3.2.1 Delays

Note that whenever the state of a wire from a given state (`LSN` or `MSN`) is used to decide whether to drain or fill a wire *in that same stage*, the state used to make the decision must be delayed by 2-4 gate delays. This ensures that the result of a firing action is not observed until after the `fire` pulse has subsided.

3.2.2 Keepers

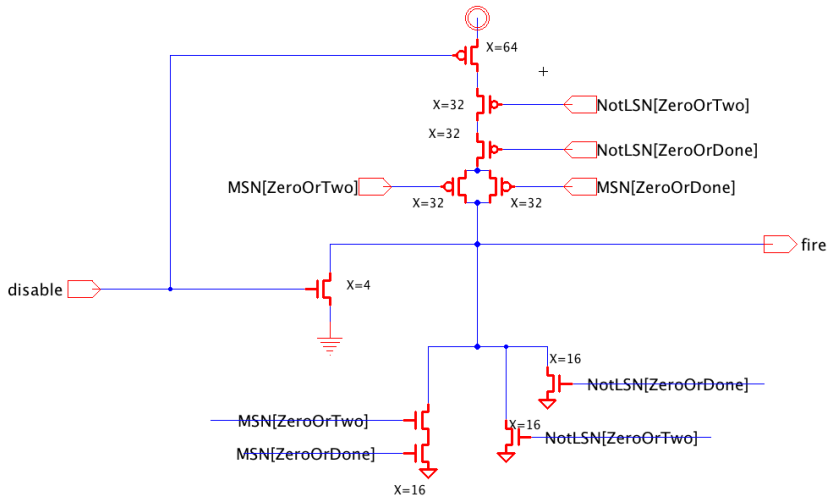
In addition to the behaviors which actively drive the state wires to particular values, the five state wire drivers also include a *keeper* action or "passive" behavior. This is most easily understood by analogy to a game of catch between two people: whoever catches the ball is responsible for holding it until they decide to throw it again. Likewise, whichever side of the wire decides to raise it must first watch for it to be lowered and *hold it low* until that time comes. The fact that the gate which drives a wire *high* bears the responsibility of holding it *low* can be counterintuitive at first.

The situation becomes more complex when a particular GasP module both raises and lowers a given state wire, as happens with `MSN[ZeroOrTwo]`. In this case, the holding condition involves watching not just the wire being held, *but also the other wires whose states play a part in the decision to raise it.*

3.3 The Firing Circuit

In 4/2 GasP the `fire` signal is a high-going pulse driven by a single stage of logic. This single stage of of logic computes the enabling condition. In this case, that condition (E1) is equivalent to "enable when either `MSN[ZeroOrDone]` or `MSN[ZeroOrTwo]` is empty and both of `LSN[ZeroOrDone]` and `LSN[ZeroOrTwo]` are full."

A circuit which computes this condition is shown below:



Note the additional `disable` signal, which can be used to suspend the operation of the firing circuit. This will be used during loading of the counter.

We expect that in the case where performance matters the most, it will be the fall of one of the MSN signals (usually `MSN[ZeroOrDone]`) which triggers the firing; therefore we position the two PMOS transistors driven by the MSN signals farthest from the rail (closest to the output).

4 Results

A SPICE netlist was extracted from the schematics in Electric, including transistor sizes. This netlist was then simulated using Synopsys Nanosim and the TSMC 90nm device libraries.

Each of the plots below shows the behavior of a six bit counter using twelve graphs. The first graph shows the `fire` signal of the *most significant GasP module* (the module whose MSN is fixed at `Done`). The next graph after that shows two waveforms, one for each of the state wires on the *less significant side* of the GasP module from the previous graph. The remaining ten graphs each repeat this pattern, in descending order of bit significance.

When printed in color, the `fire` signals appear in red, the `ZeroOrTwo` signals appear in blue, and the `ZeroOrDone` signals appear in green.

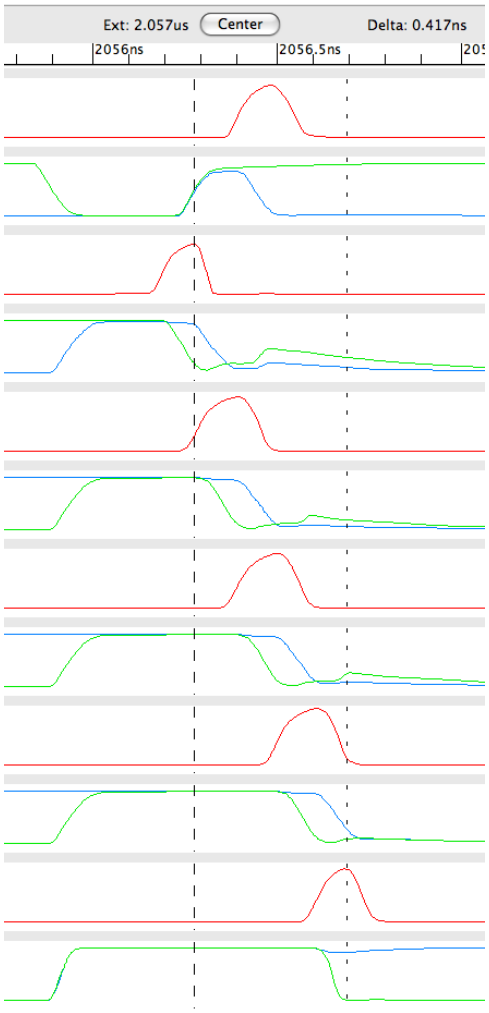
4.1 Plot 1

Plot 1 shows the counter with a starting value of 32, or `6'b100000`.

The two vertical dashed lines mark the distance in time between the firing of the fifth and first GasP modules. The time interval between these firings is 417ps, giving a borrow rate of 105ps per bit.

The sequence of states is approximately:

```
D100000
D020000
DD20000
DD12000
DD11200
DD11120
DD11112
```

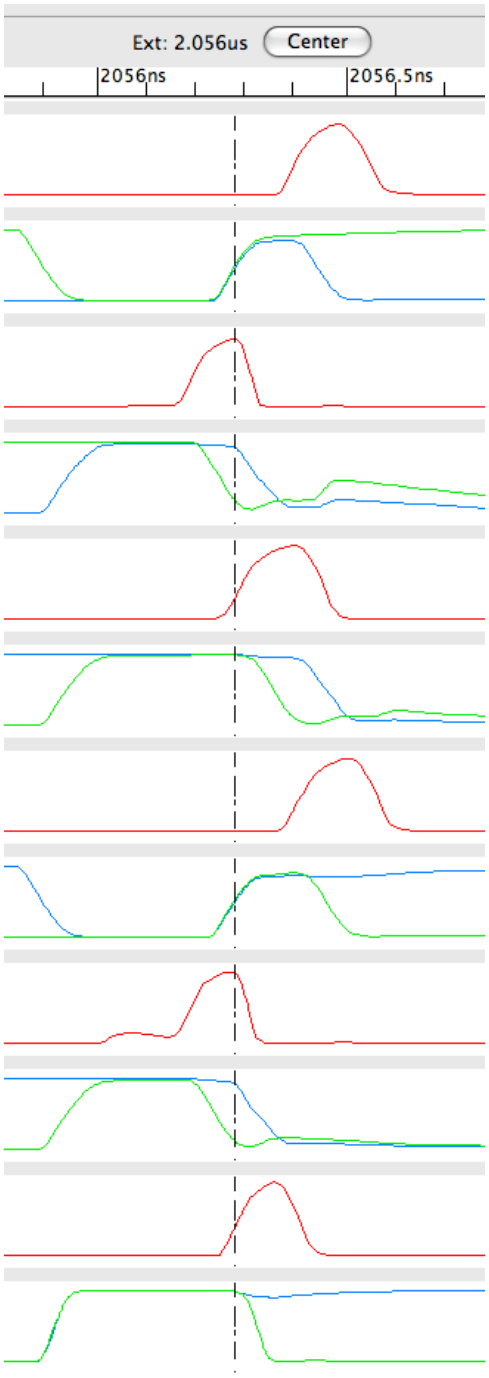


4.2 Plot 2

Plot 2 shows the counter with a starting value of 36, or `6'b100100`. Note the concurrency in the borrowing; the fifth bit borrows from the sixth bit simultaneously with the second bit borrowing from the third.

The sequence of states is approximately:

```
D100100
D020020
DD12012
DD11212
```



4.3 Plot 3

Plot 3 shows the counter with a starting value of 16, or `6'b010000`. Because the imaginary "seventh bit" holds a fixed value of `Done`, the sixth bit will fail to borrow from it (and become `Done`) at approximately the same time that the fifth bit is borrowed from and becomes zero. This means that the GasP module between the fifth and sixth bits sees the two halves of its enabling condition occur at the same time.

The sequence of states is approximately:

```
D010000
DD02000
DDD1200
DDD1120
DDD1112
```