Constant Time Arbitrary Length Synchronous Binary Counters

J. E. Vuillemin
Digital Equipment Corp.
Paris Research Laboratory
85 Av. Victor Hugo
92500 Rueil-Malmaison, France

Abstract

We introduce a synchronous binary counter which can be operated under a high clock frequency, independent of the counter's length n: all signals traverse at most two 3-inputs logic gates during each clock phase. The proposed design is simple enough to have practical implications, as illustrated by a cMOS programmable gate array implementation which has counted up to 2^{40} with a 40MHz clock. The area required for laying out our design is no larger than that of the (much slower) carry-ripple counter.

1 Introduction

Binary counters are found in nearly all digital systems, and their design is extensively covered in all hardware courses and textbooks.

- Since their implementation is so simple, binary counters are seldom in the *critical path* of a synchronous digital design. Whatever device the counter is controlling is typically more complex, hence slower than our small friend.
- Even in technologies where a synchronous counter can be operated with a 1GHz (1ns period) clock, one can only count up to about 2^{50} during the course of one day. Practical uses for longer counters are thus doubtful.

We have nevertheless two reasons for being interested in fast counters.

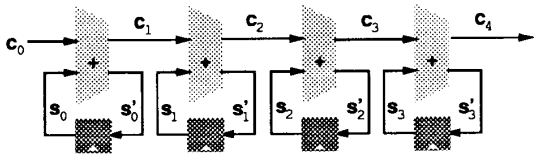
1. Counters provide the most basic mean for testing, debugging and measuring digital systems. In order for such an instrumentation to be effective, it needs to operate *faster* than the system under observation. In this context, counters thus naturally need to be the *fastest possible designs* in any given technology.
2. The theory of binary arithmetic circuits has reached a very clean conceptual state: time $log_2(n)$ is necessary and sufficient for both n bits addition ([2],[7],[4]) and multiplication ([6],[8],[5]).

   While the $log_2(n)$ lower bound of [7] also applies to circuits performing combinatorial incrementation, it breaks down for *synchronous* counters. This is shown by [3], who construct a synchronous binary counter whose clock period is independent of the counter size, thus showing that incrementation can be performed substantially faster than addition.
3. The counter presented here is similar in nature to that of [3]; it is, however, somewhat simpler, using half as many flip-flops, with the same clock speed. In a comparative 64 bits test implementation (see below), our counter's area is twice smaller than that of [3].

We exclude from our discussion carry-save, pipelined and asynchronous counters, which share some, but not all features in our design. Our timing model is the simple gate-depth count, with limited fan-in. It ignores fan-out and far-away signal distribution problems, which have to be delt with specifically for each implementation technology.

2 Linear time binary counters

Let us review some classical designs, starting with the *carry ripple counter*:



4 bits carry ripple counter.

The counter's only input (besides the clock) is the *increment* signal c_0. Each sum bit $S = {}_2[s_0 s_1 \cdots] = \sum_{i \geq 0} s_i 2^i$ is the output $s_i = \rho_t(s'_i)$ of a synchronous register whose input is the corresponding bit of the next sum $S' = {}_2[s'_0 s'_1 \cdots] = \sum_{i \geq 0} s'_i 2^i = S + c_0$. Each bit of S' is the *exclusive or* $s'_i = s_i \oplus c_i = s_i + c_i - 2s_i \times c_i$

of the corresponding sum and carry bits. Each consecutive carry $c_{i+1} = s_i \wedge c_i = s_i \times c_i$ is the *logical and* between the previous carry and sum bit. The carry ripple counter is therefore completely specified by the following equations:

$$s_i = \rho_t(s'_i), \quad s'_i = s_i \oplus c_i, \quad c_{i+1} = s_i \wedge c_i \text{ for } i \geq 0.$$

Let $c_0^{(0)} c_0^{(1)} \cdots c_0^{(t)} \cdots$ denote the boolean values of the counter's increment at times $0\,1 \cdots t \cdots$; the corresponding counter's values $S^{(0)} S^{(1)} \ldots S^{(t)} \ldots$ sum up in binary the increments seen up to time t:

$$S^{(t)} = {}_2[s_0^{(t)} s_1^{(t)} \cdots] = \sum_{i \geq 0} s_i^{(t)} 2^i = \sum_{0 \leq k < t} c_0^{(k)}.$$

The value of S' at time t is that of S at time $t+1$, so $S'^{(t)} = S^{(t+1)} = S^{(t)} + c_0^{(t)}$. The carry vector $C^{(t)} = S^{(t)} \oplus S'^{(t)} = {}_2[c_0^{(t)} c_1^{(t)} \cdots] = \sum_{i \geq 0} c_i^{(t)} 2^i$ is the bit-wise exclusive or of S and S'. A small simulation will refresh our memory of the binary number system:

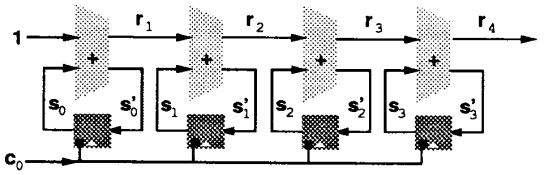
t	$s_0^{(t)}$	$s_1^{(t)}$	$s_2^{(t)}$	$s_3^{(t)}$	$c_0^{(t)}$	$c_1^{(t)}$	$c_2^{(t)}$	$c_3^{(t)}$	$c_4^{(t)}$
0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	1	1	0	0	0
2	0	1	0	0	1	0	0	0	0
3	1	1	0	0	1	1	1	0	0
4	0	0	1	0	1	0	0	0	0
5	1	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	0	0
7	1	1	1	0	1	1	1	1	0
8	0	0	0	1	1	0	0	0	0

Let us assume that any logic gate with up to 3 inputs has a *unit delay* τ. The minimal *clock period* under which we may correctly operate a n bit carry ripple counter (crc) is:

$$T_{crc}(n) \quad = \quad n \times \tau. \qquad (1)$$

Attempts to operate the *crc* under a lower clock period will fail: consider a time t when the counter's value is $S = 2^n - 1 = {}_2[11\cdots]$, with $c_0^{(t)} = 0$, so $C^{(t)} = 0 = {}_2[00\cdots]$. An increment $c_0^{(t+1)} = 1$ in that state causes all carries and sum bits to change; this requires a $n \times \tau$ combinatorial delay for the increment signal to ripple through up to the n-th bit.

Consider now the following *carry anticipate counter*:



4 bits carry anticipate counter.

It is derived from the carry ripple counter by setting the initial carry to 1, and by enabling the sum registers through the increment signal c_0. Its defining equations are thus:

$$\begin{aligned} s_i &= \rho_t(if\ c_0 = 1\ then\ s'_i\ else\ s_i), \\ s'_i &= s_i \oplus r_i, \text{ for } i \geq 0, \\ r_{i+1} &= s_i \wedge r_i, \text{ for } i \geq 0 \text{ with } r_0 = 1. \end{aligned}$$

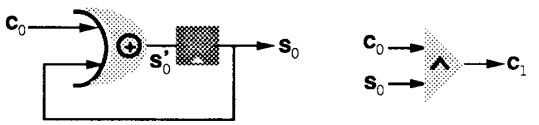
Using the same notations as before, we have: $S^{(t)} = \sum_{0 \leq k < t} c_0^{(k)}$, $S' = S+1$, $R = S \oplus S'$. Our interest for the carry anticipate counter (cas) is not immediate, since

$$T_{cac}(n) \quad = \quad n \times \tau, \qquad (2)$$

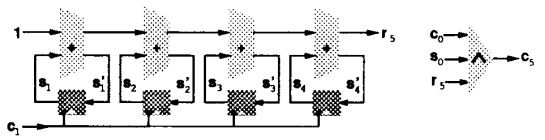
which gets established through the very same argument as (1). We can however operate a n bits carry anticipate counter at full clock speed $T_{crc} = \tau$ provided that we restrict the non-zero increments to only occur during clocks phases which are at least n cycles apart. Indeed in this case, two time instants t_1 and t_2, for which $c_0^{(t_1)} = c_0^{(t_2)} = 1$ and $c_0^{(t)} = 0$ for $t_1 < t < t_2$, are such that $t_2 - t_1 \geq n \times \tau$. It follows that the length $n-1$ carry chain $R^{(t_1)} = S^{(t_1)} \oplus (S^{(t_1)} + 1)$ reaches a *stable state* no later than time $t_2 - \tau$. Any subsequent non-zero increment $c_0^{(t)}$ at time $t \geq t_2$ will make the input multiplexer to each sum bit switch within delay τ after the clock pulse, so the correct value of S' will be latched. This observation, together with an (easy) analysis of the frequency of non-zero carries c_k is the key to designing a *constant time* binary counter.

3 Constant time binary counter

The constant time $T(n) = 2 \times \tau$ (independent of n) counter is organized as a sequence of carry anticipate counter blocks. In an actual implementation, choices for the blocks' lengths may be different from the ones given below. Our choices match the clock period $2 \times \tau$, under the assumption that any logic gate up to 3 inputs has internal delay less than τ. The first block is a one bit *crc*:



with defining equations: $s_0 = \rho_t(c_0 \oplus s_0)$, $c_1 = c_0 \wedge s_0$. The second block is a 4 bits carry anticipate counter:



with defining sum equations:

$$s_1 = \rho_t(if\ c_1 = 1\ then\ s_1 \oplus 1\ else\ s_1),$$

$$s_2 = \rho_t(if\ c_1 = 1\ then\ s_2 \oplus r_2\ else\ s_2),$$

$$s_3 = \rho_t(if\ c_1 = 1\ then\ s_3 \oplus r_3\ else\ s_3),$$

$$s_4 = \rho_t(if\ c_1 = 1\ then\ s_4 \oplus r_4\ else\ s_4).$$

The carry equations for this 4 bits *cac* block are:

$$r_2 = s_1,\ r_3 = r_2 \wedge s_2,\ r_4 = r_3 \wedge s_3,\quad r_5 = r_4 \wedge s_4,\ c_5 = r_5 \wedge s_0 \wedge c_0. \tag{3}$$

In order to verify that this 5 bits counter correctly operates within clock period $2 \times \tau$, consider a time t when the carry chain $R_{2,5} = {}_2[r_2 r_3 r_4 r_5]$ changes state, i.e. $s_1^{t-1} \neq s_1^t$. Since the value of s_1 changes at most every second clock tick, the earliest time t' when state $R_{2,5}$ may change again is such that $t' - t \geq 4 \times \tau$. The gate depth of equations (3) being less than $3 \times \tau$ for each r_i $(2 \leq i \leq 5)$, we see that r_2, r_3, r_4 and r_5 reach their stable value no later than time $t' - \tau$; it follows that ${}_2[s_1' s_2' s_3' s_4']$ has the correct value $1 + {}_2[s_1 s_2 s_3 s_4]$ no later than time t'. By the same reasoning carry c_5 has its correct value at all times.

For most practical purposes, we are through with our counter design. When properly adapted to the characteristics of a given technology, the first counter block will typically have $k \sim 3$ to 8 bits, and the second block more than 2^k bits, so there is no need for a third block. In order to implement such a counter in a given technology, one has to:

1. Derive from the technology parameters an *a-priori* estimate for gate, signal distribution and register delays, in order to determine a structure for the first k bits counter block. The logic gate computing the enabling carry $c_k = c_0 \wedge r_k$ to the next block must be carefully implemented so as to be *glitch-free.*

2. Design and optimize for the given technology the second level *cac* block.

3. Implement, test and measure the resulting counter. If measurement does not validates the initial assumptions, another design iteration is called for.

With help from Alan Skea, we have carried out this task for an existing $\tau = 10ns$ (100MHz) cMOS process, based upon Xilinx's programmable gate array [9] and our own PAM technology [BRV89]. The resulting counter, with a $k = 3$ bits first block, has been tested and measured. It runs at 40MHz (slightly under the theoretical 50MHz limit), for over 40 bits, which is as far as we could test it within a work-day. It has proved over 20 % faster than a similarly inspired counter, designed by Peter Alfke, which is part of the standard library in that technology [9].

For the sole benefit of our theoretically minded reader, let us pursue our counter's construction. The 3-rd block is a carry anticipate counter of length $64 = 2 \times 2^5$, with defining sum equations:

$$s_i = \rho_t(if\ c_5 = 1\ then\ s_i \oplus r_i\ else\ s_i) \text{ for } 5 \leq i \leq 64.$$

The corresponding carry equations are: $r_6 = s_6$, $r_{i+1} = r_i \wedge s_i$ for $5 < i \leq 64$. This 63 long carry chain has enough $(63 \times \tau)$ time to settle, since state changes in bits $s_5 s_6 \cdots$ of the counter are at least 32 cycles $(\geq 64\tau)$ apart. By now, the reader has presumably infered the general pattern for our counter. The 4-th block is an intimidating 2^{69} bits long *cac*[1], with enable signal

$$c_{69} = (c_0 \wedge s_0 \wedge (r_5 \wedge r_{69})).$$

An observer of gate c_{69} and r_{69} better be patient, since not much happens until clock ticks T_1 and T_2, determined by $\sum_{k<T_1} c_0^{(k)} = 2^{69} - 32$ and $\sum_{k<T_2} c_0^{(k)} = 2^{69}$. By clock tick $T_2 - 1$, enough cycles have gone by so r_{69} has settled to 1, no later than time $(2T_2 - 1)\tau$, and no earlier than time $2T_1\tau$. At this point, $c_0 = r_5 = r_{69} = 1$, but $s_0 = 0$ so $c_{69} = 0$. During the next clock cycle T_2, signals $c_0 = r_5 = r_{69} = 1$ keep their value 1, but our counter has just changed its parity $s_0 = 1$; so c_{69} becomes 1 no later than one gate delay τ after the clock tick; just in time to enable the multiplexer controlling the next value of bit s_{69}. Bit s_{69} is thus set to 1 at the next clock tick while all 69 previous bits *and* carries are reset to 0.

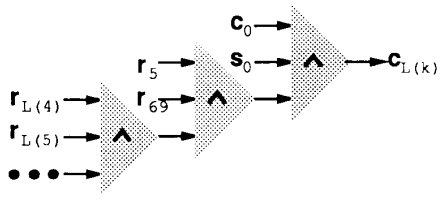
The 5-th block[2] is a *cac* of length $l(5) = 2^{T_2} - 1$. The carry enable $c_{L(4)}$, with $L(4) = T_2 = 2^{69}$ into this block gets computed by :

$$c_{L(4)} = (c_0 \wedge s_0 \wedge (r_5 \wedge r_{69} \wedge r_{L(4)})).$$

In general, the k-th *cac* block has length $l(k) = 2^{L(k)} - k/2$, where $L(k) = \sum_{i<k} l(i)$ is the sum of the lengths of the preceding blocks. The incoming carry to block k is:

$$c_{L(k)} = (c_0 \wedge s_0 \wedge (r_{L(2)} \wedge r_{L(3)}(\wedge r_{L(4)} \cdots \wedge r_{L(k)}) \cdots)).$$

In picture:



While the depth of such gates grows as $k/2$, this (tiny) delay gets absorbed by our choice $l(k) = 2^{L(k)} - k/2$ in the block length.

[1] over a billion Tera bits!

[2] by now, we have far exceeded the time and space limits of the known Universe!

4 Conclusion

What makes this design work is the very structure of the binary number system. Let the *2adic valuation* $v_2(n)$ of integer $n > 0$ be the exponent of the largest power of 2 which divides n, so $n = 2^{v_2(n)}(2p+1)$ for some natural number $p \geq 0$. The number of non-zero carries in the n-th line of the binary table is precisely $1 + v_2(n+1)$. The total number of carries $C(n)$ effectively propagated when we increment consecutive integers from 1 up to n is $C(n) = \sum_{0<i\leq n}(1 + v_2(n+1)) = 2n - \nu_1(n)$, where $\nu_1(n) = \sum_{i\geq 0} n_i$ is the number of non zero bits when one writes $n = {}_2[n_0 n_1 \cdots]$ in binary. It follows that $n < C(n) < 2n$, and we can say that the *average carry chain* in a counter has length (almost) 2. What our counter effectively does is to *amortize* throughout all cycles the bursts of long yet rare carry chains which plague the worst case behaviour of naive implementations. From the above analysis ($C(n)/n \sim 2$), we are tempted to conjecture that the clock period $T = 2 \times \tau$ cannot be reduced to $T = \tau$, under our limited 3 fan-in assumption.

To conclude, we leave the following as an open question: is it possible to design a synchronous, arbitrary length, constant time up-down[3] counter?

Acknowledgements

Thanks to Alan Skea for a fine implementation and challenging test of the counter. Thanks to Patrice Bertin for stimulating observations.

[3] a counter which may be incremented or decremented at each cycle.

References

[1] Xilinx, *The Programmable Gate Array Data Book,* Product Briefs, Xilinx Inc., 1987.

[2] V. S. Burtsev, "Accelerating Multiplication and Division Operations in High Speed Digital Computers," *Exact Mechanics and Computing Techniques,* Moscow Academy of Sciences, 1958.

[3] M. Ercegovac, T. Lang, "Binary Counter with Counting Period of One Half Adder Independent of Counter Size," *IEEE Trans. on Circuits and Systems,* Vol. 36, No 6, pp. 924-926, 1989.

[4] L. Guibas, J. E. Vuillemin, "On fast binary addition in n-MOS technologies," *Proc. of ICCC 82,* IEEE New York, pp. 147-151, 1982.

[5] J. E. Vuillemin, *A very fast multiplication algorithm for VLSI implementation,* INTEGRATION, the VLSI Journal, Vol. 1, No 1, pp. 39-52 1983.

[6] C. S. Wallace, *A suggestion for a Fast Multiplier,* IEEE Trans. El. Comp., Vol. EC-13, No 1, pp. 14-17, 1964.

[7] S. Winograd, *On the time required to perform addition,* J. ACM, 12,2, pp. 277-285, 1965.

[8] S. Winograd, *On the time required to perform multiplication,* J. ACM, 14,4, pp. 793-802, 1967.

[9] Xilinx, *The Programmable Gate Array Data Book,* Product Briefs, Xilinx Inc., 1987.