# Long and Fast Up/Down Counters

Mircea R. Stan, *Member, IEEE*, Alexandre F. Tenca,
and Milos D. Ercegovac, *Member, IEEE Computer Society*

**Abstract**—This paper presents recent advances in the design of constant-time up/down counters in the general context of fast counter design. An overview of existing techniques for the design of long and fast counters reveals several methods closely related to the design of fast adders, as well as some techniques that are only valid for counter design. The main idea behind the novel up/down counters is to recognize that the only extra difficulty with an up/down (vs. up-only or down-only) counter is when the counter changes direction from counting up to counting down (and vice-versa). For dealing with this difficulty, the new design uses a "shadow" register for storing the previous counter state. When counting only up or only down, the counter functions like a standard up-only or down-only constant time counter, but, when it changes direction instead of trying to **compute** the new value (which typically requires carry propagation), it simply uses the contents of the shadow register which contains the exact desired previous value. An alternative approach for restoring the previous state in constant time is to store the carry bits in a Carry/Borrow register.

**Index Terms**—Binary counter, constant time counter, serial counter, parallel counter, prescaler, up/down counter.

## 1 Introduction

Counting, when viewed as incrementing an integer number by one, is one of the simplest arithmetic operations, and many design techniques used for speeding up more complex arithmetic operations, especially addition, can be applied to counters and exemplified with their help. Traditionally, counters have been presented as simple state machine examples in textbooks on digital logic design and this has prevented the dissemination of more advanced design techniques which have a more natural place in a computer arithmetic treatise. This paper attempts to present the state of the art in fast counter design with special emphasis on recent results in the design of constant-time up/down counters [22], [23]. Previously published results and patent examples will be analyzed with sometimes surprising conclusions relating to the amount of overlap between different techniques and the existence of many recent patents on textbook-type implementations.

The simplest type of counter is the binary modulo-$2^N$ $N$-bit counter where the value $s(t)$ of the counter is incremented by one in each clock cycle:

$$s(t+1) = s(t) \bmod 2^N.$$

Besides this basic behavior, most counter types have several other features, the most important ones being illustrated with the help of a "black-box" model as in Fig. 1a:

- *resettable*—the counter value is reset to all zeros when the RESET input is active,
- *loadable*—the counter is loaded with the $N$-bit value at the In input lines when the LOAD input is active,
- *reversible*—the counter counts "up" (increments) when the $\overline{\text{UP}}$ / DOWN input signal is inactive and counts "down" (decrements) when the $\overline{\text{UP}}$ / DOWN input signal is active,
- *count enable*—the counter increments every clock cycle only when the CNT input is active,
- *terminal count*—TC output signal active when the counter reaches the maximum value (all ones) counting up or reaches the minimum value (all zeros) when counting down,
- *readable on-the-fly*—the counter state (Out) can be read reliably without stopping the clock. Ideally, this "sampling rate" should be equal to the clock rate.

Some of the above features can be combined in order to obtain a more complex counter behavior. For example, the terminal count can be used with a loadable modulo-$2^N$ counter in order to obtain an arbitrary modulo-$P$ counter (with $P \leq 2^N$) simply by loading the value $2^N - P$ each time TC becomes active. Alternatively, if the counter has only RESET and no LOAD, a modulo-$P$ counter can be obtained by decoding state $P$ and resetting the counter to zero at that moment.

In many cases, counters that are both long and fast are necessary, but speed and size are conflicting qualities for counters because of the carry propagation from low-order to high-order bits. One must be careful with the definition of "speed" though, especially for asynchronous counters. For example, the simple *asynchronous* ripple-carry counter in Fig. 2a can be considered both very fast, since it can have a very high frequency clock (just one load on the clock line), but also very slow, since the delay of the most significant bit is large and grows linearly with the counter size. For some applications, like frequency division, the long delay for carry propagation is not a problem if the clock can still have a high frequency, the simple asynchronous ripple-carry counter being widely used and one of the fastest available [20]. It should be noted that, for historical reasons, asynchronous counters are generally counting *down*.

- *M.R. Stan is with the Department of Electrical Engineering, University of Virginia, Charlottesville, VA 22903. E-mail: mircea@virginia.edu.*
- *A.F. Tenca is with the Department of Electrical and Computer Engineering, Oregon State University, Corvallis, OR 97331, and the University of Sao Paulo, Brazil. E-mail: tenca@ece.orst.edu.*
- *M.D. Ercegovac is with the Computer Science Department, University of California, Los Angeles, CA 90024. E-mail: milos@cs.ucla.edu.*

*For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 106607.*

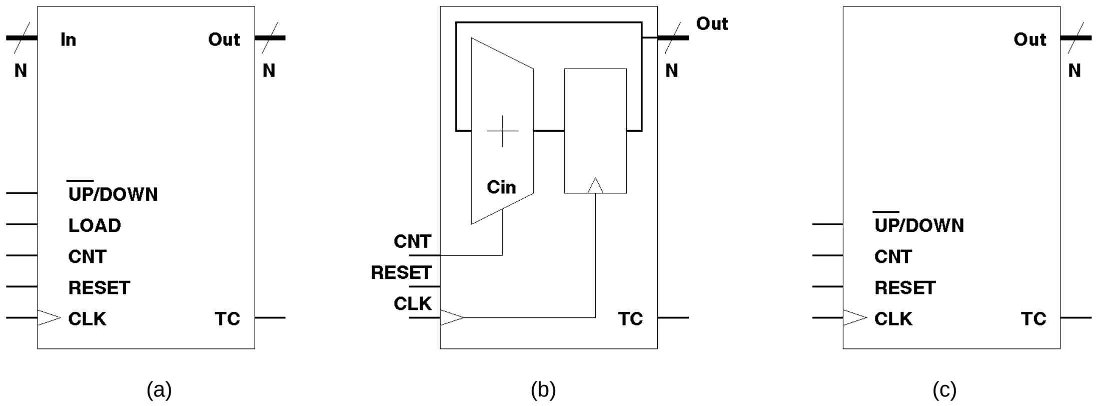


Fig. 1. (a) “Black-box” generic counter model with the most common control signals, (b) counter with the “exposed” adder structure, (c) “black-box” model for nonloadable counter.

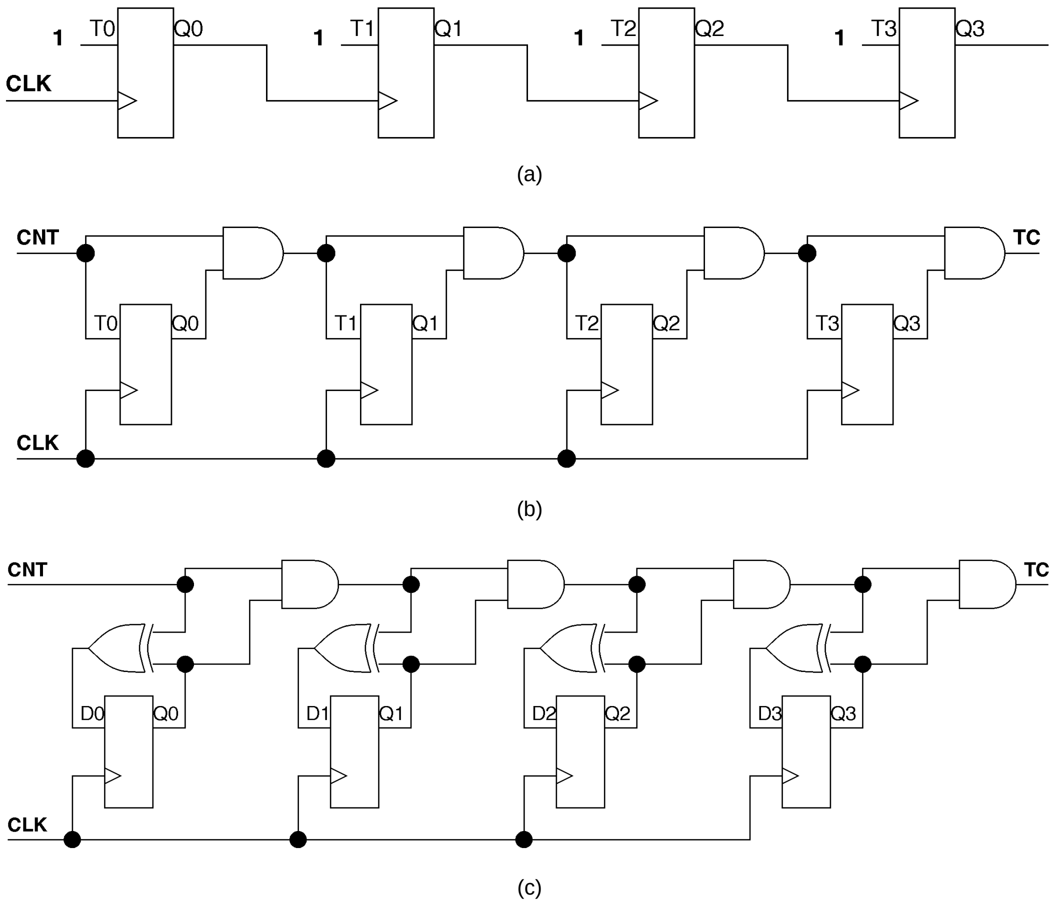


Fig. 2. (a) Asynchronous ripple-carry counter using T-flip-flops, (b) synchronous ripple-carry counter using T-flip-flops, (c) synchronous ripple-carry counter using D-flip-flops and half-adders.

Most counter applications require a synchronous design and the simple ripple-carry *synchronous* counter in Fig. 2b has a very large clock period that increases linearly with the size of the counter, due to the worst case carry propagation, through $(N - 1)$ AND gates. For the rest of the paper, we will assume the “synchronous paradigm” for which the clock is a perfect broadcast signal, hence, all delays will be due to combinational logic. It should be clear, though, that, for “real” designs, the delay of the clock also increases with the number of loads and also becomes a function of the number of flip-flops.

The combinational carry-ripple through AND gates represents the simplest circuit for adding a one to the counter bits with a carry-ripple adder [11]. The adder structure can be clearly revealed if we replace the T-flip-flops traditionally used in counter representation with D-flip-flops and half-adders (HAs) as in Fig. 2c. The rest of this section will discuss several state-of-the-art techniques and a counter classification, Section 2 will introduce the idea of prescaled counters, and Section 3 will present two novel up/down constant time counter structures.

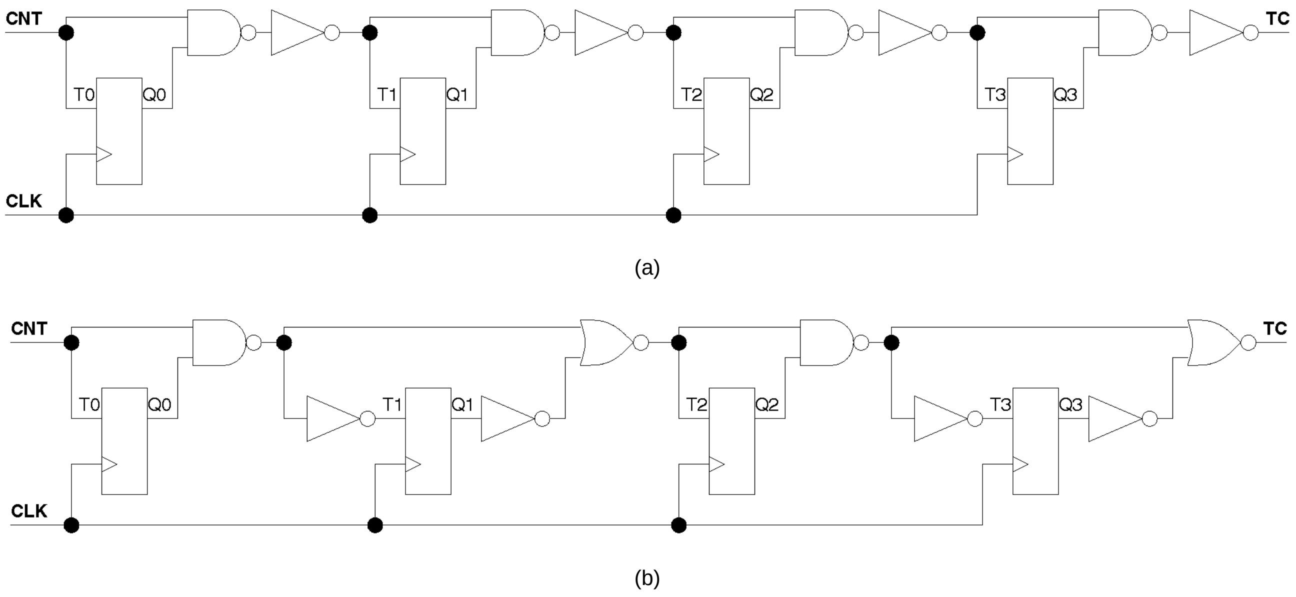


Fig. 3. (a) CMOS implementation of AND carry chain using NAND and INV gates, (b) fast (half the delay) alternating NAND/NOR carry chain.

## 1.1 Counters as Arithmetic Circuits

Revealing the adder "inside" the black-box counter as in Fig. 1b enables the use of all the known techniques for designing fast adders [11] to be applied for fast counter design. There are many known techniques for speeding up addition, and most of them can be also applied to the design of fast counters.

A first observation is that addition can be made faster by using a tree instead of a CARRY-chain for achieving only a logarithmic increase in delay. Surprisingly, this idea is protected by a recent patent [10]. Another observation is that static CMOS gates can implement only inverting functions (NAND, NOR, etc.), hence, the delay of a noninverting function increases due to the delay of a required inverter at the output. A textbook [18] technique for speeding up carry-ripple CMOS adders is to eliminate the inverters in the carry chain by using the "inverting property" of the binary adder by observing that inverting all the inputs of the adder results in inverted outputs and alternating "normal" binary adder cells with "inverted" cells. With this observation, the standard AND carry chain in Fig. 2b (which, for a CMOS circuit, has "hidden" inverters for each AND gate, as in Fig. 3a) can be replaced by a faster alternating NAND/NOR chain which reduces the number of gate delays by half, as in Fig. 3b. Surprisingly again, there is a recent patent [13] covering exactly this technique for counters. An extensive patent search revealed many other patents for counters that simply apply well-known adder structures in the context of counter design. For example, there is a patent on a counter with a Manchester carry-chain [19] and several versions of carry-lookahead [10] and binary tree carry propagation [7] counter structures.

Other traditional approaches for speeding up counters try to improve the circuit implementation of various gates and, especially, flip-flops, for example, by using true-single phase (TSPC) flip-flops [29], [30], [12], [20], but these techniques are not uniquely suited to counters viewed as arithmetic circuits.

In this section, it was advantageous to view counters as a collection of an adder and a state register, since it allowed all the fast adder design techniques to be used for counters, but this view can soon become a limitation when trying to further improve counter performance. Lower bounds on adder delay are well-known [11], [24]; intuitively, the delay of an $N$-bit adder is on the order $O(\log N)$ based on arguments related to tree function implementation with gates with limited fan-in, hence, the minimum clock period for such a counter is also of the order $O(\log N)$. It turns out that going again to the "black-box" model in Fig. 1a and viewing counters as state machines can result in a clock period of order $O(1)$ (constant-time).

## 1.2 Counters as State Machines

One way of breaking the $O(\log N)$ lower bound on the clock period is to pipeline the carry propagation in a "systolic" manner [14], [16] as in Fig. 4. This method doubles the number of required flip-flops and the counter "value" is now represented in a redundant [11] Carry-Save form [16], which, for the case of counters (only one operand), is reduced to the recently introduced Half-Adder form [14]. For many applications which need fast synchronous counting but don't require a binary sequence (e.g., synchronous frequency dividers), a systolic counter is a simple and fast solution. The minimum clock period for a systolic counter is equal to the delay through a flip-flop plus the delay through a logic gate and is independent of the number of bits.

Another way of breaking the $O(\log N)$ lower bound on the clock period is to relax the definition of a counter to sequences other than binary increasing numbers even more. A generalized counter then becomes, by definition, any state machine with a "circular" state diagram, as in Fig. 5. An up-counter will correspond to a state diagram where the states are traversed only in clockwise sequence, a down-counter will have a counterclockwise sequence, and, for an up/down counter, the states can be traversed both clockwise

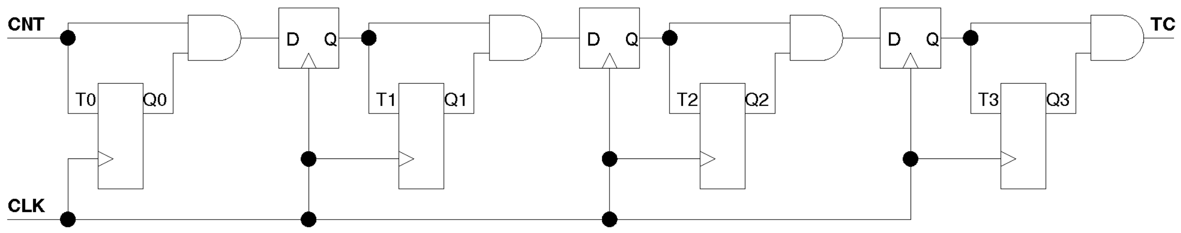


Fig. 4. Pipelined carry chain for a “systolic” counter.

and counterclockwise. An arbitrary state can be chosen as the zero state (ideally encoded as all-zeros) so that the RESET signal will bring the state machine into that initial state. Loading such a generalized counter with meaningful values may be difficult, depending on the state encoding, also, comparing two counter values in order to see which one is “greater” may become impossible.

Generalizing the counter definition in this way opens the possibility of using radically different designs. In principle, any state-encoding and minimization tool can be used for deriving the implementation of such a state machine, but there are some known structures that implicitly have such a circular state diagram. For example, the linear-feedback shift register (LFSR) in Fig. 6a [17] has $O(1)$ (constant) clock period and $O(N)$ (same as binary counter) space complexity, but has a nonbinary output sequence. LFSRs have the feedback connections corresponding to a polynomial with binary coefficients and, for a primitive polynomial, the number of states is $2^N - 1$, hence, an LFSR has a state diagram equivalent to a modulo-($2^N - 1$) counter (all $N$-bit patterns except the all-zeros state). There are ways to obtain a modulo-$2^N$ LFSR (with the all-zeros state) but that may affect the clock period. For many applications which need fast counters but don’t require a binary sequence (e.g., address pointers to circular FIFOs [21], frequency dividers [15]), the LFSR is a good solution, being simple and fast. The minimum clock period for an LFSR is equal to the delay through a flip-flop plus the delay through an XOR and is independent of the number of bits.

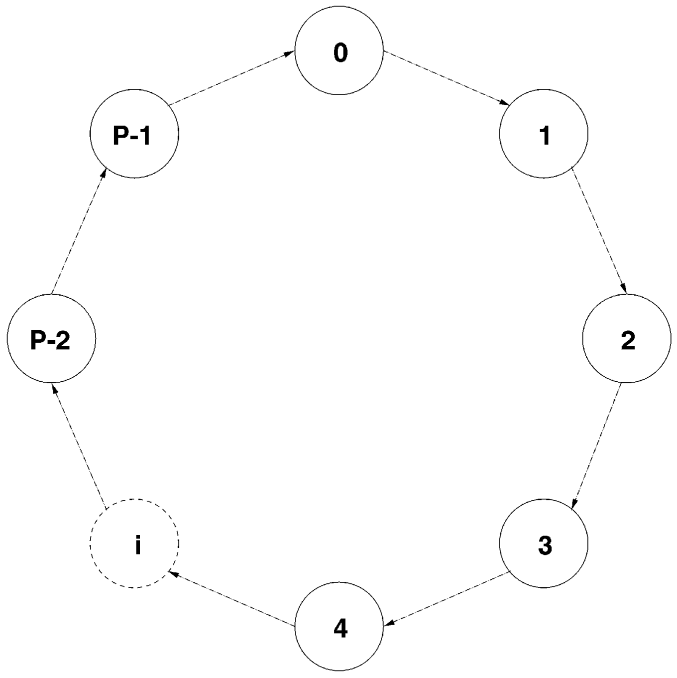


Fig. 5. Circular state diagram for generic modulo-*P* counter.

## 1.3 Ring Counters

The shortest *theoretical* delay for a state machine is obtained when the combinational logic is completely eliminated and the whole sequential structure becomes a shift-register connected in a ring, as in Fig. 6b. The minimum clock period for such a ring counter is only limited by the delay through a flip-flop. If the $P$-bit ring counter is initialized to the “00...001” state, then the state diagram will consist of a circular sequence of $P$ one-hot encoded states. This $P$-bit one-hot sequence is the longest possible for a ring counter and is easy to decode (need to look only at the “hot” bit). Non-one-hot states are harder to decode and may generate less than $P$ distinct states. For example, the “0101...0101” initial state generates a sequence with only two states (“0101...0101” and “1010...1010”).

A clear disadvantage of ring counters is their exponential complexity compared with binary counters. In order to emulate a simple 8-bit binary counter, a long 256-bit ring counter is needed; furthermore, although the delay of the ring counter is theoretically independent of size (synchronous paradigm), for such exponential increases, it is likely that for a real design a 256-bit ring counter may be slower than the much simpler 8-bit binary counterpart. It follows then that ring counters should be used only when the number of states is relatively small. A well-known technique to reduce the complexity of a ring counter in half is to use a twisted-tail (also known as Johnson or Moebius) counter as in Fig. 6c, which also has the advantage that it can be initialized to the all-zeros state. Decoding any state for a twisted-tail counter needs two bits, hence, it can also be done in constant time as with the twice longer “standard” ring counter.

## 1.4 Differential Counters

For applications that need even faster counting, it is possible to derive structures that count *differentially* with a structure which has some similarities to a ring counter [8]. When measuring very short time intervals using a “regular” counter, the accuracy of any time measurement will be determined by the clock period, hence, when the interval to be measured is of the same order of magnitude as the clock period, any measurement becomes meaningless. Counting differentially allows the accuracy to be determined by the *difference* between two different periods. Assuming that we

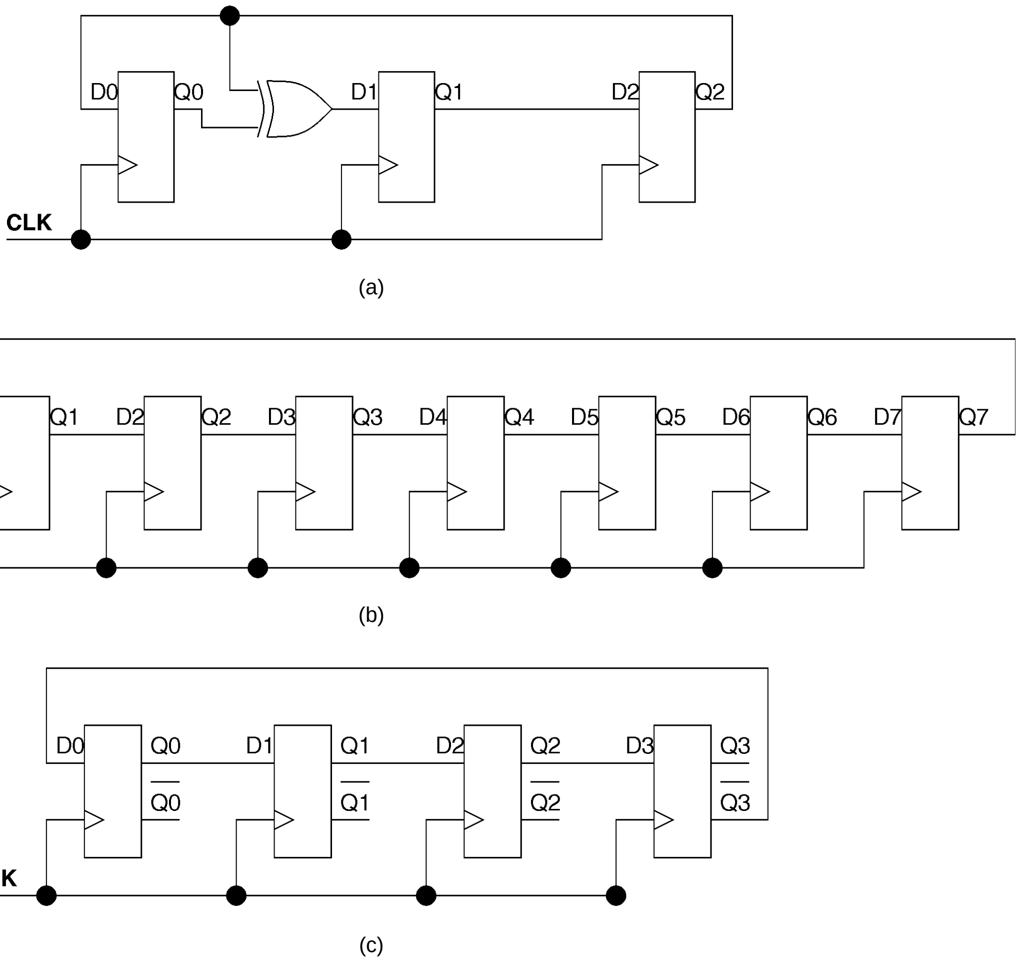

D0
Q0
D1
Q1
D2
Q2
CLK
(a)
D0 Q0 D1 Q1 D2 Q2 D3 Q3 D4 Q4 D5 Q5 D6 Q6 D7 Q7
CLK
(b)
D0 Q0 D1 Q1 D2 Q2 D3 Q3
$\overline{Q0}$ $\overline{Q1}$ $\overline{Q2}$ $\overline{Q3}$
CLK
(c)

Fig. 6. (a) Linear-feedback shift register equivalent to a modulo-7 counter, (b) ring counter equivalent to a modulo-8 counter, (c) Johnson counter equivalent to a modulo-8 counter.

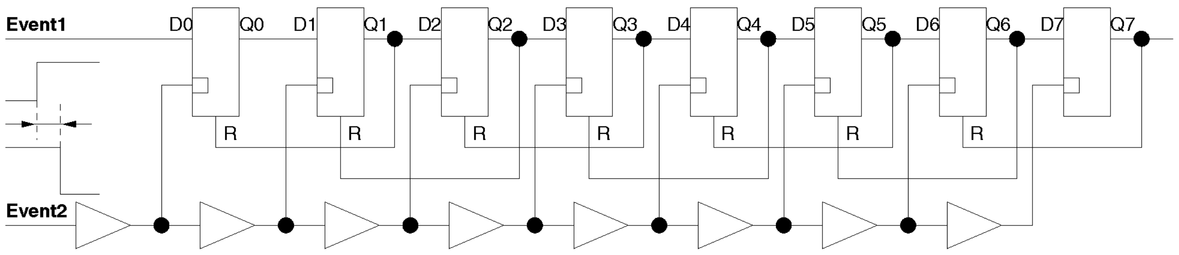


Fig. 7. "Differential" counter. The flip-flops are transparent latches with reset. The first coming "event" is propagated on the slower (latch) path, the second "event" is propagated on the faster (buffer) path. The latches are transparent before the second event reaches them. The output will be a one-hot representation of the moment when the second event has reached the first one.

can accurately control the two periods, very short intervals can be measured with high accuracy, even with relatively slow logic. A "differential counter," as in Fig. 7 [8], has two "periods" which are combinational delays, the faster one through a buffer, the slower one through a transparent latch. The idea is to measure the short interval between two events (signal edges) by propagating the first coming signal through the slower path (transparent latches) and the second one through the faster path (buffers). Since the path for the second signal is faster, there will be a moment when it will catch up the first signal, and the circuit in Fig. 7 captures that moment into a one-hot representation. If the two delays are denoted as $\delta 1$ and $\delta 2$ and the second signal catches up after $k$ stages, then the time interval between the two events is $\triangle = k \cdot (\delta 1 - \delta 2)$ and it can be seen that very small intervals can indeed be measured accurately if the difference $(\delta 1 - \delta 2)$ can be made small enough.

## 1.5 Counters Classification

As we saw, there are many variations possible on the basic counter behavior, hence, there is a need for classifying counters starting from the basic "black-box" model. The following list is not exhaustive but captures the most commonly found cases.

Depending on whether the counter can be initialized to one state or to any state counters can be classified as:

- *Noninitializable*—The simplest case, it can only be used for specific applications (e.g., frequency divider).
- *Resettable*—Necessary for most applications and also necessary for testing purposes, without a large penalty in performance or area.
- *Loadable*—This is a more general case but typically has lower performance and higher complexity.

Depending on whether the counter traverses the circular state diagram in one direction only or in both, counters can be:

- *Up-only*—Most common case, easy to understand.
- *Down-only*—Equivalent to the up-only case in the same sense as subtraction is equivalent to addition, sometimes preferred for convenience (e.g., for a modulo-$P$ counter it is more "convenient" to decode zero state and load a down-counter with the value $P$ than to load an up-counter with the value $-P$).
- *Up/down*—Most versatile but typically at the cost of lower performance (sometimes called a "reversible" counter).

Depending on whether all the state registers are clocked with the same signal counters can be:

- *Asynchronous*—Simple structure, cannot be read "on-the-fly," can have registers that are clocked by other signals than the clock (like the ripple-carry counter in Fig. 2a, sometimes called serial counter [3]) or some other nonclocked sequential structure [2].
- *Semisynchronous*—A "hybrid" attempt of combining the simplicity of asynchronous designs with a synchronous behavior on only some of the outputs (e.g., the terminal count TC [12]).
- *Synchronous*—Most robust and can be read on-the-fly, but the routing and loading of the clock can become a performance bottleneck.

Sometimes it is possible to use "counters" with a state diagram that does not return from the last state to the initial state. Depending on whether this happens or not, counters can be classified as:

- *Periodic*—This is the normal case with a circular state diagram.
- *Aperiodic*—This is the case where the counter does not return to the initial state, an example being the differential counter in Section 1.4.

Periodic counters can be classified according to the number of states:

- *Modulo-$2^N$*—Special case of $2^N$ states, typical for an $N$-bit binary counter, the counter "wraps-around" from the last state by itself, the case of the LFSR with $2^N - 1$ states being similar.
- *Modulo-P*—Although apparently a more general case than the modulo-$2^N$, the modulo-$P$ counter is many times obtained from a modulo-$2^N$ counter, either by decoding the state $P$ and resetting to zero, or by loading with $P$ when the counter reaches zero. Modulo-$P$ *ring* counters are obtained without a need to decode states or explicitly load the counter.

Depending on the state encoding, counters can be classified as:

- *Binary*—The most common case in which the sequence of states is the ascending or descending binary sequence.
- *Quasi-binary*—The case where the relation between a state and its binary equivalent can be easily determined (e.g., for the Half-Adder form of systolic counters in Section 1.2, the binary value can be obtained by adding the Sum and Carry parts). A Gray-code encoded counter used for driving decoders without glitches [26], [1] can be also considered quasi-binary since the binary state can be easily obtained with XORs from the Gray-code state.
- *Nonbinary*—The state encoding is not related to the binary sequence, like in the case of LFSRs and ring counters.

The nonbinary fast counters described until now are adequate for many applications, but many times it is desirable to design binary constant-time counters. *Prescaled* counters, which are the focus of the rest of the paper, are synchronous circuits having the following characteristics:

- Binary counting sequence.
- Clock period independent of counter size.
- Readable on the fly with the sampling rate being equal to the counting rate.
- Space complexity linear in the number of bits (i.e., $O(N)$).
- Count "up," "down," or "up / down".
- Resettable.

## 2 Constant-Time Binary Counters

Being able to design *binary* counters with $O(1)$ period is nonintuitive considering that adders have an $O(\log N)$ period and incrementing is a special case of addition, as we saw in Section 1.1. The following observations give a justification for why constant time binary counters are feasible:

1) The binary number system has a special periodicity in the way the CARRY-in to high order bits is generated, which makes it both predictable and with a low frequency [25],
2) The black-box model of a *nonloadable* counter (Fig. 1c) has only a limited number of inputs: Clock (CLK), RESET, and Count Enable (CNT). This explains why the binary tree logic decomposition which leads to the $O(\log N)$ delay for an adder or loadable counter can be circumvented for nonloadable counters.

An ascending sequence of binary numbers has many interesting properties, as can be seen in Table 1, which shows a 4-bit counter divided into two 2-bit blocks. The higher order bits are stable for long periods of time and the terminal count ($TC_1$) output from the two least significant bits, which becomes a CARRY-in into the most significant block, is periodic with a lower frequency than the clock signal. For an $M$-bit counter block, the terminal count will have a frequency $2^M$ lower than the clock, with the moment when the terminal count from low-order bits is active being exactly the time when higher order bits need to be incremented. This means that the "virtual" frequency at which high-order bits

TABLE 1
BINARY SEQUENCE COUNTING UP
(< 1 REPRESENTS THE CARRY)

| number | $b_3b_2$ | $TC_1$ | $b_1b_0$ |
|---|---|---|---|
| 0 | 00 | 0 | 00 |
| 1 | 00 | 0 | 01 |
| 2 | 00 | 0 | 10 |
| 3 | 00 | <1 | 11 |
| 4 | 01 | 0 | 00 |
| 5 | 01 | 0 | 01 |
| 6 | 01 | 0 | 10 |
| 7 | 01 | <1 | 11 |
| 8 | 10 | 0 | 00 |
| 9 | 10 | 0 | 01 |
| 10 | 10 | 0 | 10 |
| 11 | 10 | <1 | 11 |
| 12 | 11 | 0 | 00 |
| 13 | 11 | 0 | 01 |
| 14 | 11 | 0 | 10 |
| 15 | 11 | <1 | 11 |

need to operate is much lower than for the low-order bits and this is exactly the idea behind prescaled counters. This idea has been in use for a relatively long time, starting with the "old" 74160-163 series 4-bit TTL counters [15], but only recently has been formalized in academic publications [4], [25].

## 2.1 Prescaled Counters

Prescaling long counters requires partitioning them into a series of subblocks of increasing sizes in order to take advantage of the reduced frequency required by high order bits. The simplest prescaled counters have only two such blocks, with a small and fast least-significant module called the prescaler and a slower large counter for high-order bits [20], like in Fig. 8. We were again surprised to find a recent patent [3] covering this basic well-known technique.

What makes a prescaled counter work is the fact that, due to the characteristics of the binary number system, the TC from the prescaler to the high order bits (which corresponds to the moments when the high order bits have to be incremented) has a low frequency. In this way, the "virtual clock frequency" for the slow high-order block is $2^M$ smaller than the true clock frequency, and the CARRY propagation *inside* the high-order partition can take a long time even with a fast clock.

A simple reasoning leads to a theoretically unlimited extension of the counter size, without increasing the clock period, by adding more partitions [4], [25]. For higher order blocks, successive terminal count signals from the previous stages become exponentially farther apart in time, hence, higher order blocks can have exponentially increasing sizes, and, for all practical purposes, three or four such partitions are typically enough [25]. In a correctly designed constant time counter, the clock period is limited only by the speed of the least significant block, hence, the first prescaler is typically very small (one or two bits).

The CARRY propagation *inside* a partition has to be faster than the "virtual clock" for that block. Generally, it is desired that the design be as simple (i.e., small) as possible, hence, a ripple CARRY propagation is typically chosen inside each partition. For such an arrangement, the number of bits inside a partition is determined by dividing the "virtual clock" period by the gate delay for one bit of carry propagation. The size of each subblock must be chosen such that the CARRY propagation inside the block is shorter than the delay between two successive terminal counts from the corresponding prescaler. In this way, the CARRY propagation inside the block is not on the critical path and does not affect the clock period.

## 2.2 Terminal Count Generation

The prescaled generation of the TC-in to a partition has to be synchronous with the true clock. Several different approaches have been proposed for the prescaled generation of the TC to high-order partitions. The first proposed solution, by Ercegovac and Lang [4], uses a (relatively inefficient) ring/twisted-tail counter, which practically doubles the overall complexity of the counter. The ring/twisted-tail counters are regular and their VLSI implementation may not be very inefficient. A much simpler TC generation, proposed by Vuillemin [25], uses a backward CARRY propagation chain [12] that takes the characteristics of the binary number system further into account.

## 2.3 Partitioning

Depending on the choice of the prescaled CARRY-in generation method, the partition sizes can be determined:

- In a *top-down* manner [4] by first determining the size of the most significant block, which is chosen as large as possible, and then recursively determining the sizes of the lower order blocks. By assuming *unit delays* for the combinational gates and a *unit delay* clock, an $N$-bit counter is first partitioned into an $(N - \lceil \log_2 N \rceil)$ most significant block and into another $\lceil \log_2 N \rceil$ block which is recursively partitioned in the same manner [4]. For example, in the case of a 64-bit counter, a top-down partitioning results in the following block sizes: 58, 3, 2, 1 [4]. The top-down procedure reduces the penalty paid for having ring counter prescalers, but has the disadvantage that counters of different sizes will require different partition sizes, hence, design reuse is

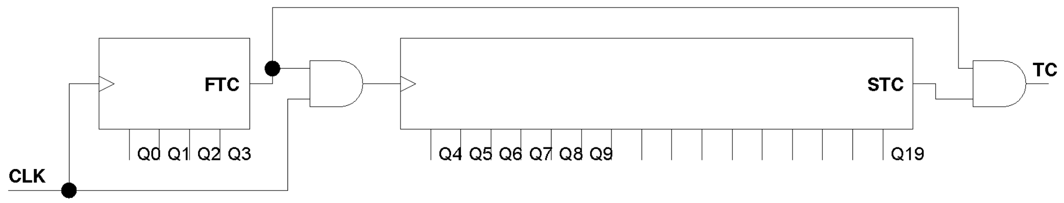


Fig. 8. Counter partitioned into a fast prescaler and a slower high-order partition.

difficult to implement. For a 128-bit counter, the top-down partitioning leads to: 121, 4, 2, 1 block sizes. This scheme has recently been refined in [23] as follows: An $N$-bit counter can be partitioned into an ($N - \lfloor log_2 N \rfloor$)-bit block and a $\lfloor log_2 N \rfloor$-bit block, whenever $\left(N - \lfloor log_2 N \rfloor\right) \le 2^{\lfloor log_2 N \rfloor}$. When the condition does not apply, the original partitioning method is used.

- In a *bottom-up* manner [25] by first deciding the size of the least significant block, then choosing the second block as large as possible without affecting the clock period, then choosing the third, etc. A bottom-up partitioning, which assumes unit delays for the combinational gates, and a unit delay clock, which determines the least significant block with $n_0 = 1$ bit, the second block with $n_1 = 2^{n_0} = 2$ bits, the third block with $n_2 = 2^{(n_0+n_1)} = 8$ bits, and so on [25]. For the same example of a 64-bit counter, a bottom-up partitioning results in the following block sizes: 53, 8, 2, 1. This bottom-up procedure has the advantage of using a few "standard size" modules as building blocks for counters of different lengths with only the most significant block of a non-standard size. For a 128-bit counter, the bottom-up partitioning leads to: 117, 8, 2, 1 block sizes. This scheme has been recently refined in [31], [28].

## 3 Up/Down Binary Counters

### 3.1 Down Counters

Each bit of an up counter can be described by the half addition $s(t) + c_{in} = 2c_{out} + s(t+1)$, while, for a down counter, each bit can be described by the half subtraction $s(t) - c_{in} = -2c_{out} + s(t+1)$. The truth table of these operations is shown in Table 2. The CARRY-out (or BORROW-out) of one module becomes TC-out, which is connected to the CARRY-in (or BORROW-in) of the next module in the chain.

As the $s$ output is the same for addition and subtraction, the function that generates $s(t+1)$ depends on variables $s(t)$ and $c_{in}$, but not on the operation to be performed. It can be seen that down-counters have very similar characteristics to up-counters, hence, designing a constant time down-counter is almost identical to designing an up-counter, the only difference being the need for a BORROW chain instead of the CARRY chain of the up-counter (practically, this can be accomplished by inverting the inputs to the AND gates that compute the chain [27]).

Unlike up-only and down-only counters, loadable counters and up/down counters do *not* exhibit the nice periodicity and predictability of the $TC_1$ (CARRY-in or BORROW-in) to high order blocks [27]. After a *load*, a loadable counter cannot guarantee enough time for CARRY propagation inside the subblocks, while an up/down counter can reverse direction at any moment, as can be seen in Table 3, which again does not guarantee enough time for CARRY (or BORROW) propagation. It is interesting to note that loadable counters have a large number of input lines (the direct load lines) which grows linearly with the number of bits, but up/down counters have only a constant number of inputs (CLK, $\overline{\text{UP}}$ / DOWN, RESET, and CNT) independent of the counter size, hence, it seems more likely to be able to design a constant time up/down counter than a constant time loadable counter. In spite of this, constant time up/down counters have only been recently reported [22], [23], while there have been several reported techniques (e.g., "pulse swallowing" and "state skipping" [27]) that enable a loadable counter to have a quasi-constant time behavior by letting the counter output be out of sequence for a period of time after loading.

TABLE 2
Addition/Subtraction

| $s(t)$ | $c_{in}$ | $s(t) + c_{in}$ | | $s(t) - c_{in}$ | |
|---|---|---|---|---|---|
| | | $c_{out}$ | $s(t+1)$ | $c_{out}$ | $s(t+1)$ |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |

TABLE 3
Binary Sequence Counting Up/Down
(< 1 Represents a CARRY, > 1 Represents a BORROW)

| number | $b_3b_2$ | $TC_1$ | $b_1b_0$ | dir |
|---|---|---|---|---|
| 0 | 00 | 0 | 00 | up |
| 1 | 00 | 0 | 01 | up |
| 2 | 00 | 0 | 10 | up |
| 3 | 00 | < 1 | 11 | up |
| 4 | 01 | 0 | 00 | up |
| 5 | 01 | 0 | 01 | up |
| 6 | 01 | 0 | 10 | up |
| 7 | 01 | < 1 | 11 | up |
| 8 | 10 | > 1 | 00 | down |
| 7 | 01 | < 1 | 11 | up |
| 8 | 10 | 0 | 00 | up |
| 9 | 10 | 0 | 01 | up |
| 10 | 10 | 0 | 10 | up |
| 11 | 10 | < 1 | 11 | up |
| 10 | 11 | 0 | 00 | down |
| 13 | 11 | 0 | 01 | up |
| 14 | 11 | 0 | 10 | up |
| 15 | 11 | < 1 | 11 | up |

### 3.2 Constant-Time Up/Down Counters

The main idea behind the technique for designing constant time up/down counters is to realize that it is easy to have a *configurable* counter (configured as an up-counter, it will have a CARRY chain and, configured as a down-counter, it will have a BORROW chain) and the only extra difficulty vs. an up-only or down-only counter is when the counter changes direction. This change of direction is the only moment when the CARRY (or BORROW) chain *inside* a block may not have enough time to propagate until the next TC from the corresponding prescaler. The solution proposed here is to have the desired value *prestored* and simply load this value when necessary, instead of trying to compute it. This can be easily accomplished by using a "shadow" register that is always loaded with the previous block value whenever the block is loaded with a new value.

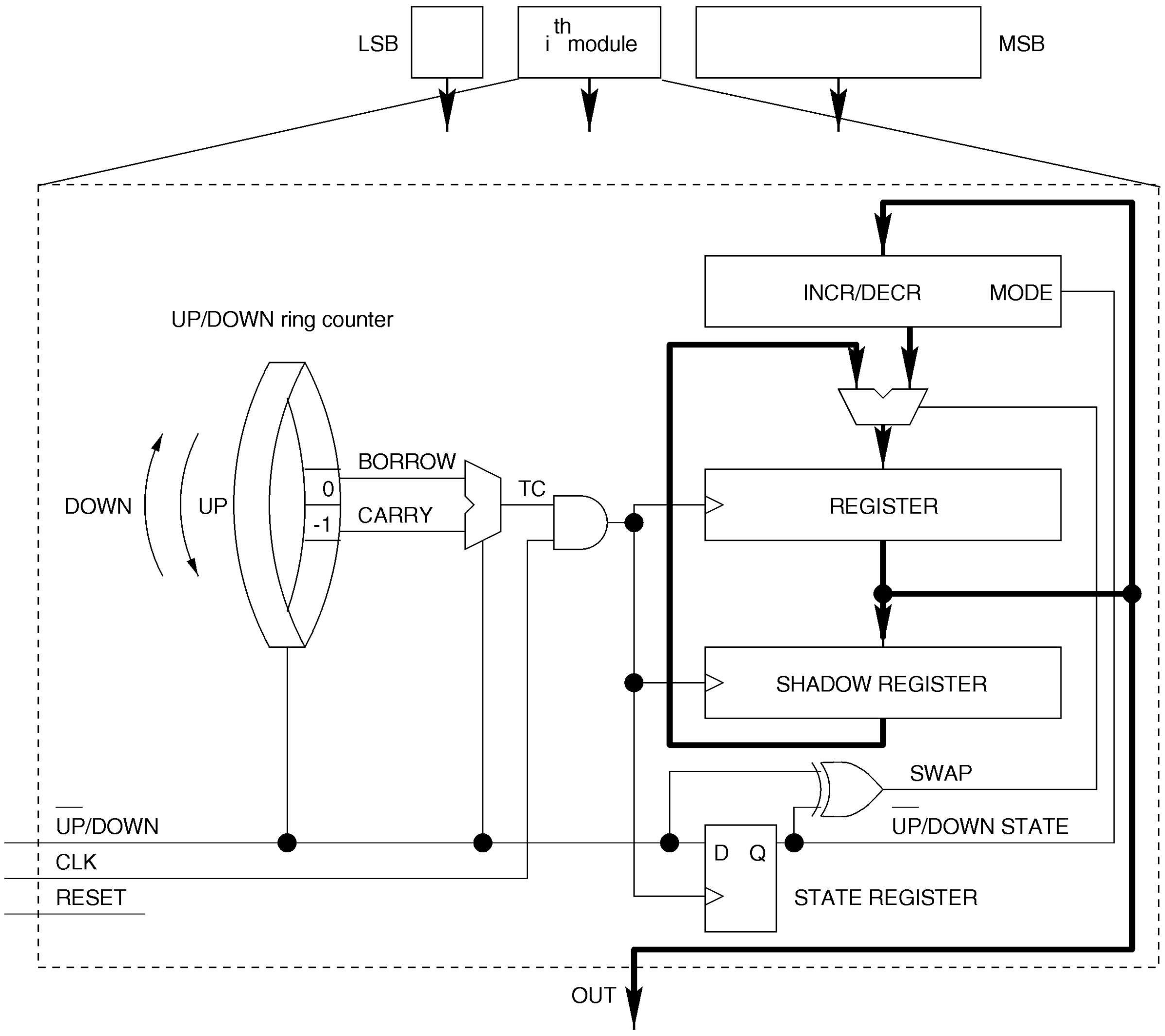


Fig. 9. Block diagram of the constant time up/down counter.

The block diagram of the proposed up/down constant time counter is shown in Fig. 9. The design is synchronous, with a CLK active on the rising edge, a RESET active HI, and an $\overline{\text{UP}}$ / DOWN input, which is LO for counting up and HI for counting down. If desired, a separate Count Enable (CNT) can be easily added by gating the CLK, or by AND-ing CNT with the local signals that enable counting if clock gating is not desirable. The following issues have determined the structure of the new counter:

- The prescaled TC generation must itself be up/down, hence, it can be implemented as an up/down ring counter similar to the ring counter proposed by Ercegovac and Lang [4]. This also implies that a top-down partitioning method [4] is needed in order to minimize the size of the ring counters. Unfortunately, a combinational chain, as proposed by Vuillemin [25], does not seem to work for an up/down counter because the counter has to be able to change direction in any cycle.
- Each block needs to be configurable for counting either up or down. A separate configuration bit for each block is needed to keep track of the block configuration.
- Each subblock has a shadow register that stores the previous block value (i.e., decremented or incremented by one block-least-significant bit depending on the configuration). When the block configuration is “up,” the shadow stores the present value minus one LSB and, when the configuration is “down,” it stores the present value plus one LSB.

The subblocks in this design function practically independently of each other, the ring counter inside each block effectively replacing the need for receiving the TC from lower-order blocks. The complexity of the up/down counter is approximately twice as large as that for an up-only counter, as in [4], and four times as large as that in [25], because of the extra shadow register and the configurable CARRY chain.

## 3.3 Least-Significant Bit Counter

A 1-bit counter counts in the same sequence, no matter if it is up-only, down-only, or up/down, hence, the first block (see Fig. 10) can be a simple 1-bit counter which acts both as the 1-bit least significant bit and as a ring counter for the second block. There is no need for a shadow register or configuration bit for the first block.

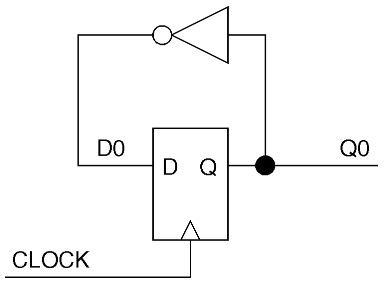


Fig. 10. The least-significant bit block.

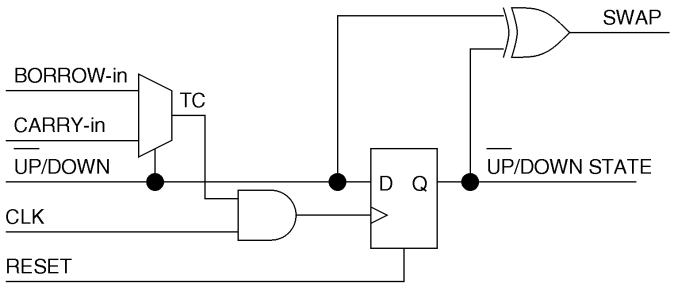

SWAP
BORROW-in
TC
CARRY-in
$\overline{\text{UP}}$/DOWN
$\overline{\text{UP}}$/DOWN STATE
D Q
CLK
RESET

Fig. 11. The configuration bit inside each block.

## 3.4 Configuration Bit

A configuration bit for each higher-order block keeps track of how the block is configured (up or down). A CARRY-in can occur only if the $\overline{\text{UP}}$ / DOWN input signal is 0 (UP), while a BORROW-in can occur only if the $\overline{\text{UP}}$ / DOWN input signal is 1 (DOWN). There are four possible situations:

- The block is configured "up" and a CARRY-in comes from the ring counter. The configuration remains the same ("up") and the block behaves like a normal up-only constant time counter. The shadow register gets loaded with the present block value, while the block gets loaded with its next (incremented) value. Since the present configuration is "up," this means that the previous "event" was also a CARRY-in, hence, enough time has passed for CARRY propagation inside the Incrementer/Decrementer (which is configured as an incrementer).
- The block is configured "down" and a BORROW-in comes from the ring counter. The configuration remains the same ("down") and the block behaves like a normal down-only constant time counter. The shadow register gets loaded with the present block value, while the block gets loaded with its next (decremented) value. Since the present configuration is "down," this means that the previous "event" was also a BORROW-in, hence, enough time has passed for BORROW propagation inside the Incrementer/Decrementer (which is configured as an decrementer).
- The block is configured "up" and a BORROW-in comes from the ring counter. The block changes configuration to "down" and it swaps the present value with the shadow register. The Incrementer/Decrementer output is disabled in this case, hence, there is no need in this case to wait for BORROW propagation.
- The block is configured "down" and a CARRY-in comes from the ring counter. The block changes configuration to "up" and it swaps the present value with the shadow register. The Incrementer/Decrementer output is disabled in this case, hence, there is no need in this case to wait for CARRY propagation.

There is a somewhat subtle point in realizing that the configuration bits for different blocks can be different at times. This happens, for example, when, after counting up for a number of cycles, the counter changes "direction" by only changing lower order bits. In such a case, the high-order blocks will still remain configured "up" and will only change configuration when a BORROW-in comes from the corresponding ring counter. If the counter changes direction again, before a BORROW-in, the higher order block will never "know" that the lower order blocks were in a different configuration for a period of time.

The configuration register is implemented as a simple D edge-triggered flip-flop (Fig. 11). The configuration of each block can only change when a CARRY-in (or BORROW-in) is received from the ring counter. When the configuration changes (the present configuration is "up" and the next one is "down," or vice-versa), the SWAP signal becomes active, which enables swapping the value of the block with the shadow register. When the configuration stays the same, the block register is loaded from the Incrementer/Decrementer and the shadow register is updated with the previous block value.

The main block register and the shadow register are implemented with the same D-type edge-triggered registers as the configuration bit.

## 3.5 Clock Period

For simplicity, we will assume unit delays for all the combinational gates in the circuit (including multiplexers and XOR gates). There are several critical paths in the circuit that determine the minimum clock cycle to be larger than one unit delay, as in the case of the up-only counter:

- The least significant bit block has a unit delay, so it does not represent the critical path.
- Incrementing/decrementing the ring counter requires two unit delays, since the ring counter is a bidirectional shift register.
- Loading (actually "swapping") the value of the block with the value of the shadow register requires a unit delay through a multiplexer and, in parallel, the multiplexer control signal requires two unit delays (see Fig. 9). The timing of the $\overline{\text{UP}}$ / DOWN signal is on the critical path and the signal needs to be synchronized with the clock.
- By choosing a proper size for each block, the delay of Incrementer/Decrementer which takes care of CARRY (or BORROW) propagation inside the block can be masked, and this delay should not be on the critical path.

The clock frequency is independent of counter size but is lower (by a constant) than for an up-only counter because of the extra complexity. Instead of being limited only by the low order prescaler, the speed is also limited by the extra logic needed for swapping with the shadow register. In a

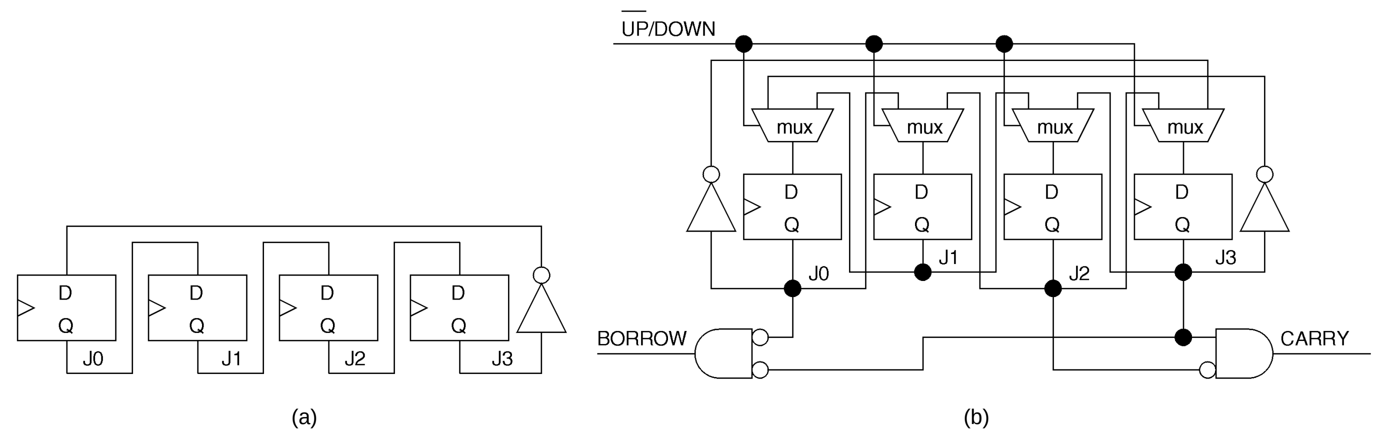


Fig. 12. Four-bit (eight states) twisted-tail ring counters (Johnson or Moebius): (a) up-counter, (b)up/down counter.

real design, the clock period can be larger than two unit delays due to particular implementation details and fan-out effects on long lines.

## 3.6 Up/Down Ring Counter

A $2^k$-bit twisted-tail ring counter (see Section 1.3) has $2^{(k+1)}$ distinct states and clock period independent of size, hence, it can function as a ($k$ + 1)-bit counter prescaler [4] (see Fig. 12a for an up-only counter). In the case of the proposed up / down counter, an up / down ring counter is needed, and this can be easily obtained, as in Fig. 6b. The ring counter inside each block is used in order to generate, in constant time, the TC-in (CARRY-in or BORROW-in) for the block. The TC signal is obtained from different conditions depending if the counter is counting up or down. When counting up, TC = 1 when the state of the enable counter is $s(t)$ = (100...00) (one state before the counter goes back to state 0) and CNT = 1. When counting down, TC = 1 when $s(t)$ = (000...00) and CNT = 1. The state bits in the twisted-tail counter are such that the $s(t)$ = (100...00) state can be detected by testing the two most significant bits and the $s(t)$ = (000...00) state can be detected by testing the most and least significant bits (see Fig. 13).

## 3.7 Partitioning

Determining the partition sizes for the proposed up / down counter proceeds top-down, similarly to [4], the only difference being that the minimum clock period ($T_{clk}$) is larger than the combinational unit delay due to the extra complexity. If we consider $T_{clk} = p \cdot \delta$, where $\delta$ is the unit delay, the partitioning first divides the $N$-bit counter into a most significant $N - \left\lceil \left(\log_2 \frac{N}{p}\right) \right\rceil$ block and into another $\left\lceil \left(\log_2 \frac{N}{p}\right) \right\rceil$ block which is recursively divided in the same manner until the smaller block is a 1-bit counter. For $p$ = 2 and $N$ = 64, the partitioning leads to the sizes: 59, 3, 1, 1. For $p$ = 4 and $N$ = 64, the partitioning leads to the sizes: 60, 3, 1.

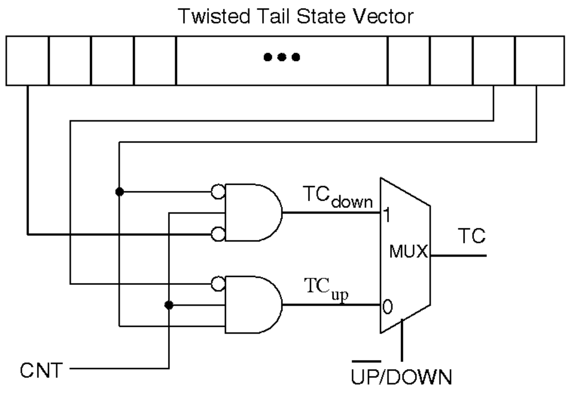


Fig. 13. Up/down twisted-tail TC circuit.

## 3.8 Incrementer/Decrementer

The Incrementer / Decrementer can be easily implemented as a ripple chain, as in Fig. 14. Surprisingly again, this straightforward textbook design is protected by a patent [5]. For an $n$-bit block, the delay through the ripple chain will be $n$ times the unit logic delay, and, if this delay is less than the time between two consecutive CARRY-ins (or BORROW-ins) from the ring counter (which should always be true by partitioning), the Incrementer / Decrementer is not on the critical path. The configuration is controlled by the configuration bit for each block.

## 3.9 Initializing the Counter

For a “regular” $N$-bit counter which has $2^N$ possible states, all the states are legal and a RESET signal may not even be needed for some applications (although desirable, at least for testing [9]). In the case of the proposed constant time up / down counter, which has many extra state flip-flops (the configuration bits, the ring counters, the shadow registers), it is very important to initialize the counter to a legal state. A RESET signal is needed to initialize all the configuration bits to 0 (counting “up”), the counter block values to all-zeros, the shadow registers to all-ones (block value minus 1), and the ring counters to all-zeros. It would be hard to load the counter with an arbitrary legal value as required by a loadable counter.

## 3.10 Alternative Design

As explained earlier, for an up / down counter, there is no time to wait for the carries or borrows to propagate when the direction of counting changes. To solve the problem, we have proposed the use of a “shadow” register to store the previous counter state [22]. An alternative solution is to store the bit-wise XOR between the previous state and the

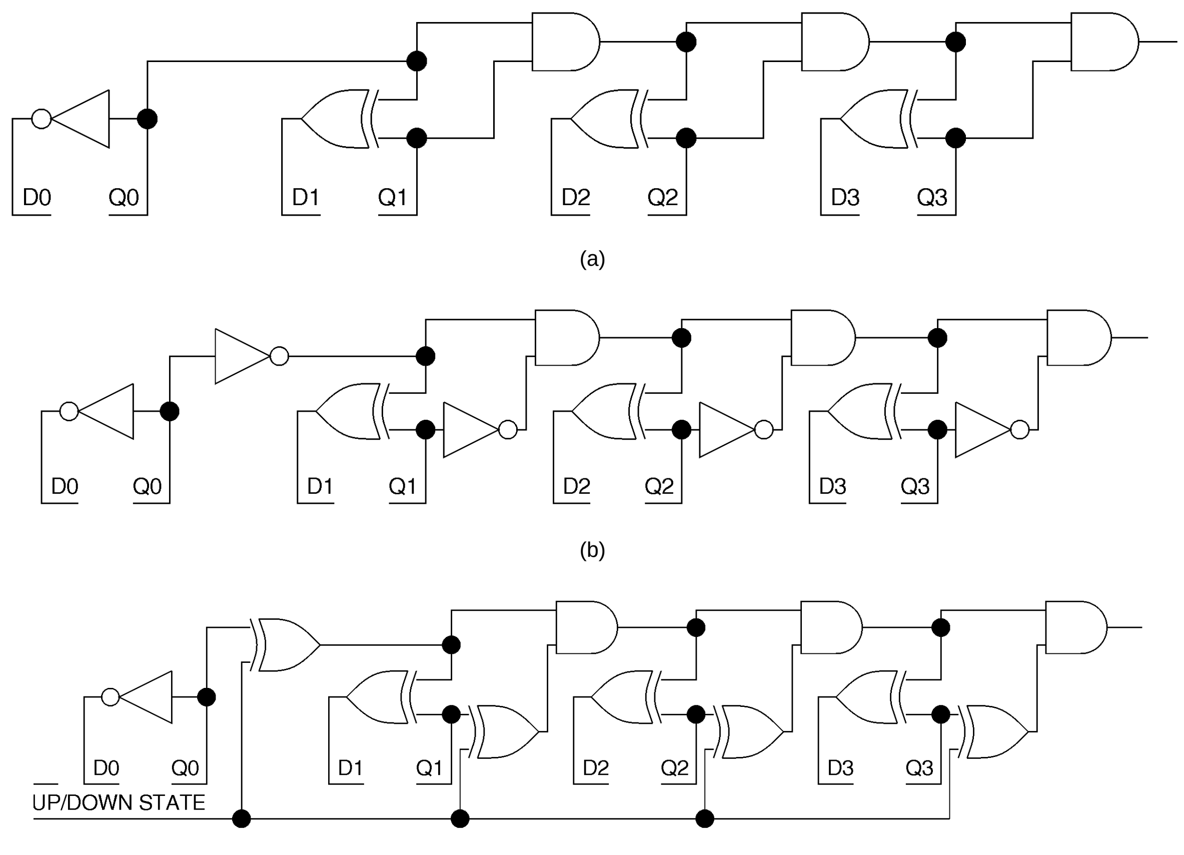


Fig. 14. Ripple chain: (a) incrementer (CARRY chain), (b) decrementer (BORROW chain), (c) incrementer/Decrementer.

current state in a *Carry/Borrow Register* (CBReg) [23]. With this information available, it is possible to restore the desired previous state in one gate delay. For comparison, both solutions are presented in Fig. 16. In Fig. 16a, the carry bit that was used in the last transition is stored in CBReg and is used in the place of the carry computed by the incrementer/decrementer chain [23]. In Fig. 16b, the value of the previous state is stored in a "shadow" register and can replace the next state that is being computed by the incrementer/decrementer [22].

### 3.11 Experimental Results

The up/down counter was implemented for both designs shown in the previous sections in two different technologies. The first design (using the Shadow Register [22]) was implemented in an Atmel AT6000 FPGA, while the second design (using the Carry/Borrow Register [23]) was implemented in an Xilinx XC4000 FPGA. The synthesis results were obtained without imposition of constraints on the synthesis tools. No manual placements or routings were performed, which leaves some space for optimizations and better performance. The counter [23] was also tested in the EVC board with low operating frequency, only to verify the counter operation.

Both designs implement a 64-bit up/down counter partitioned into three modules with 60, 3, 1 bits that runs at 40MHz in the Atmel FPGA and at 47.2MHz in the Xilinx FPGA. The partitioning has resulted from the minimum clock period which is $p = 4$ times the minimum combinational delay.

A functional simulation at 40MHz of the 64-bit up/down counter using the Atmel part is shown in Fig. 15. As can be seen, the counter counts up and down and can change direction each cycle.

## 4 Conclusions

We have presented the methodology behind designing synchronous up/down counters of arbitrary length with period independent of counter size, which was an open problem until recently [25], [22], [23]. The main idea is to store the previous state of the counter for use when the counter reverses direction [22], [23]. A somewhat related idea was proposed by Hendry for storing one bit per state to speed-up counting [6].

The experimental results for the up/down counters were obtained using simulation for a 64-bit design and estimates of the area and delay for other cases. It should be relatively easy to migrate this design to a different architecture or counter size. Since logic synthesis tools are not going to "discover" such a design, it is best to put it in a module library which can be parameterized by the counter length. Such a design only makes sense when relatively long (more than 24 bits) up/down counters are needed. For short counters, better (faster or simpler) results can be probably obtained with other approaches which asymptotically are worse but are better for small numbers.

The proposed up/down counters are not loadable, hence, designing a constant-time loadable binary counter is still an

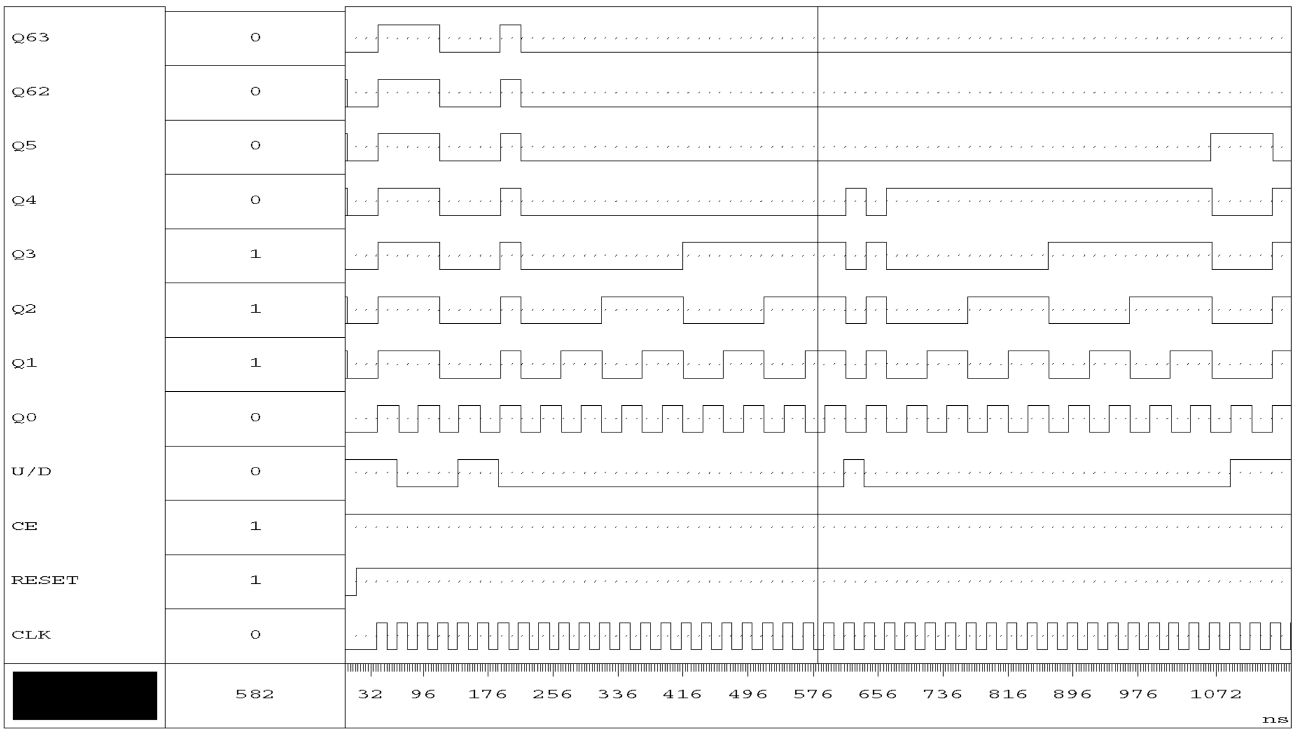


Fig. 15. Simulation of the 64-bit up/down counter at 40MHz. Only the six least-significant and the two most-significant bits are shown.

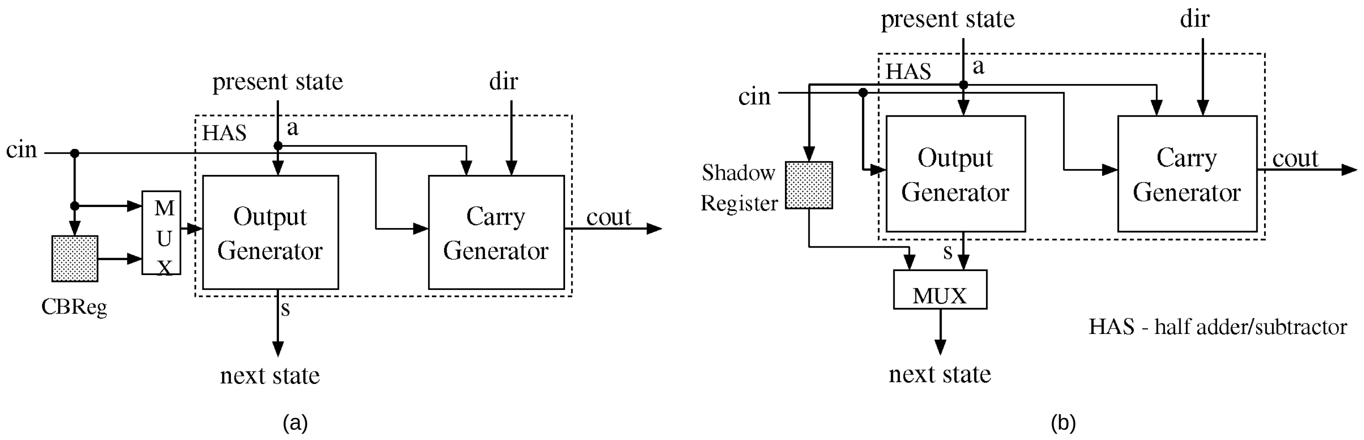


Fig. 16. Designs of the next state generator: (a) Using carry/borrow from previous state transition, (b) using shadow register to store the previous state bit.

open problem. In the context of constant-time state machines (counters being just an example), it would be interesting to be able to determine when a state machine can have period $O(1)$ just by looking at the state transition graph and the state encoding. Even more interesting would be to determine a state encoding that enables a $O(1)$ period for a given state transition graph (STG) if such an encoding exists.

## Acknowledgments

This research has been supported in part by the U.S. National Science Foundation Career Grant MIP-9703440 "Advances in Theory, Design Methods and CAD for Low-Power VLSI," the U.S. National Science Foundation Grant MIP-9314172 "Arithmetic Algorithms and Structures for Low-Power Systems," CNPq, Xilinx, and Atmel.

## References

[1] S. Amuro, "Adjacent Code System," U.S. Patent no. 5,329,280, July 1994.
[2] D. Chu and M. Ward, "Data Capture in an Uninterrupted Counter," U.S. Patent no. 4,519,091, May 1985.
[3] D.H. Eby, "Synchronous Programmable Two-Stage Serial/Parallel Counter," U.S. Patent no. 4,905,262, Feb. 1990.
[4] M.D. Ercegovac and T. Lang, "Binary Counter with Counting Period of One Half Adder Independent of Counter Size," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 924-926, June 1989.
[5] M.W. Evans, "Minimal Logic Synchronous Up/Down Counter Implemenations for CMOS," U.S. Patent no. 4,611,337, Sept. 1986.
[6] D.C. Hendry, "Sequential Lookahead Method for Digital Counters," *Electronics Letters*, vol. 32, pp. 160-161, Feb. 1996.

[7] T. Iida and T. Ikarashi, "Synchronous Binary Counter," U.S. Patent no. 4,679,216, July 1987.

[8] J. Kalisz, R. Szplet, R. Pelka, and A. Poniecki, "Single-Chip Interpolating Time Counter with 200-ps Resolution and 43-s Range," *IEEE Trans. Instrumentation and Measurement*, vol. 46, pp. 851-856, Aug. 1997.

[9] M. Katoozi and M. Soma, "A Testable CMOS Synchronous Counter," *IEEE J. Solid-State Circuits*, vol. 23, pp. 1,241-1,248, Oct. 1988.

[10] M. Kondo and T. Watnabe, "Synchronous Counter," U.S. Patent no. 5,526,393, June 1996.

[11] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, N.J.: Prentice Hall, 1993.

[12] P. Larsson and J.R. Yuan, "Novel Carry Propagation in High-Speed Synchronous Counters and Dividers," *Electronics Letters*, vol. 29, pp. 1,457-1,458, Aug. 1993.

[13] S.-K. Lee, "Binary Counter with Sped-Up Ripple Carry," U.S. Patent no. 5,559,844, Sept. 1996.

[14] D.R. Lutz and D.N. Jaysimha, "Programmable Modulo-k Counter," *IEEE Trans. Circuits and Systems*, vol. 43, pp. 939-941, Nov. 1996.

[15] B. Norris, *Digital Integrated Circuits and Operational-Amplifier and Optoelectronic Circuit Design*. New York: McGraw-Hill, 1976.

[16] K.Z. Pekmestzi and N. Thanasouras, "Systolic Frequency-Dividers Counters," *IEEE Trans. Circuits and Systems*, vol. 41, pp. 775-776, Nov. 1994.

[17] W.W. Person, *Error Correcting Codes*. Cambridge, Mass.: MIT Press, 1961.

[18] J.M. Rabaey, *Digital Integrated Circuits*. Upper Saddle River, N.J.: Prentice Hall, 1996.

[19] S.R. Ramirez, "Counter Cell and Counter Circuit," U.S. Patent no. 5,495,513, Feb. 1996.

[20] R. Rogenmoser, Q. Huang, and F. Piazza, "1.57 GHz Asynchronous and 1.4 GHz Dual-Modulus 1.2 $\mu$m CMOS Prescalers," *Proc. Custom Integrated Circuits Conf.*, pp. 387-390, May 1994.

[21] M.R. Stan, "Shift-Register Generators for Circular FIFOs," *Electronic Eng.*, pp. 26-27, Feb. 1991.

[22] M.R. Stan, "Synchronous Up/Down Counter with Period Independent of Counter Size," *Proc. IEEE Symp. Computer Arithmetic*, pp. 274-281, Asilomar, Calif., July 1997.

[23] A.F. Tenca and M.D. Ercegovac, "Synchronous Up/Down Binary Counter for LUT FPGAs with Counting Frequency Independent of Counter Size," *Proc. Int'l Symp. FPGAs*, pp. 159-165, Monterey, Calif., Feb. 1997.

[24] J.D. Ullman, *Computational Aspects of VLSI*. Rockville, Md.: Computer Science Press, 1984.

[25] J.E. Vuillemin, "Constant Time Arbitrary Length Synchronous Binary Counters," *Proc. IEEE Symp. Computer Arithmetic*, pp. 180-183, Grenoble, France, June 1991.

[26] K. Windmiller, "Integrated High Speed Synchronous Counter with Asynchronous Read-Out," U.S. Patent no. 5,045,854, Sept. 1991.

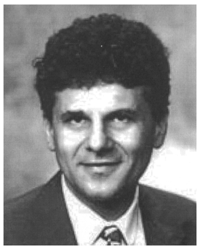
[27] Xilinx, *The Programmable Logic Data Book*. Aug. 1993.

[28] G. Yasar, Y. Tsyrkina, and D. Thygesen, "Fpga Fast Counter Design," *Proc. Canadian Workshop FPDs*, pp. 37-43, Toronto, Canada, May 1996.

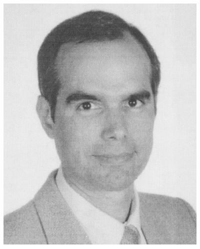
[29] J.R. Yuan, "Efficient CMOS Counter Circuits," *Electronics Letters*, vol. 24, pp. 1,311-1,313, Oct. 1988.

[30] J.R. Yuan and C. Svensson, "Fast CMOS Nonbinary Divider and Counter," *Electronics Letters*, vol. 29, pp. 1,222-1,223, June 1993.

[31] Z. Zilic, G. Lemieux, K.L.S. Brown, and Z. Vranesic, "Designing for High Speed-Performance in cplds and fpgas," *Proc. Canadian Workshop FPDs*, Montreal, Canada, May 1995.



**Mircea R. Stan** (M-94) received the diploma in electronics from the Polytechnic Institute of Bucharest, Romania, in 1984, and the MS and PhD degrees in electrical and computer engineering from the University of Massachusetts at Amherst in 1994 and 1996, respectively. Since 1996, he has been an assistant professor in the Electrical Engineering Department and the Center for Semicustom Integrated Systems at the University of Virginia. Dr. Stan is teaching and doing research in the areas of low-power VLSI, mixed-mode analog and digital circuits, computer arithmetic, and embedded systems. Before his academic career, he accumulated eight years of industrial experience as an R&D engineer in Bucharest, Tokyo, and Atlanta. In 1997, Dr. Stan received the U.S. National Science Foundation CAREER Award for investigating low-power design techniques. He is on the technical committees for the VLSI Design Conference and the International Symposium on Low Power Electronics and Design and is a member of the IEEE, the ACM, and Usenix, and also of Phi Kappa Phi and Sigma Xi.



**Alexandre F. Tenca** received the BS and MS degrees in electrical engineering from the University of São Paulo (USP), Brazil, in 1981 and 1990, and the MS and PhD degrees from the University of California at Los Angeles, in 1994 and 1998, respectively. From 1981 to 1992, he worked as a researcher at USP, developing projects on distributed computer systems, computer network protocols, packet-switching network communication nodes, and control systems. In 1989, he joined the faculty group of the Computer Engineering and Digital Systems Department of Escola Politécnica, USP, where he lectured classes in computer networks and computer architecture until the beginning of his doctoral studies in 1992. Upon completion of his doctorate, he became an assistant professor of computer engineering at Oregon State University, Corvallis. His research interests include computer architecture, computer networks, computer arithmetic, and reconfigurable systems.

**Milos D. Ercegovac** earned his BS in electrical engineering (1965) from the University of Belgrade, Yugoslavia, and his MS (1972) and PhD (1975) in computer science from the University of Illinois, Urbana-Champaign. He is a professor in the Computer Science Department, School of Engineering and Applied Science at the University of California, Los Angeles. Dr. Ercegovac specializes in research and teaching in digital arithmetic, digital design, and computer system architecture. His most recent research is in the areas of arithmetic design for low power and for field programmable gate arrays (FPGAs). His research contributions have been extensively published in journals and conference proceedings. He is a coauthor of a textbook on digital design and of a monograph in the area of digital arithmetic. Dr. Ercegovac has been involved in organizing the IEEE Symposia on Computer Arithmetic. He served as an editor of the *IEEE Transactions on Computers* and as a subject area editor for the *Journal of Parallel and Distributed Computing*. Dr. Ercegovac is a member of the ACM and the IEEE Computer Society.