Synchronous Up/Down Counter with Clock Period Independent of Counter Size

Mircea R. Stan
Electrical Engineering Department
University of Virginia, Charlottesville, VA 22903
mircea@virginia.edu

Abstract

The theory and practice of up-only or down-only prescaled (or **constant time***) counters is well understood both in industry and in the academia. Such counters are obtained by partitioning the counter into sub-blocks in order to be able to anticipate the CARRY propagation inside each block (similar to a carry-select adder). When properly designed, prescaled counters have a clock period independent of counter size. Until now it was not known whether it is possible to design a constant time* **up/down** *binary counter.*

This paper presents the theory behind building a synchronous up/down counter of arbitrary length and with period independent of counter size. The main idea behind the novel up/down counter is to recognize that the only extra difficulty with an up/down (vs. up-only or down-only) constant time 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 inside each sub-block with the purpose of always storing the previous block value. 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.*

A 64-bit up/down counter running at 40MHz was implemented in an Atmel AT6000 FPGA and similar up/down counters can be implemented in any technology.

Keywords - prescalers, up/down counters, constant time counters.

1 Introduction

Counters are basic building blocks in many digital systems with some applications requiring counters that are both fast and long, but speed and size are conflicting qualities because of the carry propagation from low order to high order bits.

By using prescaling techniques it is possible to design counters that are both long and fast, and examples of such counters can be found both in industry (e.g. most FPGA data books [1, 6, 12, 13] have application notes describing prescaled counters with an emphasis on the practical aspects of speeding-up long counters) and in the academia (e.g. [3, 11, 14] where the emphasis is on the theoretical implications of having constant time counters of arbitrary length). The following are desirable qualities for counters [2, 3] which this work tries to satisfy (except for the last one, loadable):

- Clock period independent of counter size. In terms of complexity theory, the clock period of a prescaled counter is constant (i.e. $O(1)$) no matter what the size N of the counter is. This powerful theoretical result is only partially valid in practice because it is based on the "synchronous paradigm" for which the clock is a perfect broadcast signal.

- Readable on the fly with the sampling rate being equal to the counting rate. These are the characteristics of a synchronous design. If a synchronous behavior is not needed (e.g. for frequency dividers) a simpler ripple-carry (asynchronous) counter [11] can be used.

- Space complexity linear in the number of bits (i.e. $O(N)$). If counters with a larger asymptotic complexity are acceptable (e.g. when N is small or when "super fast" counters are needed) ring ("twisted-tail", Johnson or Moebius) counters which have $O(2^N)$ space complexity can be used.

- Binary counting sequence. If a binary sequence is not needed, a simple linear feedback shift register (LFSR) [5, 8] can be used, which has $O(1)$ period and $O(N)$ space complexity, but has a nonbinary output sequence.

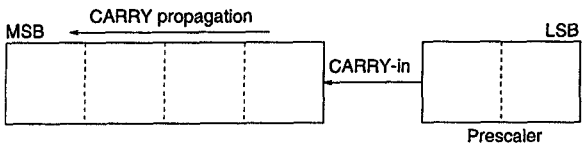

Figure 1. Prescaled counter partitioned into K-bit (here $K = 2$) prescaler and 2^K high-order block.

- Count either "up" or "down" or "up/down". Up-only counter have an increasing output sequence modulo-2^N, down-only counters have a decreasing output sequence modulo-2^N, while up/down counters can change "direction" in any clock cycle under the control of an input signal.
- Loadable with a new value and be able to continue counting from the new value in the next cycle. This is a very difficult property for constant time counters and is not addressed in this work.

Vuillemin [11] posed as an open problem the feasibility of a constant time *up/down* counter and until now many believed that such counters are not feasible (e.g. the Xilinx databook [12] at page 8-68 states that "the ... prescaler technique ... cannot be used in counters that are up/down"). This paper answers this open problem by presenting the design of a constant time up/down synchronous counter. Section 2 briefly presents the main ideas behind previously proposed constant time counters while section 3 describes the novel up/down constant time counter with an example of a 64-bit up/down counter that runs at 40MHz in an Atmel FPGA.

2 Constant time up-counters

Prescaling long counters requires partitioning them into a series of sub-blocks of increasing sizes. 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 like in figure 1 [7]. What makes a prescaled counter work is the fact that, due to the characteristics of the binary number system, the CARRY-in 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 (e.g. for a K-bit prescaler, the CARRY-in will have a frequency 2^K times lower than the clock frequency). In this way, the "virtual clock frequency" for the slow high-order block is 2^K 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 [3, 11]. For higher order blocks, successive CARRY-ins from the previous stages become exponentially farther apart in time, hence higher order blocks can have exponentially increasing sizes and for all practical purposes 3 or 4 such partitions are typically enough [11]. 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 (1 or 2 bits).

The following two issues arise in the practical design of constant time counters and determine the sub-block sizes:

- 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 prescaled generation of the CARRY-in to a partition has to be synchronous with the true clock. Several different approaches have been proposed for the prescaled generation of the CARRY-in to high order partitions. The first proposed solution, by Ercegovac and Lang [3], uses a (relatively inefficient) ring counter, which practically doubles the overall complexity of the counter. A much simpler CARRY-in generation, proposed by Vuillemin [11], uses a combinational chain that takes further into account the characteristics of the binary number system.

2.1 Partitioning the counter for a constant time behavior

The size of each sub-block must be chosen such that the CARRY propagation inside the block is shorter than the delay between two successive CARRY-ins 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.

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

- In a *top-down* manner [3] 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 $(N - \lceil \log N \rceil)$ most significant block, and into another $\lceil \log N \rceil$ block

which is recursively partitioned in the same manner [3]. For example, in the case of a 64-bit counter, a top-down partitioning results in the following block sizes: 58, 3, 2, 1 [3]. 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 difficult to implement. For a 128-bit counter the top-down partitioning leads to: 121, 4, 2, 1 block sizes.

- In a *bottom-up* manner [11] 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 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 [11]. 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.

2.2 Other types of counters

Being able to design counters with $O(1)$ period is non-intuitive considering that adders have a $O(\log N)$ period and incrementing is a special case of addition. The following observations give a justification for why constant time counters are feasible:

1. the binary number system has periodicity in the way the CARRY-in to high order bits is generated, which makes it both predictable and with a low frequency [11],
2. the black-box model of a non-loadable counter (as opposed to an adder) has only a limited number of inputs: Clock, Reset and Count Enable (CE). This explains why the binary tree logic decomposition which leads to the $O(\log N)$ delay for an adder can be circumvented for counters.

Considering other types of counters, 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 [12]). Unlike up-only and down-only counters, loadable counters and up/down counters do *not* exhibit the nice periodicity and predictability of the CARRY-in (or BORROW-in) to high order blocks [12]. After a *load*, a loadable counter cannot guarantee enough time for CARRY propagation inside the sub-blocks, while an up/down counter can reverse direction at any moment, 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 (Clock, Up/Down (U/D), Reset and CE) 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 [9, 10], while there have been several reported techniques (e.g. "pulse swallowing" and "state skipping" [12]) that enable a loadable counter to have a quasi-constant time behavior by letting the counter output to be out of sequence for a period of time after loading.

In the following section we describe our design of a non-loadable constant time up/down counter.

3 Constant time up/down counter

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 CARRY-in (or BORROW-in) 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.

The block diagram of the proposed up/down constant time counter is shown in figure 2. The design is synchronous, with a Clock active on the rising edge, a Reset active HI and an Up/Down input which is LO for counting up and HI for counting down. If desired, a separate Count Enable (CE) can be easily added by gating the Clock, or by AND-ing CE 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 CARRY-in generation must be itself

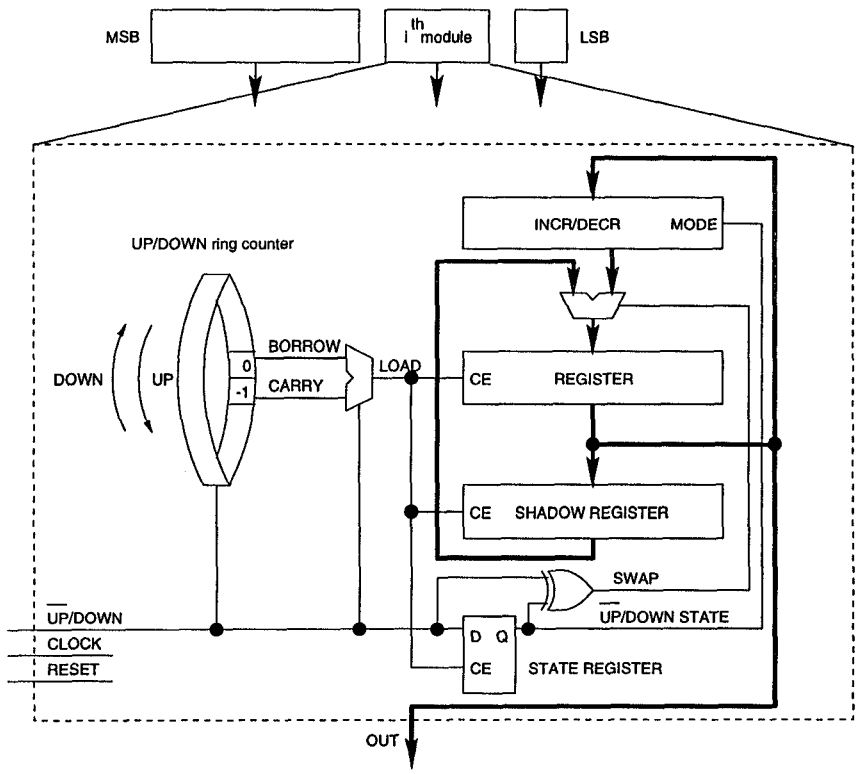

Figure 2. Block diagram of the constant time up/down counter

up/down, hence it can be implemented as an up/down ring counter similar to the ring counter proposed by Ercegovac and Lang [3]. This also implies that a top-down partitioning method [3] is needed in order to minimize the size of the ring counters. Unfortunately, a combinational chain as proposed by Vuillemin [11] 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 sub-block 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 sub-blocks in this design function practically independent of each other, the ring counter inside each block effectively replacing the need for receiving the CARRY-in from lower-order blocks. The complexity of the up/down counter is approximately twice larger than for an up-only counter as in [3] and four times larger than in [11] because of the extra shadow register and the configurable CARRY chain.

3.1 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. 3) 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.

3.2 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 UP/DOWN input signal is 0 (UP), while a BORROW-in can occur only if the 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

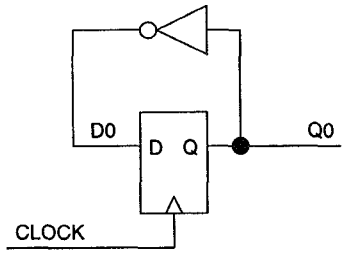

Figure 3. The least-significant bit block.

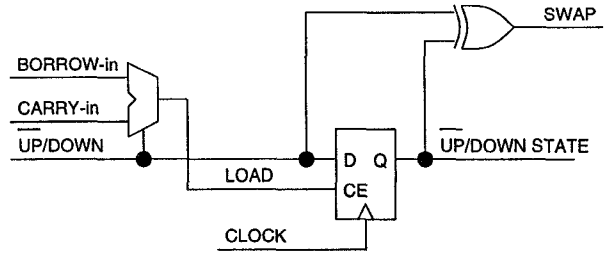

Figure 4. The configuration bit inside each block

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 somehow 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 register with a Clock Enable (CE) input (figure 4) and a Reset (not shown). 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 in "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 with CE and Reset as the configuration bit.

3.3 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 two (or more, depending on the granularity of the basic cell) unit delays (instead of 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 basically a bidirectional shift register.

- Loading (actually "swapping") the value of the block with the value of the shadow register requires a unit

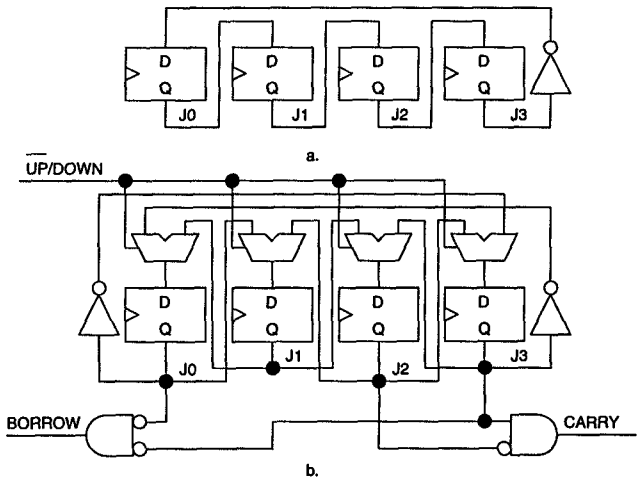

Figure 5. 4-bit (8 states) ring counters (Johnson or Moebius): a.) up-counter, b.) up/down counter.

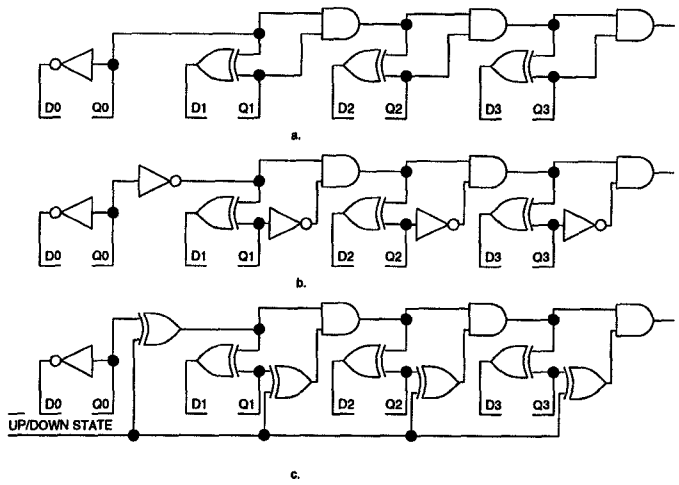

Figure 6. Ripple chain: a. Incrementer (CARRY chain), b. Decrementer (BORROW chain), c. Incrementer/Decrementer.

delay through a mux, and in parallel, the mux control signal requires two unit delays (see figure 2). The timing of the Up/Down (U/D) 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 is not 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 limited by the extra logic needed for swapping with the shadow register. In a real design the clock period can be even larger than two unit delays due to particular implementation details and fan-out effects on long lines.

3.4 Up/down ring counter

A 2^k-bit twisted-tail ring counter (also known as a Johnson or Moebius counter) has $2^{(k+1)}$ distinct states and clock period independent of size, hence it can function as a $(k+1)$-bit counter prescaler [3] (see figure 5 a.) 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 figure 5 b. The ring counter inside each block is used in order to generate in constant time the CARRY-in (or BORROW-in) for the block. A CARRY-in must be generated when the least significant bits before the block are all 1, which in terms of the ring counter translates into the most significant bit = 1 and all other bits = 0. A BORROW-in must be generated when the least significant bits before the block are all 0, which in terms of the ring counter translates into all bits = 0. Both the condition for CARRY-in and for BORROW-in can be easily decoded in constant time by analyzing only 2 bits of the ring counter (this is a general property of the ring counter if the contents is a legal value with a run of all-zeros and a run of all-ones).

3.5 Partitioning

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

3.6 Incrementer/decrementer

The Incrementer/Decrementer can be easily implemented as a CARRY-ripple chain (see figure 6). For a 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

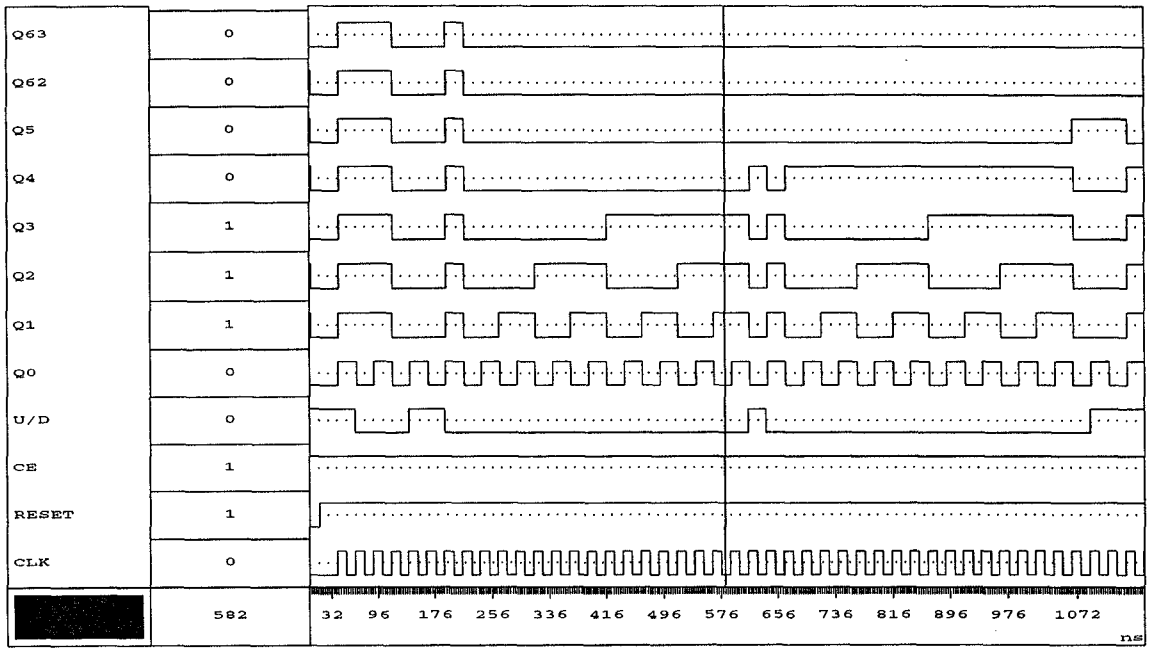

Figure 7. Simulation of the 64-bit up/down counter at 40MHz. Only the 6 least-significant and the 2 most-significant bits are shown.

two consecutive CARRY-ins (or BORROW-ins) from the ring counter (which is always true by partitioning) the Incrementer/Decrementer is not on the critical path. The configuration is controlled by the configuration bit for each block.

3.7 Initializing the counter

For a "standard" N-bit counter which has 2^N possible states, all the states are legal and a Reset signal may not be needed for some applications (although desirable at least for testing). In the case of the proposed constant time up/down counter which has many extra registers (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.

3.8 FPGA implementation

We have implemented the design of a 64-bit up/down counter partitioned into three modules with 60, 3, 1 bits that runs at 40MHz in an Atmel FPGA. The partitioning has resulted from the minimum clock period which is $p = 4$ times the minimum combinational delay due to the simple cells in the chosen device. The Atmel SRAM-based AT6000 FPGAs are 2-dimensional arrays of relatively fine-grained simple logic cells which can implement a limited number of registered or combinational functions with up to 3 inputs and 2 outputs. Each cell has direct connections with its 4 neighboring cells and there are also local and express connections for routing to distant cells. The cell registers have a limited number of features and the ones that influenced the design are:

- the registers have only true (Q) outputs,
- there is no clock enable on the registers but registers with enable are offered as macros in the design library,
- registers only have a RESET but no SET input.

A functional simulation at 40MHz of the 64-bit up/down counter is shown in fig. 7. As can be seen the counter counts up and down and can change direction each cycle.

4 Conclusion

We presented a methodology behind designing synchronous up/down counters of arbitrary length with period independent of counter size. Recently there has been a report of a related design implemented in a LUT-based FPGA by

Tenca and Ercegovac [10]. Instead of the "shadow register", the design in [10] stores the values on the CARRY (or BORROW) chain in order to recover the correct previous value in case the block changes direction. This slightly complicates the configuration state machine inside the block.

An example of a 64-bit counter that runs at 40MHz in an Atmel AT6000 FPGA shows a practical implementation of our methodology. It should be relatively easy to migrate the design to a different technology 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. This only makes sense when relatively long (more than 24 bits) up/down counters are needed. Better (faster or simpler) results can be probably obtained for short counters with other approaches which are not constant time but are better for small numbers.

From the point of view of state machines with a large number of states but with a limited number of inputs, the fact that counters can be constant time becomes less intriguing considering that $O(1)$ state machines with a state transition graph (STG) similar to a binary counter are well known: LFSRs, Johnson counters, etc. It would be interesting to be able to determine when a state machine can have period $O(1)$ just by looking at the STG and the state encoding. Even more interesting would be to determine a state encoding that enables a $O(1)$ period for a given STG if such an encoding exists.

5 Acknowledgements

I thank Joel Rosenberg of Atmel Corp. for an FPGA development tools grant and professors Israel Koren and Wayne Burleson from UMass/Amherst for inspiring my interest in Computer Arithmetic.

References

[1] "Configurable logic design and application book", Atmel, 1994-1995.

[2] D. Chu, "Phase digitizing sharpens timing measurements", *IEEE Spectrum*, July 1988, pp. 28-32.

[3] M. Ercegovac, T. Lang, "Binary counter with counting period of one half adder independent of counter size", *IEEE Trans. on Circ. and Systems*, vol. 36, no. 6, June 1989, pp. 924-926.

[4] I. Koren, "Computer Arithmetic Algorithms", Prentice-Hall, 1993.

[5] W. W. Peterson, "Error Correcting Codes", MIT Press, 1961.

[6] "Very high speed FPGAs data book", QuickLogic, 1992.

[7] R. Rogenmoser, Q. Huang, F. Piazza, "1.57GHz asynchronous and 1.4GHz dual-modulus 1.2 μm CMOS prescalers", *IEEE Custom Integrated Circuits Conference*, 1994.

[8] M. R. Stan, "Shift register generators for circular FIFOs", *Electronic Engineering*, Feb. 1991, pp. 26-27.

[9] M. R. Stan, W. P. Burleson, "Synchronous up/down counter with period independent of counter size", poster at FPGA'96 - ACM/SIGDA Symposium on Field Programmable Gate Arrays, 1996.

[10] A. F. Tenca, M. D. Ercegovac, "Synchronous up/down binary counter for LUT FPGAs with counting frequency independent of counter size", FPGA'97 - ACM/SIGDA Symposium on Field Programmable Gate Arrays, pp. 159-165, 1997.

[11] J. E. Vuillemin, "Constant time arbitrary length synchronous binary counters", *Proc. IEEE 10th Symp. on Comp. Arithmetic - Grenoble, France*, June 26-28, 1991, pp. 180-183.

[12] "The programmable logic data book", Xilinx, 1994.

[13] G. Yasar, Y. Tsyrkina, D. Thygesen "FPGA fast counter design", 4^{th} *Canadian Workshop on Field-Programmable Devices*, May 1996, Toronto, Canada.

[14] Z. Zilic, G. Lemieux, K. Loveless, S. Brown and Z. G. Vranesic, "Designing for High Speed-Performance in CPLDs and FPGAs", 3^{rd} *Canadian Workshop on Field-Programmable Devices*, June 1995, Montreal, Canada.