Synchronous Up/Down Counter with Period Independent of Counter Size

Mircea R. Stan, Wayne P. Burleson
mstan@risky.ecs.umass.edu

Abstract— **The theory and practice of up-only (or down-only) prescaled counters is well understood both in industry and in the academia but until now it was not known if the design of a prescaled up/down binary counter is possible. This paper presents the theory behind building such a synchronous up/down counter of arbitrary length and with period independent of counter size by describing the design of a 64-bit up/down counter running at 40MHz implemented in an Atmel AT6000 SRAM-based FPGA. The main idea behind the novel up/down counter design is to recognize that the only extra difficulty with an up/down (vs. an up-only or down-only) constant time counter is when the counter changes "state" from counting up to counting down and vice-versa. For dealing with this difficulty the new design uses a "shadow" register that is always kept loaded with the previous counter value. When counting only up or only down the counter functions like a normal up-only or down-only prescaled counter but when it changes "state" from up to down or from down to up instead of trying to compute the new counter value and having to wait for carry propagation it simply swaps with the shadow register which contains the desired previous value. An Atmel AT6000 (former Concurrent Logic CLi6000 family) FPGA was used in this design but similar up/down counters can be implemented in any other technology or FPGA family.**

Keywords— **Prescaled counter, constant time counter, up/down counter.**

I. Introduction

COUNTERS are a basic building block in many digital designs and for some applications counters that are both fast and long are needed. Speed and size are generally conflicting qualities because carries from lower order bits have to propagate to higher order bits and if the counter is long the carry path becomes longer.

By using prescale 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], [9] have several application notes describing prescaled counters with an emphasis on the practical aspects of speeding-up long counters) and in the academia ([3], [8] where the emphasis is on the theoretical implications of having constant time counters of arbitrary length).

The qualities that make prescaled synchronous counters extremely useful are [2], [3]:

- Clock period theoretically independent of counter size. In terms of complexity theory the clock period of a prescaled counter is $O(1)$ no matter what the size N of the counter is. This powerful theoretical result will be only partially valid in practice because it depends on the synchronous paradigm for which the clock is a perfect broadcast signal.
- Can be read on the fly and the sampling rate is equal to the counting rate. These are the characteristics of a synchronous design. If these qualities are not needed (e.g. in frequency dividers) a much simpler "ripple-carry" counter [8] can be used.
- Space complexity $O(N)$. If counters with a larger asymptotic complexity are acceptable (e.g. when N is small or when "super fast" counters are needed) ring counters or Johnson counters with $O(2^N)$ space complexity can be used.

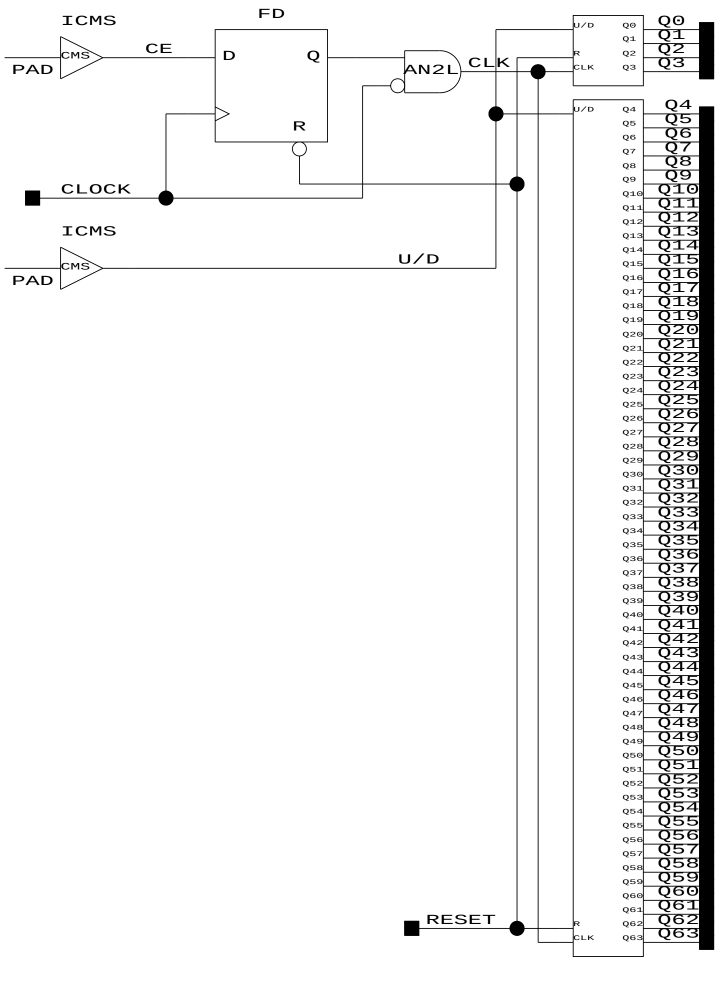

Fig. 1. Prescaled 64-bit counter. As any prescaled design this counter is partitioned into several (in this case three) submodules of exponentially increasing sizes (in this case 1, 3 and 60 bits).

This work was supported in part by a development tools grant from Atmel Corp.

The authors are with the Department of Electrical and Computer Engineering, University of Massachusetts at Amherst, MA 01003

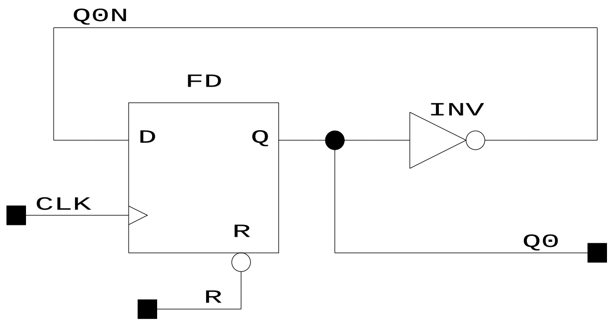

Fig. 2. The first prescaler stage. This 1-bit counter can be used as a prescaler for up-only, down-only or up/down counters.

- Binary counting sequence. When this is not needed a simpler linear feedback shift register (LFSR) [5], [7] with the same $O(1)$ period and $O(N)$ space complexity but with nonbinary output sequence can be used.

Until now it was not known whether it is possible to design a prescaled up/down counter (in [9] at page 8-68 it is stated that "the ... prescaler technique ... cannot be used in counters that are up/down" while [8] concludes more cautiously by leaving this as an open problem).

Section II briefly presents the main ideas behind standard prescaled counter design while section III describes the novel up/down prescaled counter technique with an example of a 64-bit up/down counter that runs at 40MHz.

II. Prescaled up-only counters

Prescaling techniques speed up long counters by first partitioning them into a series of submodules of exponentially increasing sizes like in fig. 1. Historically the first prescaled counters had only two such partitions with a small and fast least significant module called the prescaler and a slow and large counter for high-order bits. What makes a prescaled counter work is the fact that the CARRY out from the low-order prescaler (the moment when the high order bits have to increment) has a low frequency (e.g. for an N-bit prescaler the CARRY out will have a frequency 2^N times lower than the main clock frequency) and in this way the "virtual clock frequency" for the slow high-order partition will be smaller than the true clock frequency. This means that the CARRY propagation *inside* the high-order partition can take a long time even with a very fast clock.

A simple reasoning leads to a theoretical unlimited extension of the counter size without affecting the clock period [3], [8] by adding more and more partitions. For higher order partitions the CARRY in from the previous stages becomes exponentially farther apart and so higher order partitions can also increase exponentially in size. For all practical purposes probably 3 or 4 such partitions are enough. In a correctly designed constant time counter the clock period is limited by the speed of the least significant partition and this is why this should be designed as fast as possible. This generally results in a very small (1 or 2 bits) first prescaler like in fig. 2.

There are two distinct issues that arise in the practical design of constant time counters and the way these are implemented determines how the long-counter partitioning is done:

- the CARRY propagation *inside* a module,
- the prescaled generation of the CARRY in to a high order module.

Since it is desired that the design be as simple as possible generally a ripple CARRY propagation inside each module is chosen but several different approaches have been proposed for the prescaled CARRY generation for higher order modules. This can be done either with a (relatively inefficient) ring counter (as in [3]) or with a much simpler combinational chain that takes into account once more the characteristics of the binary number system (as in [8]). Depending on how this prescaling is implemented the sizes of the individual modules of a long counter can be decided:

- top-down (as in [3]) by first determining the size of the last module which is chosen as large as possible and then recursively determining the sizes of the other modules. This top-down decision minimizes the penalty paid for having relatively large ring counter prescalers but has the disadvantage that counters of different sizes will result in different partitions and a different design will be necessary for each counter size.
- bottom-up (as in [8]) by first deciding the size of the least significant partition, then choosing a second partition as large as possible without affecting the clock period, then choosing the third, etc. This bottom up decision has the advantage of potentially using the same modules as building blocks for counters of different sizes and just "truncating" the most significant module.

The theoretical implications of being able to have a constant time ($O(1)$ period) counter independent of size are very powerful considering that an adder (the counter adds 1 each cycle) has a $O(\log N)$ period. The $O(1)$ period counter is possible because of two characteristics:

1. the periodicity of the binary number system which results in the fact that the CARRY in to high order bits is predictable and has a low frequency,
2. when viewed as a "black-box" a non-loadable counter (as opposed to an adder) has only a very limited number of inputs: a Clock Enable (CE) and a clock. This is why the standard binary tree logic decomposition which explains the $O(\log N)$ adder period can be circumvented.

Down-only counters have the same above characteristics and designing a down constant time counter is exactly the same as designing an up counter and only changing the CARRY chain with a BORROW chain (practically this is done by inverting the inputs to the AND gates that compute the chain [9]). On the other hand loadable counters and up/down counters do *not* exhibit the above periodicity and predictability of the CARRY in to higher order modules [9]. After a load a loadable counter does not guarantee enough time for CARRY propagation and an up/down counter can reverse direction at any time which again does

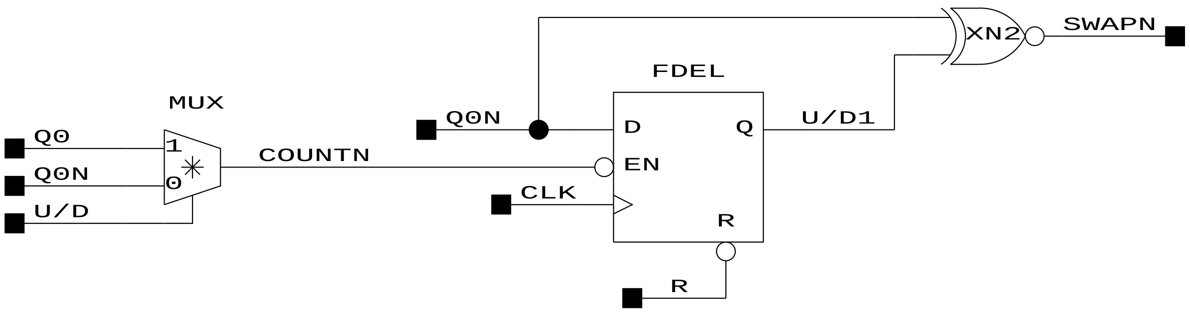

Fig. 3. The "state" bit for the 3-bit stage of the up/down counter. The U/D1 output configures the 3-bit counter stage as an up-counter (when U/D1 is LOW) or as a down-counter (when U/D1 is HIGH). Loading of this state bit is enabled only when there is a CARRY or a BORROW from the 1-bit prescaler.

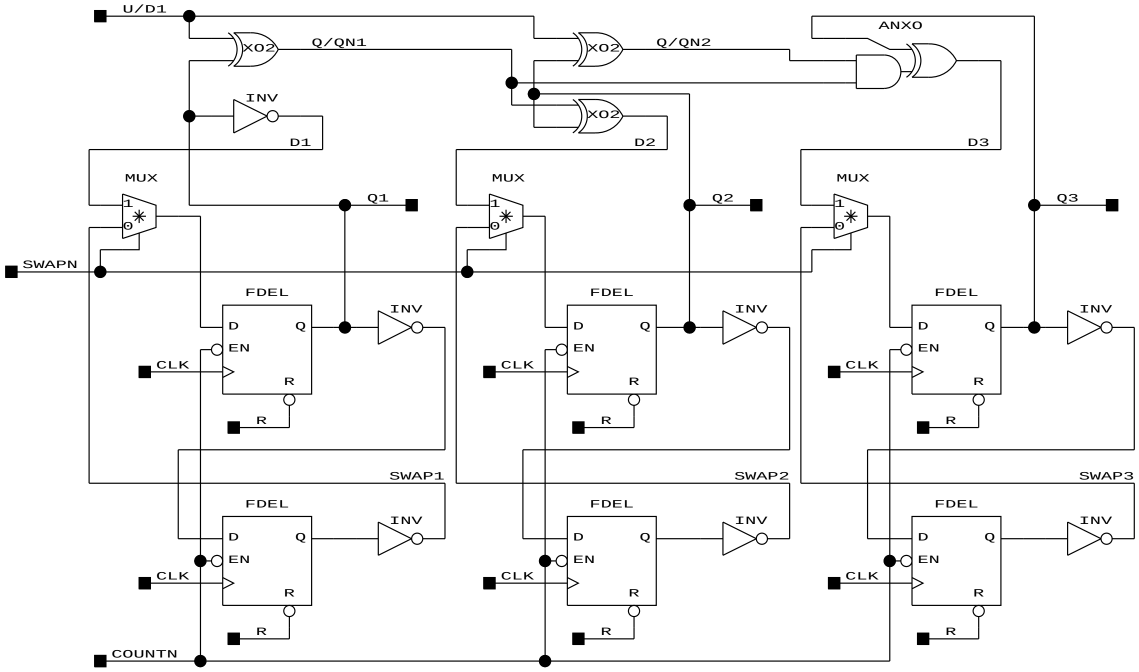

Fig. 4. The 3-bit stage of the up/down counter. The top XOR and AND gates represent the CARRY (or BORROW depending on the U/D1 state bit) chain. The middle row of registers represents the counter bits while the lower row of registers keeps the "shadow" bits.

not let enough time for CARRY propagation. It is interesting to note that although loadable counters have a large number of input lines, up/down counters still have only an extra Up/Down (U/D) input. There are reasons then to believe that it is more likely to be able to design a constant time up/down counter than a constant time loadable counter. It is somehow ironic from this point of view that until now no constant time up/down counter was designed but there are several techniques (e.g. "pulse swallowing" and "state skipping" [9]) that enable a loadable counter to have constant time independent of counter size by letting the output be out of sequence for a period of time after loading.

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

III. Prescaled 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 (when configured as an up counter it will have a CARRY chain and when 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 up/down counter changes "state" from counting up to down and vice-versa. This change of state is the only

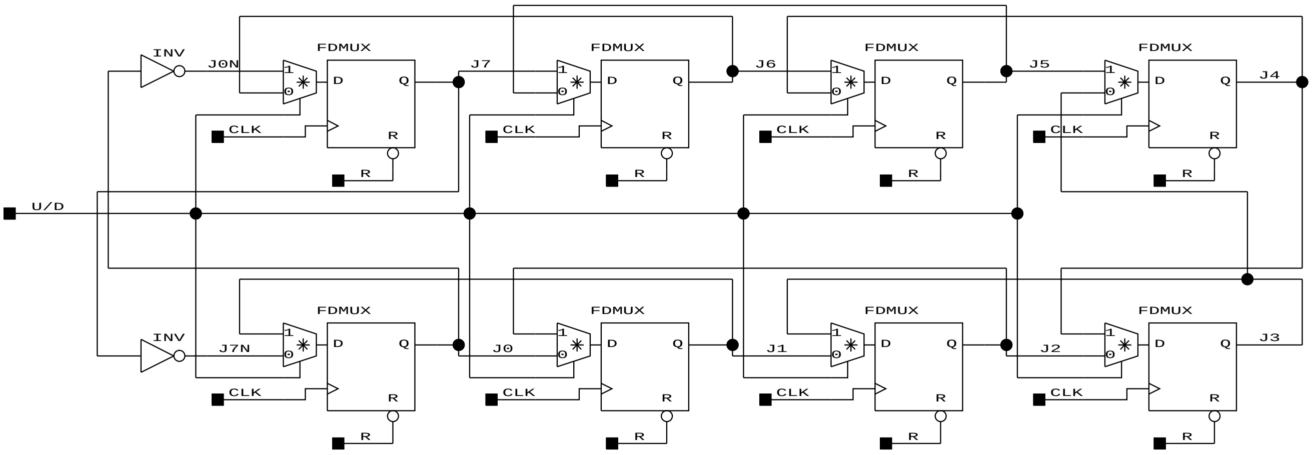

Fig. 5. 8-bit up/down Johnson prescaler for the third (60-bit) stage. This ring-shaped drawing is reminiscent of the physical layout of the Johnson counter in the FPGA. This 8-bit Johnson counter has 16 states and is equivalent to a 4-bit up/down binary counter.

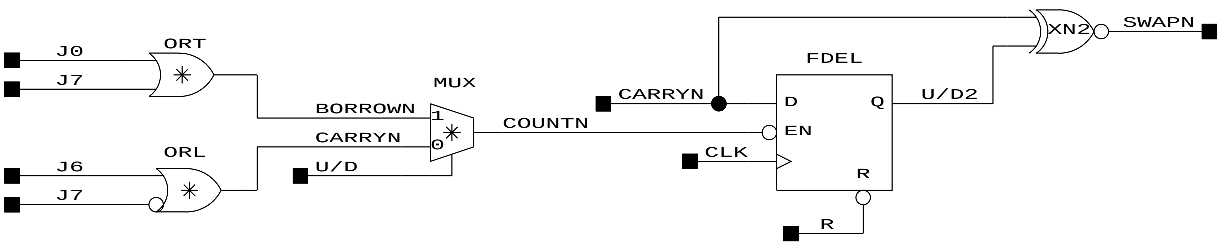

Fig. 6. The "state" bit for the 60-bit stage of the up/down counter. The U/D2 output configures the 60-bit counter stage as an up-counter (when U/D2 is LOW) or as a down-counter (when U/D2 is HIGH). Loading of this state bit is enabled only when there is a CARRY or a BORROW from the 8-bit Johnson prescaler.

moment when the CARRY chain in a module potentially has no time to propagate until the next CARRY in from the prescaler. The solution is then to have the desired value "precomputed" and simply load it instead of computing it. This can be easily done by using a "shadow" register that is always loaded with the "previous" module value. A state bit keeps track of how the module is configured (up or down). If the module was previously configured "up" and a BORROW comes or previously configured "down" and a CARRY comes from the prescaler, the module, instead of trying to compute the new value, it simply swaps with the value of the shadow register. Otherwise, if configured "up" with a CARRY coming or "down" with a BORROW coming from the prescaler, the counter will behave like a standard up-only or down-only prescaled counter with the only extra step of allways loading the shadow register with the previous value each time the module increments or decrements.

The issues that arise in designing this up/down constant time counter are then:

- the prescaler must be itself an up/down prescaler and can only be implemented as a ring counter or something similar as in [3]. For an up/down counter combinational prescaler chains like in [8] do not work because the counter has to be able to change direction each cycle. This also means that a top-down partitioning method similar to [3] is needed in order to minimize the size of the ring counter prescalers.

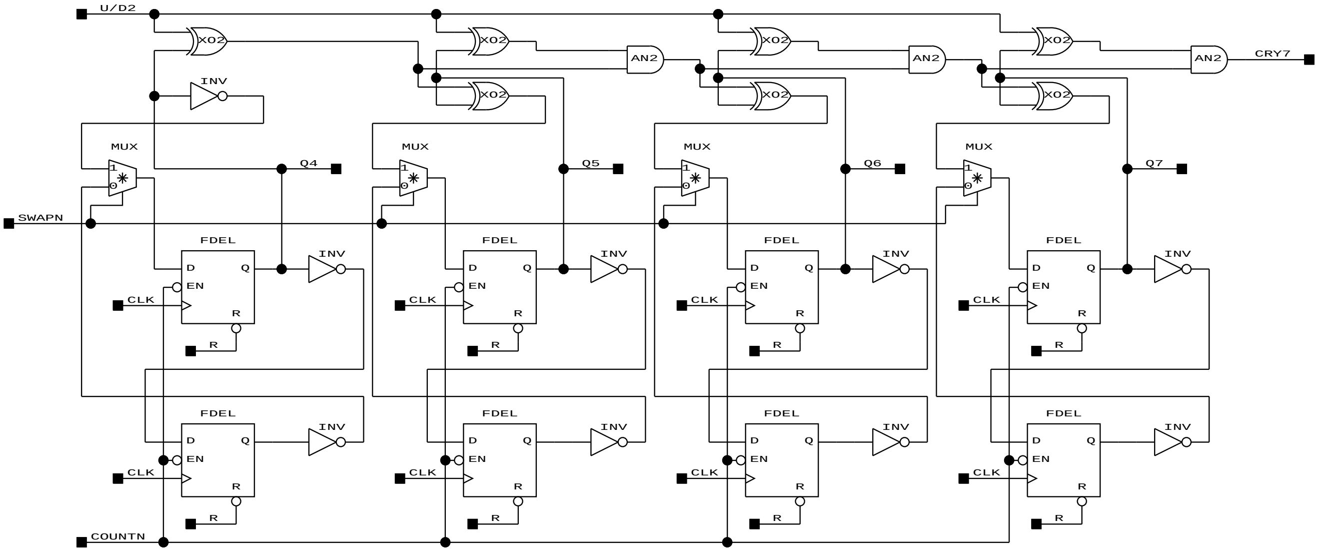

Fig. 7. The least significant 4 bits (Q4 through Q7) of the third counter stage. The upper XORs and ANDs represent the CARRY chain, the middle row of registers represents the counter bits while the lower row of registers keeps the "shadow" bits.

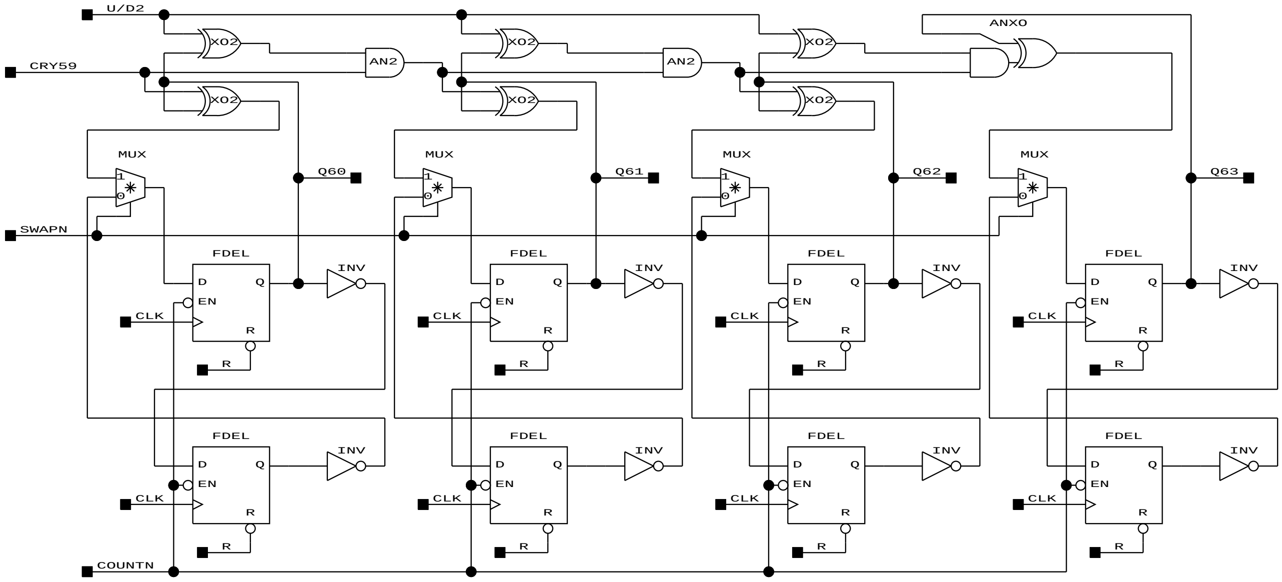

Fig. 8. The most significant 4 bits (Q60 through Q63) of the third counter stage. The upper XORs and ANDs represent the CARRY chain, the middle row of registers represents the counter bits while the lower row of registers keeps the "shadow" bits.

- each module will need to be configurable for counting either up or down. A separate state bit for each module will be needed.
- each module will have a shadow register that will store the previous module value (previous means decremented or incremented by one module LSB depending on the state of the module). When the module state is "up" the shadow will keep the present value minus one LSB and when the state is "down" it will keep the present value plus one LSB.

The complexity of the up/down counter is approximately twice that of an up-only counter because of the extra shadow register and the configurable CARRY chain. The clock frequency is independent of counter size but is lower than for an up-only counter because of the extra complexity. Instead of only being limited by how fast the low order prescaler is the speed is also limited by the extra logic needed for swapping with the shadow register. The fact that the clock is slower means the size of the higher level modules grows even faster.

We have implemented the design of a 64-bit up/down counter partitioned into three modules with 1, 3, 60 bits that runs at 40MHz in an Atmel FPGA (see fig. 1). The global CE was implemented by simply gating the global clock in order to keep the overall design simple. This is the only place where the clock is gated and in case this gating is not desired it is relatively easy to implement the CE without gating the clock by slightly complicating the design. 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 (e.g. like in the Xilinx FPGAs [9]) but registers with enable are offered as macros in the design library,
- registers only have a RESET but no SET input.

We have chosen to implement the synchronous module prescalers as up/down Johnson counters instead of the ring counters of [3]. Johnson counters are half the size of a ring counter with the same number of states.

The first module (see fig. 2) is a simple 1-bit counter which acts both as a 1-bit up/down counter and as a prescaler for the next 3-bit module.

The 3-bit module consists of a state register (fig. 3), a 3-bit shadow register, a 3-bit present value register and a configurable CARRY/BORROW chain (fig. 4). After RESET the module is in the "up" state (U/D1 = 0), the present value is all-zeros and the shadow register is all-ones (present value minus LSB). Because there are no registers with asynchronous SET in the Atmel FPGAs the shadow register needs to use inverters on inputs and outputs (see fig. 4) in order to be initialized with all-ones.

The 3-bit module is normally enabled only every other clock cycle by the signal COUNTN (active LOW, see fig. 3) when there is a CARRY (U/D = 0 and Q0 = 1) or a BORROW (U/D = 1 and Q0 = 0) from the prescaler.

When COUNTN = 0 (enable count for this module) and the state register does *not* change state then SWAPN (active LOW) is not asserted (SWAPN = 1) and the counter simply counts up or down by loading the value of the ripple-carry chain. The chain is configurable by conditionally (depending on the state bit U/D1) inverting the inputs to the AND chain. Each time COUNTN = 0 the module will count (Q1, Q2, Q3 will increment or decrement) and the previous value will also be loaded into the shadow register.

When COUNTN = 0 (enable count for this module) and the state register *does* change state then SWAPN gets asserted (SWAPN = 0) and the counter registers swap their value with the shadow register.

The third 60-bit module works almost identical to the 3-bit module. A difference is that an 8-bit up/down Johnson counter (see fig. 5) is needed as a prescaler for this module. The state bit for this module (see fig. 6) is similar to the one for the 3-bit module. The main module register, shadow register and CARRY chain are also similar to the 3-bit module. Only the least significant 4 bits (fig. 7) and most significant 4 bits (fig. 8) are shown.

There is a somehow subtle point in realizing that the state bits for the modules can be different at times. This happens for example when after the counter counts up for a number of cycles it then changes "direction" by only changing lower order bits. In such a case the higher order modules will still be configured "up" and they will only change state if a BORROW comes from the prescaler. If the counter again changes direction before that BORROW comes the higher order module will never "know" (or "care") that the lower order modules were in a different state for a period of time.

A functional simulation at 40MHz of the 64-bit up/down counter can be seen in fig. 9. As can be seen the counter counts up and down and can change direction every cycle.

IV. Conclusion

We presented the methodology behind designing synchronous up/down counters of arbitrary length with period independent of counter size. An example of a 64-bit counter that runs at 40MHz in an Atmel AT6000 FPGA gives details of a practical implementation of the methodology. 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.

When looked upon as 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

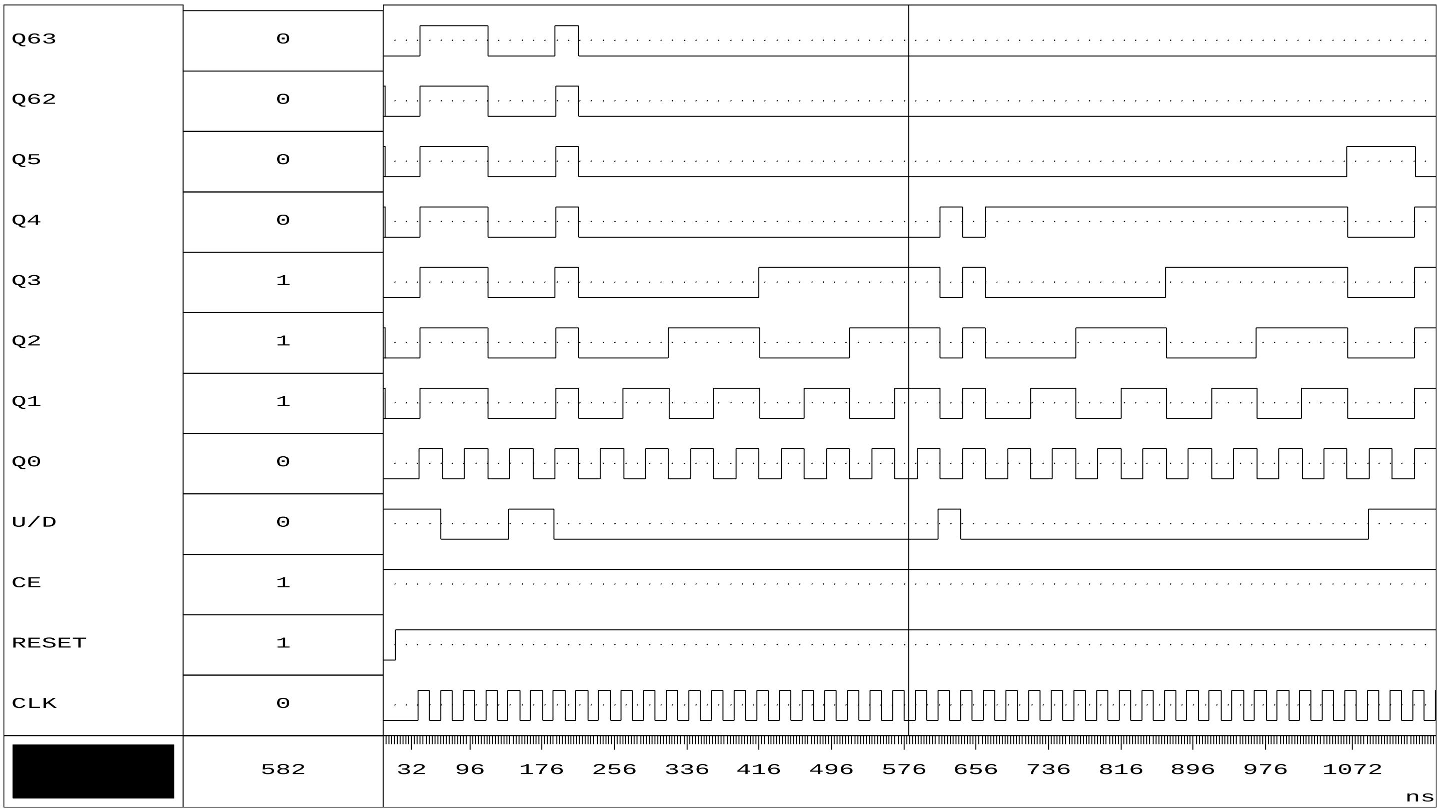

Fig. 9. Functional simulation at 40MHz of the 64-bit up/down counter. Only the 6 least-significant and the 2 most-significant bits are displayed. The Count Enable (CE) input is always kept HIGH, the asynchronous RESET is used for initialization and the Up/Down (U/D) input is active LOW for counting up and active HIGH for counting down.

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 would enable a $O(1)$ period for a given STG.

V. Acknowledgement

I thank Joel Rosenberg of Atmel Corp. for an FPGA development tools grant.

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, 1992.
[5] W. W. Peterson, "Error correcting codes", MIT Press, 1961.
[6] "Very high speed FPGAs data book", QuickLogic, 1992.
[7] M. Stan, "Shift register generators for circular FIFOs", *Electronic Engineering*, Feb. 1991, pp. 26-27.
[8] 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.
[9] "The programmable logic data book", Xilinx, 1994.