Synchronous Up/Down Binary Counter for LUT FPGAs with Counting Frequency Independent of Counter Size

Alexandre F. Tenca
tenca@cs.ucla.edu

Miloš D. Ercegovac
milos@cs.ucla.edu

Computer Science Department
University of California, Los Angeles

Abstract

This paper presents the design of a fast up/down binary counter for LUT FPGAs. The counter has a cycle time independent of the counter size. The key aspects of the design are described and applied to a 64-bit synchronous binary counter implemented in a XC4010 FPGA chip. Experimental results show that the counter can scale up to hundreds of bits while keeping a short cycle time.

1 Introduction

Counters are very common in many digital circuits. The basic synchronous modulo-2^n up counter structure has an incrementer and a state register. The incrementer generates the next state that is stored in the state register when a count signal (cnt) is active. The state transition implemented by an up-counter is:

$$s(t+1) = \begin{cases} s(t) & \text{if } cnt = 0 \\ (s(t)+1) \bmod 2^n & \text{if } cnt = 1 \end{cases} \quad (1)$$

where $s(t)$ is the counter state at time t.

The incrementer used to obtain $(s(t)+1) \bmod 2^n$ can be implemented in many different ways. In this paper we consider the simplest case, where the circuit is organized as a chain of Half-Adders (HA). The HA has only one gate in the path to generate the carry or sum bit. The delay of such a circuit increases linearly with the length, in bits, of the number to be incremented (in this case n). So, for large counters, the delay to generate the next state becomes unacceptable.

Ercegovac and Lang in [1] describe an implementation method that partitions a large counter into smaller ones (sub-counters). Each sub-counter has a circuit that, at the proper time, enables the sub-counter to change state. The partitioning is made such a way that the delay of the incrementer of a sub-counter is accommodated by the counting period of the sub-counter assigned to least significant bits. For example: if we have two sub-counters, one of n bits and another of m bits, such that the state is composed by the concatenation of these bits, the value represented by the most significant n bits is incremented in periods of 2^m clock cycles, when all m bits go to zero. If $n \leq 2^m$, there is sufficient time for carry propagation in the HAs' chain that generates the next state in the n-bit sub-counter, before the condition to change the state of this sub-counter is reached. We describe the partitioning algorithm later. Using this partitioning method, the cycle time of the counter can be made as low as one gate delay, independent of the length of the counter. Theoretically, the method presented in [1] has no limit (except for broadcasting of control signals).

A scheme presented by Vuillemin [3] uses the same idea of counter partitioning, but instead of using independent sub-counters, he combines the carries generated by each least significant sub-counter to obtain the count enable of the leftmost sub-counter. The length of each sub-counter is adjusted (reduced) in order to absorb the delay caused by the combination of carries. The advantage of the approach is to use fewer flip-flops – FFs – (since there is no circuit to enable the load of the next state in each sub-counter) but the cycle time is limited to at least 2 gate delays.

The previous schemes present the counter delay as a function of standard gate delays. In this work we present the delays in terms of *Function Generators* of the XC4000 LUT FPGA. We assume the reader is familiar with the organization of Xilinx FPGAs. The delays are referenced in this paper as FMAP delay, for F or G function generators only, or FHMAP delay, for the case of the delay involved with F and H function generators in series. These values are given in the Xilinx manual [5].

The Xilinx Data Book [4] shows several counters for the XC4000 FPGAs. The fastest design uses *prescaler* technique [4, 2]. An example shows a 16-bit counter with a clock frequency of 111Mhz. The author of the design points out that the length of the counter can theoretically go up to 87 bits, with the same cycle time, but is really limited by the broadcast of control signals. The practical number of bits is 23. The counter also doesn't have a count input. The counter makes use of the fast carry logic available in the device. The area used is about 1 CLB per bit. CLB stands

for Configurable Logic Block.

All of these designs consider only the up-counting case, and in particular, in [3] the following question is proposed: *"is it possible to design a synchronous, arbitrary length, constant time up-down counter?"*.

Designs of up/down counters found in [4] show clock cycle times that increase significantly with the size of the counter. This paper presents a design of an up/down counter that has a clock cycle time independent of the counter size [1].

Reasons for long counters are presented in [1, 3]. This paper is organized as follows: initially, we present a short discussion of the constant time variable size up-counter proposed in [1], the next section presents an extension of the original design to allow up/down counting, and in the last section some experimental results are discussed.

2 Constant time variable size up-counter

Based on the fact that the worst case delay of a counter is caused by the incrementer circuit and that the delay is linearly related to the counter size, Ercegovac and Lang [1] proposed a design methodology for an up-counter that recursively constructs the counter by breaking it into sub-counters. A n-bit counter M is broken into M_1 (most significant) and M_2, such that M_1 is a $(n - \lceil log_2 n \rceil)$-bit counter and M_2 is a $\lceil log_2 n \rceil$-bit counter. The partitioning process repeats for M_2 and to all other modules dealing with least significant bits until a module of length one is obtained. A partitioning scheme for a 64-bit counter is shown in the figure 1. Notice that we swapped the sub-counter sizes in the last partition.

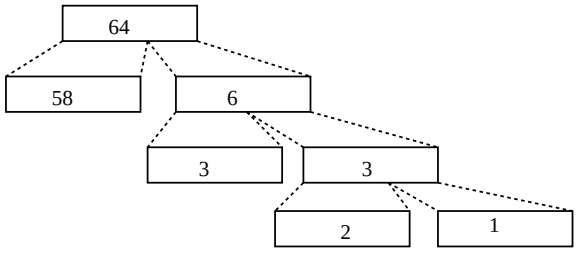

Figure 1. counter partitioning

Using this counter partitioning, M_1 has a delay that is smaller or equal to $n-1$ gate delays (chain of $n-1$ HAs), and the carry-in bit of M_1 comes in intervals greater or equal to n clock cycles. So, if the clock cycle is made the same as a gate delay (plus some other delays: interconnection and FF delay), there is enough time for M_1 to have the incrementer stable before the carry bit arrives from M_2.

Instead of using the actual carry-out bit from M_2, M_1 has an *enable counter* that generates the enable signal to the latch that stores the counter state. The basic structure is presented in figure 2. We use the notation $M^{k,m}$ where k is the modulo of the enable counter and m is the length of the subcounter, in bits. The enable signal to the state register of M_1 is generated in the same cycle when the delay-free carry signal from M_2 happens. As the enable counter needs to be fast, because it uses the high frequency clock of the system, a ring or a twisted-tail counter is utilized. Twisted-tail counters use fewer components than ring counters and are used in the implementation presented in this paper. A modulo-k Twisted-Tail Counter ($k = 2^q$) has a $k/2$-bit state vector $y(t) = (y_{k/2-1}(t), ..., y_1(t), y_0(t))$ and the following transition function: $y_i(t+1) = y_{i-1}(t)$, for $i > 0$ and $y_0(t+1) = (y_{k/2-1}(t))'$, where the apostrophe symbol represents logic complementation. However, twisted-tail counters have the disadvantage, with respect to ring counters, of requiring extra logic to obtain the TC signal (Terminal Count).

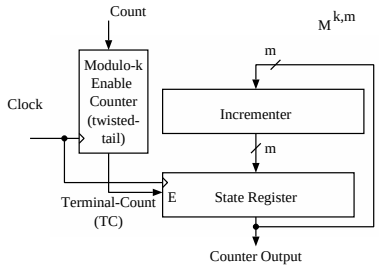

Figure 2. Structure of a $M^{k,m}$ sub-counter

The partitioning method could be further optimized, reducing the number and size of enable counters that was obtained with the original partitioning method. This optimized partitioning method breaks a n-bit counter M into M_1, as a $(n - \lfloor log_2 n \rfloor)$-bit counter, and M_2, as a $\lfloor log_2 n \rfloor$-bit counter, whenever $(n - \lfloor log_2 n \rfloor) \leq 2^{\lfloor log_2 n \rfloor}$. When the condition doesn't apply, we use the first partitioning method instead. An example of this partitioning method is shown in figure 3. Notice that a smaller enable counter is used in $M^{4,4}$ when compared to $M^{8,3}$, obtained in the the previous case.

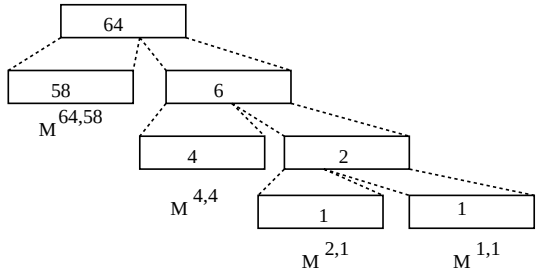

Figure 3. optimized counter partitioning

In this counter design methodology, each sub-counter is decoupled from the others. The enable counters are synchronized and they change state at the same time. More details are presented

[1] a similar design of synchronous up/down counter with frequency independent of the counter size, based also in [1], is described in an unpublished paper [7]. We were made aware of [7] by one of the reviewers of this paper

in [1].

3 Up/Down counter design

In this section we present some modifications to allow down counting capability to the counter presented in the previous section. During normal operation, the next state is obtained incrementing or decrementing the present state of the counter. So, an incrementer/decrementer circuit is needed.

The basic problem is the computation of the next state when the counting direction changes. The time available to have the next counter state computed in any one of the sub-counters can be as low as one clock cycle, and it violates the assumptions in the method presented for the up-counter. The same problem happens when the counter was cleared ($s(0) = 0$), the counting direction starts as *count down* and the *cnt* input is active. To obtain the next state, it is necessary to have propagation of borrows over the length of the sub-counters, which may take more than one clock cycle.

The solution for the problem is to make the counter memorize the last state transition and recover the state when necessary, independently of the delay involved in the incrementer/decrementer circuit. Consider the counting sequence shown in figure 4, assuming that the count input is always active. We show the internal next state being computed ($s(t+1)$) and the present state ($s(t)$) of each sub-counter. When the direction changes, the next state computed for up-counting cannot be used. It must be obtained from the information on the previous state of the counter.

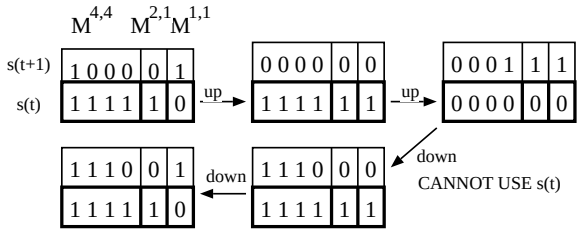

Figure 4. Example of counting sequence with change in counting direction

Each sub-counter may need to recover the state at different instants in time, depending on the state at the time the direction changed. Considering the structure of the counter composed of sub-counters $M^{4,4}$, $M^{2,1}$ and $M^{1,1}$, and the present state is (010110), when the direction changes, each sub-counter is going to recover the previous state after 3, 1 and 1 clock cycles respectively. In order to have this feature, the twisted tail counter that generates the enable signal for the registers should be able to count up and down, what will force the inclusion of some extra circuits in the twisted tail counter to get the next state and a slightly more complex detection of the TC condition. We present in the next subsections a possible solution for these problems.

The block diagram of the proposed design is presented in figure 5. Both enable counter and incrementer module are modified to provide down counting capability. An extra register was included to store the carries/borrows generated in the previous state transition.

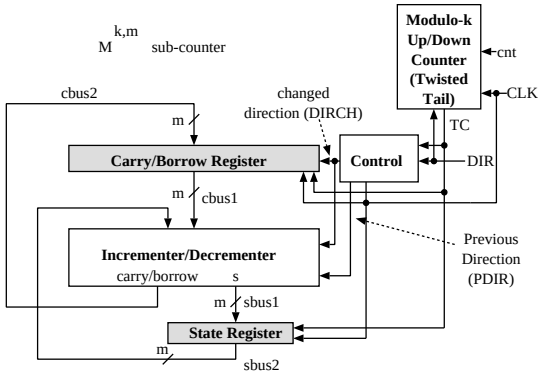

Figure 5. Scheme of the Up/Down sub-counter

3.1 Up/down Twisted Tail Counter

A modulo-k twisted-tail up/down counter with $k = 6$ is shown in figure 6. Using multiplexers, the connections between the flip-

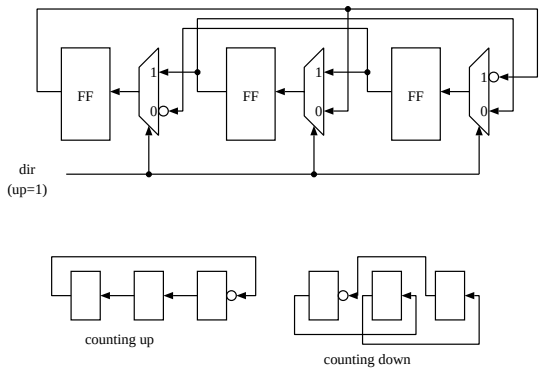

Figure 6. Up/down twisted-tail counter

flips (FF) are modified depending on the direction of counting.

The TC signal, though, must be taken 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 detection of zero state is done testing the extremes of the state vector. This circuit takes 1 CLB in the XC4000 FPGA, and increases the delay to generate TC, when compared to the up-counter (FHMAP delay against a FMAP delay in the original counter). The scheme is presented in figure 7.

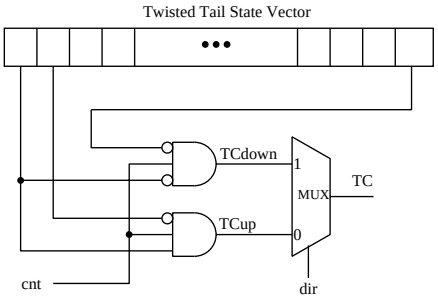

Figure 7. Up/Down Twisted-Tail TC circuit

3.2 Incrementer/Decrementer Circuit and Carry/Borrow Register

The incrementer/decrementer circuit is considered in this work as a chain of Half Adder/Subtractors (HAS). An HAS has a control signal ($OPER$) that commands the circuit to perform addition ($a + c_{in} = 2c_{out} + s$) or subtraction ($a - c_{in} = -2c_{out} + s$). The truth table of these operations is shown below. The carry-out of one module is connected to the carry-in of the next module in the chain.

a	c_{in}	$a + c_{in}$		$a - c_{in}$	
		c_{out}	s	c_{out}	s
0	0	0	0	0	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	0	0	0

As the s output is the same for addition and subtraction, the function that generates s depends on variables a and c_{in}, but not the operation to be performed. The c_{out} output depends on 3 variables (a, c_{in} and $OPER$).

As explained earlier, there's no time to wait for the carries or borrows to propagate when the direction of counting changes. To solve the problem we can store the carry/borrow bits and use them to recover the previous state. The use of carry/borrow bits restricts the use of the Fast Carry Logic available in the XC4000 device. A solution proposed in [7] allows the use of this dedicated circuit.

The storage of the carry/borrow bits will imply an increase in the number of FFs of about the length of the counter. If a carry or borrow came to a certain bit position in the last state change, the bit was inverted and must be inverted again if we want to restore the previous state. Otherwise the bit didn't change and must be kept the same. Once the carry/borrow bits are stored in a register, the previous state is obtained using a XOR function of this register contents and the present state register. The carry/borrow register must be initialized with 1 values, to allow the condition when the counter starts with the down count direction, and the initial state is zero.

The restoration of the previous state must be controlled by each sub-counter independently. When the counter is initialized and the direction is $down$ or when the direction changes during regular operation, the control circuit transmits the information of direction change and forces the incrementer/decrementer to use the carry/borrow register contents to obtain the next state. The condition is kept until a TC signal is generated by the enable counter, or the direction is restored to the previous situation. It is important to note that only the generation of the next state works based on the new direction, the carry generation continues to work in the previous counting direction ($PDIR$). That is important because the direction can change more than once during the counting period of a sub-counter (that can take many clock cycles) and the sub-counter should be able to resume the computation of the new next state that cannot be restored from the register. So, the HAS should work with the carry/borrow chain independently of the output generation (next state), such that both up and down next states can be available when the enable signal is generated.

To make the incrementer/decrementer generate the output based on the carry/borrow register, we use a control signal named $DIRCH$ (indicates that a new counting direction was given to the counter). This signal makes the HAS module use the carry-in from the previous module in the chain ($DIRCH = 0$), or the value stored in the carry/borrow register ($DIRCH = 1$). A possible mapping of the HAS module, with the needed modifications, to the XC4000 function generators is presented in figure 8, with the logical equations for each output. The input *cbreg* is a bit that comes from the carry/borrow register.

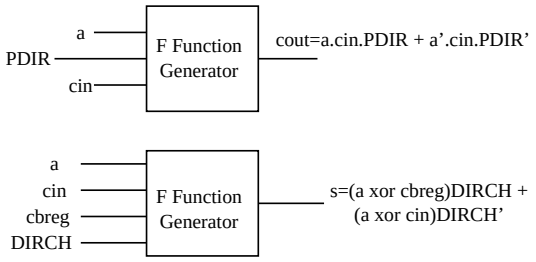

Figure 8. Mapping of the HAS module with state recovering to FPGA Function Generators

The state register is clocked when TC is generated by the enable counter. The carry/borrow register changes state based on TC and the $DIRCH = 0$. When $DIRCH = 1$, the counting direction changed, the value in the carry/borrow register must be kept until the next TC. This feature is necessary because we don't get valid carry/borrow information from the incrementer/decrementer when the counting direction changes. If the direction signal is stable for more than one TC pulse, the sub-counter resumes regular operation.

3.3 Control Circuit

The control circuit is a sequential system that has the behavior represented by the state diagram in figure 9. The state changes every time TC is activated by the enable counter. The initial state S_0 is necessary for the case of counting down after clearing the counter. The output signals of the controller are: $DIRCH$ and $PDIR$. The $DIRCH$ indicates that the counting direction changed. The $PDIR$ signal indicates the previous counting direction used. The state of the circuit changes when the Enable Counter generates the TC signal.

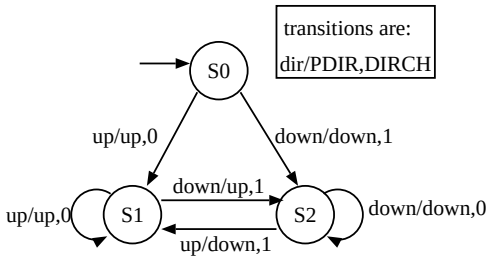

Figure 9. Control Circuit State Diagram

Extra delays caused by the control logic will increase the clock cycle time of this design, when compared to the up-counter. This is discussed in the next section.

4 Experimental Results

The design was specified in Viewlogic VHDL and the synthesis results are presented in this section. The results were obtained without imposition of constraints to the synthesis tools. No manual placement or routing were performed, what leaves some space for optimizations and better performance. The design was also tested in the EVC board [6] to verify the counter operation.

We split the results in two subsections. The first one shows the implementation of the up-counter using the methodology, and the second shows the implementation of the up/down counter.

4.1 Up-counter implementation

The minimum clock period for the up-counter should be 1 FMAP delay plus interconnection delay and propagation time of a FF (set up time is included in the FMAP delay). This value can be made as short as 10ns. Unfortunately, the implementation results shows that the broadcast of the enable signal, from the Enable Counter to the State Register, makes the propagation delay of this signal greater than 10ns. It is caused by the large fan-out of the signal and it becomes worse as we enlarge the counter size. On the other hand, this problem can be solved independently of the counter size, in order to reduce its effect to a minimum. We discuss this problem by the end of this section.

length	P1	P2	P3	P4	# FFs	#CLBs (FCL)	#CLBs (w/o FCL)
32	27	3	1	1	51	26	27
36	31	3	1	1	55	28	28
37	32	3	1	1	56	28	30
38	32	4	1	1	73	37	39
40	34	4	1	1	75	38	39
50	44	4	1	1	85	43	45
60	54	4	1	1	95	48	50
64	58	4	1	1	99	50	51
70	64	4	1	1	105	53	54
71	64	4	2	1	140	70	70
128	121	4	2	1	197	99	99

Table 1. Estimation of the Counter Design Area

Table 1 shows the partitioning results obtained from the equations presented in section 2 and area estimates in terms of number of CLBs for some counter sizes. We are assuming the optimized partitioning method. The number of CLBs used is presented for two different implementations, one considering Fast Carry Logic (FCL) and the other disregarding Fast Carry Logic (w/o FCL). To implement the design we used two versions of a 64-bit counter, both described in Viewlogic VHDL. The first one uses standard structures and describes the incrementer as a chain of HAs. The second uses XBLOX add/sub module as the incrementer. The first implementation does not take advantage of the Fast Carry Logic and the second does. Because of that, the first design uses a slower incrementer. The area is almost the same for both cases.

A good implementation of the first case would give the following parameter estimates for the incrementer:

$$delay = t_p(\lceil \frac{n-4}{3} \rceil + 1) \text{ (ns)}$$

$$area = 2(\lceil \frac{n-4}{3} \rceil + 1) \text{ (CLBs)}$$

assuming a CLB delay of t_p ns (FMAP delay, interconnect delay and FF propagation time). The structure is shown in the figure 10, for an incrementer of 10 bits, using 4-input LUTs that are available in the XC4000 series. From the CLBs used in the incrementer, only $\lceil \frac{n-4}{3} \rceil$ FFs can be used by the twisted tail counter. Other FFs that are needed for the twisted tail counter will increase the area used by the final sub-counter (2 FFs per CLB). This was considered in the table. The synthesis tool used 62 CLBs.

In the second implementation, using Fast Carry Logic, the incrementer delay can be estimated using the equation: $8.5 + 0.75n$ (ns) (based on application notes [4]) for a XC4000-5 device. The area used by the incrementer is only $n/2$ CLBs. The total number of CLBs used in this implementation equals to half the number of FFs needed in the design (50 CLBs in a 64-bit counter).

Our implementation consumes more area than Vuillemin's counter. This increase in area corresponds to the Enable Counter used.

The use of 4-input LUT FPGA technology allows the implementation of incrementers of up to 4 bits with only one CLB delay. The counter partitioning used is quite appropriate to 4-input LUTs

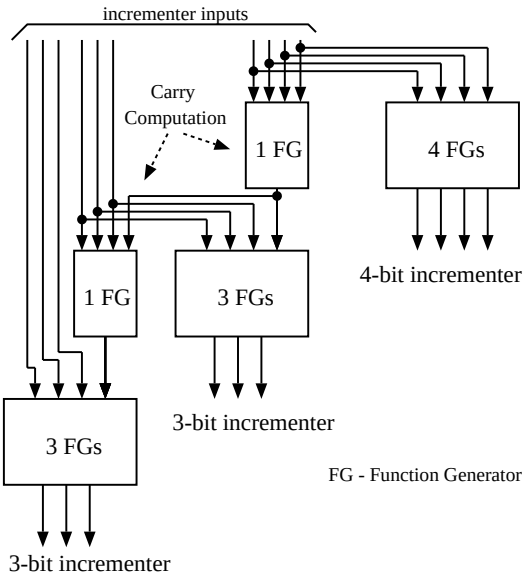

Figure 10. Example of the incrementer circuit partitioning (10 bits), without Fast Carry Logic

since 3 out 4 partitions used in the cases presented in the table have at most 4 bits (for reasonable counter size).

The most important observation is that the last and most significant group of bits (leftmost sub-counter) can be made larger than 2^6 bits!! An incrementer of 58 bits, using fast carry logic will have a estimated delay of 52ns (without considering interconnection delays) what is far below the enable signal period of the $M^{64,58}$ sub-counter that uses it, that is $640ns$ (counter clock cycle of 10ns). Even the regular implementation using the chain of HAs would have a delay of 190ns. Based on this observation, we know that the incrementer delay is not going to be in the critical path, and it is possible to use much larger most-significant sub-counters than was initially calculated (using the partition method). For an enable signal period of $640ns$ we could have roughly 800 bits in the leftmost sub-counter (assuming 4 sub-counters in the design). So, for all practical purposes, only 4 partitions are needed and adjustments of the counter length are done in the leftmost sub-counter, for large number of bits. These adjustments involve the width of the incrementer and state register.

It was observed that the propagation delay of the enable signal in the sub-counter is the critical path delay in the design. As the number of bits increase, the fan-out of the enable signal increases. The enable signal is generated by the combination of the state of the enable counter and the count signal. It must be broadcast to many FFs in the leftmost sub-counter, in just one clock cycle. Our experiments show that the enable signal path delay for the 64-bit counter is 24.6ns (for the $M^{64,58}$ sub-counter).

Long lines (with smaller delay and larger fan-out capacity) can be used to minimize the delay in the path. Another approach is to split the path into a tree. The enable signal is broken into two lines, for example, using two equivalent circuits that generate TC, each circuit will feed half of the original load. When combining the idea of signal split and Long Lines to broadcast the signals, we obtained a critical path of 16.2 ns (60MHz).

Another solution is to use Global Buffers (GB). The use of GBs makes the circuit that transmits the signal less sensitive to an increase in the load. Since a small number of GBs are available, a careful placement is important to reduce interconnect delay between the signal source and the buffer.

The conclusion of this discussion is that the delay caused by the broadcast of the enable signal (TC) can be reduced independently of the sub-counter size.

4.2 Up/down counter implementation

The up/down sub-counter $M^{k,m}$ consumes an area of $m + k/4+3$ CLBs (except for $M^{1,1}$ that uses always 0.5 CLB). The first term is the number of CLBs used by the incrementer/decrementer and registers, the second term is the area used by the twisted tail counter, and the last term is the area used by control logic.

For the 64-bit up/down counter, the area used is calculated as 91 CLBs. It represents an increase of 78% in area, when compared to the up-counter. This increase is caused by the inclusion of the extra register to restore the state.

The critical path in the up/down counter is related to the distribution of the control signals. In the path associated to the generation of DIRCH and the correct output of the next state from the incrementer/decrementer, there are 2 CLBs. The delay is 2*FMAP plus interconnect and FF delays.

If the load is excessively affecting the delay in the circuit, it's possible to reduce the load based on the same ideas proposed for the up-counter design.

5 Summary

The paper presents the implementation of a fast counter of arbitrary precision with constant counting period for FPGA technology. We improve the functionality of the counter making it an up/down counter. The experimental results were obtained using simulation of a 64-bit counter and estimates of the area and delay for other cases. The clock cycle time obtained for a 64-bit up-counter was 16.2ns (60MHz) but could be reduced even more, since a reasonable value is around 10ns and the cycle time is independent of the counter size. The paper gives some solutions to the problem. The up/down counter for a 64-bit implementation would consume 78% more area and have a clock cycle time of 2*FMAP delays. We estimate a frequency of almost 50MHz. The design was functionally tested in the EVC FPGA board, with a XC4010-5 with a clock frequency of 25MHz.

Acknowledgments. This research has been supported in part by the NSF Grant MIP-9314172 "Arithmetic Algorithms and Structures for Low-Power Systems" and by CNPq.

References

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

[2] Ercegovac, M. D., Lang T. and Moreno, J.; *Introduction to Digital Systems*, in preparation, John Wiley & Sons, New York, 1996.

[3] Vuillemin, J. E.; Constant Time Arbitrary Length Synchronous Binary Counters; *IEEE* 10^{th} *Symposium on Computer Arithmetic*, 1991, pp. 180-183.

[4] Xilinx; *The Programmable Logic Data Book*, August 1993.

[5] Xilinx; *The XC4000 Data Book*; August 1992.

[6] VCC– EVC1 – Engineer's Virtual Computer, User's Manual.

[7] Stan, M. R. and Burleson, W. P.; Synchronous Up/Down Counter with Period Independent of Counter Size; distributed at FPGA'96.