SYSTOLIC UP/DOWN COUNTERS WITH ZERO AND SIGN DETECTION

Behrooz Parhami

Computer Science Department
University of Waterloo

ABSTRACT

Although a state encoding scheme for systolic counters has been presented earlier, several important practical problems such as zero test, sign detection, overflow, underflow, and modulo-n (cyclic) counting have not been dealt with adequately. In this paper, design principles for unary and binary systolic up/down counters are presented. The unary counters, which are attractive when dealing with relatively small counts, are based on the systolic stack concept. The binary counters use conventional binary number representation, with several tags associated with each bit position. The binary counter design presented can be generalized to counters with higher-radix state encodings.

1. INTRODUCTION

Efficient exploitation of VLSI technology in high performance systems requires designs with cellular structures and short inter-cell communication paths. Kung's systolic arrays [KUNG79], [KUNG82] capture these requirements. As a result, the design of systolic systems and systolic algorithms has been a very active research area in computer science for the past few years.

Building up on the work of Leiserson [LEIS79], Guibas and Liang [GUIB82] have described a state encoding scheme for a systolic up/down counter. Their design is incomplete in the sense that they do not deal with several important practical problems such as zero test, sign detection, overflow, underflow, and modulo-n counting adequately. Guibas and Liang use the digit symbol set {b,0,1,c} with a binary positional number system. This is in effect a special signed-digit (SD) number system [AVIZ61] with the digit set $\{\bar{1},0,1,2\}$ and binary positional weighting.

In this paper, the above encoding scheme is shown to be insufficient for bidirectional counting with positive and negative counts. Alternative unary and binary representations are then proposed. The proposed unary counters, which consist of very simple cells, are useful when the maximum count value is relatively small. In such cases, the increase in the number of counter positions may be compensated for by the simpler cell logic. The proposed binary counters are based on the conventional binary representation. Both types of counters can be designed to be capable of zero, sign, overflow, and underflow detection and to count modulo n.

Guibas and Liang [GUIB82] use the following encoding scheme for the counter states. Initially, zero is represented by $00\cdots0$. Each counter position can be in one of four states $\{\bar{1},0,1,2\}$, with its next state determined by its present state and the current value of its right-hand (less significant) digit. The rightmost, or least significant, counter digit is assumed to be followed by:

$\bar{1}$	when counting down
0 or 1	when in steady state
2	when counting up

The transition table showing the next state as a function of the present state and the state of a cell's right neighbor is as follows:

Present State \ Right Neighbor →	$\bar{1}$	0	1	2
$\bar{1}$	0	1	1	2
0	$\bar{1}$	0	0	1
1	0	1	1	2
2	$\bar{1}$	0	0	1

The following examples of up and down counting have been constructed using the above table. Leading zeros have been deleted for brevity of representation.

Up	Down
0	0
1	$1\bar{1}$
2	$1\bar{1}0$
11	$10\bar{1}$
12	100
21	$1\bar{1}1\bar{1}$
102	$2\bar{1}0$
111	$12\bar{1}$
112	112

Using the standard binary weighting factors for different positions, the correctness of counts for the above examples can be established and easily extended to the general case.

This design is incomplete in the sense that the following problems have not been dealt with in the framework of an actual design (only an informal argument is presented for zero detection):

a. Zero detection
b. Overflow detection
c. Modulo-n counting
d. Counter initilization

For non-negative counts, zero and underflow detection are the same problem. However, to maintain negative as well as positive counts, the following additional problems must be dealt with:

e. Representation of negative values
f. Underflow detection
g. Sign detection

Even though it may seem that because of the digit set $\{\bar{1}, 0, 1, 2\}$ there should be no problem in representing negative numbers, the fact that zero has many different representations if a leading $\bar{1}$ digit is allowed ($0 = \bar{1}2 = \bar{1}\bar{1}2 = \bar{1}1\bar{1}2 = \bar{1}2\bar{1}2 = \cdots$) makes the zero detection problem nontrivial.

2. UNARY UP/DOWN COUNTERS

The challenge in the design of a systolic up/down counter is that the counter cells must behave identically for up and down counting (or even when not counting at all, for that matter), so that no global control signal is required. Thus, representing the value of n by a stack containing n 1's in a linear array of binary cells (see below) fails on the account that the behavior of cells at the boundary between 1's and 0's depends on whether the counter is counting up or down.

$\cdots$ 0 0 0 0 0 1 1 1 1 1 1 1 1 1 ← Top of Stack

For counting up, the rightmost 0 must change to a 1, while for counting down, the leftmost 1 must change to a zero.

The stack mechanism proposed by Guibas and Liang [GUIB82] can be used for alleviating this problem. Briefly, the stack is organized with a number of empty cells, as shown in Figure 1. After each counting cycle, two or three *idle* cycles (depending on the stack design) are required for the stack to rearrange its elements in such a way that the next increment or decrement operation can be performed in a propagation-free manner. If simpler cells requiring three idle cycles are used, the following string rewriting rules describe the rearranging process, assuming that the string of actual values in the stack is padded with a dummy 1 on the right and with a dummy value which is always equal to the value in the last cell, on the left:

$$011 \rightarrow 101$$
$$100 \rightarrow 010$$

With two idle cycles, the rearrangement becomes more complex.

Guibas and Liang [GUIB82] have shown that such a stack functions properly and is space-efficient in the sense that an empty cell is needed only at the top of the stack when it is full. Actually, when the stack is used as a unary counter, no empty cell is needed at all when we reach the maximum count. The example in Table I shows the counter principles.

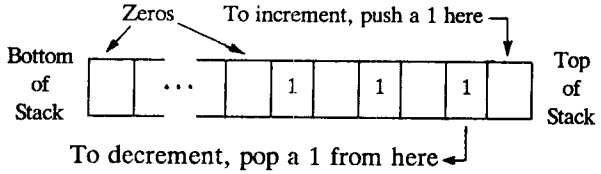

Figure 1. Systolic Unary Counter.

Table I

Unary Systolic Up/Down Counting Example.

Initial value (0):	0	0	0	0	0	0	0
Count up (to 1):	0	0	0	0	0	0	1
After 3 idle cycles:	0	0	0	0	0	1	0
Count up (to 2):	0	0	0	0	0	1	1
After 3 idle cycles:	0	0	0	1	0	1	0
Count up (to 3):	0	0	0	1	0	1	1
After 3 idle cycles:	0	0	1	1	0	1	0
Count up (to 4):	0	0	1	1	0	1	1
After 3 idle cycles:	1	0	1	1	0	1	0
Count up (to 5):	1	0	1	1	0	1	1
After 3 idle cycles:	1	1	1	1	0	1	0
Count up (to 6):	1	1	1	1	0	1	1
After 3 idle cycles:	1	1	1	1	1	1	0
Count up (to 7):	1	1	1	1	1	1	1
After 3 idle cycles:	1	1	1	1	1	1	1
Count down (to 6):	1	1	1	1	1	0	1
After 3 idle cycles:	1	1	1	1	1	1	0
Count down (to 5):	1	1	1	1	1	0	0
After 3 idle cycles:	1	1	1	1	0	1	0
Count down (to 4):	1	1	1	1	0	0	0
After 3 idle cycles:	1	1	0	1	0	1	0
Count down (to 3):	1	1	0	1	0	0	0
After 3 idle cycles:	1	0	0	1	0	1	0
Count down (to 2):	1	0	0	1	0	0	0
After 3 idle cycles:	0	0	0	1	0	1	0
Count down (to 1):	0	0	0	1	0	0	0
After 3 idle cycles:	0	0	0	0	0	1	0
Count down (to 0):	0	0	0	0	0	0	0
After 3 idle cycles:	0	0	0	0	0	0	0

This design is capable of performing all conventional counter operations. Up and down counting were illustrated above. To test for zero, simply test the second cell for zero after the required idle cycles. To test for maximum count, test the first cell for 1 after the required idle cycles. Overflow and underflow conditions can also be indicated quite simply.

Modulo-n (cyclic) counting can be accomodated by using four-state cells in a counter which can count up to $n-1$. Denoting the state set by {0,R,1,S}, we simply augment the up-count rule to change a "1" into "R", augment the down-count rule to change "0" into "S", and add the following rewriting rules for rearrangements during idle cycles:

$$\begin{array}{lcl} 1R & \rightarrow & R0 \\ RR & \rightarrow & 00 \\ S0 & \rightarrow & 11 \\ 0S & \rightarrow & S1 \\ SS & \rightarrow & 11 \end{array}$$

Remember that the dummy cell to the left is assumed to have the same state as the leftmost stack cell. For instance, counting up from 7 in the above example, we have:

Initial value (7):	1	1	1	1	1	1	1
Count up (to 0):	1	1	1	1	1	1	R
After 3 idle cycles:	1	1	1	R	0	0	0
Count up (to 1):	1	1	1	R	0	0	1
After 3 idle cycles:	R	0	0	0	0	1	0
Count up (to 2):	R	0	0	0	0	1	1
After 3 idle cycles:	0	0	0	1	0	1	0

Note that zero detection is still possible by examining the value in the second cell, even though the representation of zero is no longer unique. The symbol "R" can be thought of as *resetting* the string of 1's to its left. Similarly, "S" *sets* the string of 0's to its left. These two symbols accomplish a gradual "inversion" towards the left end of the stack which does not interfere with proper counter operation up front.

As another example, consider what might happen if we cross the seven-zero boundary several times in succession:

Initial value (7):	1	1	1	1	1	1	1
Count up (to 0):	1	1	1	1	1	1	R
After 3 idle cycles:	1	1	1	R	0	0	0
Count down (to 7):	1	1	1	R	0	S	0
After 3 idle cycles:	R	0	S	1	1	1	1
Count up (to 0):	R	0	S	1	1	1	R
After 3 idle cycles:	1	1	1	R	0	0	0
etc.							

Again, both zero and maximum count can be tested for by the previous simple procedures. In larger counters, several R's and S's may be present at the same time. This causes no problem, as they will move to the left with equal speed, resetting and setting the digits in the process.

Initialization to all 0s or all 1s is done simply by injecting an "R" or an "S" into the rightmost cell. To initialize to an arbitrary preset value, a state I must be added to each cell. The rewriting rules

$$\begin{array}{lcl} 0I & \rightarrow & Iq \\ 1I & \rightarrow & Iq \\ II & \rightarrow & qq \end{array}$$

ensure proper initialization, where q is the wired-in constant (0 or 1) associated with the initialization of the cell currently in state "I" so that the final result represents the desired initial count.

A final variation will allow the counter to count in negative as well as positive values. The negative value $-i$ can be represented by i entries of $\bar{1}$ in a stack with three-state cells. Counting up or down is accomplished by inserting a 1 or $\bar{1}$, respectively, in the stack. The following rewriting rules for the idle cycles guarantee correct operation:

$$\begin{array}{lcl} 1\bar{1}1 & \rightarrow & 001 \\ \bar{1}11 & \rightarrow & 001 \\ 011 & \rightarrow & 101 \\ 0\bar{1}\bar{1} & \rightarrow & \bar{1}0\bar{1} \\ 100 & \rightarrow & 010 \\ \bar{1}00 & \rightarrow & 0\bar{1}0 \end{array}$$

In this case, deletion from the stack is simply not needed. Zero and sign detection can be accomplished by examining the second cell after the required idle cycles. Intuitively, the stack contains either 1s or $\bar{1}$s, but not both, except possibly for a short period after the insertion. The modulo-n counting modification in this case is similar and thus will not be discussed.

Alternatively, one can use the original cells by appending a sepcial "sign" cell to the rightmost counter cell. The counter then operates according to Table II.

Table II

Updating the Sign Cell of a Counter.

Sign	Zero?	Count	Operation	New Sign
X	Yes	Up	Increment	Plus
X	Yes	Down	Increment	Minus
Plus	X	Up	Increment	Plus
Plus	No	Down	Decrement	Plus
Minus	No	Up	Decrement	Minus
Minus	X	Down	Increment	Minus

3. BINARY UP/DOWN COUNTERS

Before presenting the design of a systolic binary counter, it is instructive to review several attempts at the design which appeared in earlier versions of this paper. These earlier designs were based on the binary signed-digit number system [AVIZ61] utilizing the digit set $\{\bar{1}, 0, 1\}$ in order to take advantage of its carry-free

addition and subtraction property. Since the sign of a signed-digit number is determined by the sign of the left-most nonzero digit, a representation scheme was needed to make this digit readily available for inspection. The initial attempt consisted of "folding" the vector of signed digits so that both the least and most significant digits were available to the update and test control circuitry (Figure 2). The next design was based on manipulating the count from the most significant end (Figure 3) and required the introduction of special cell states to hold pending increments and decrements which gradually moved to the right, eventually affecting the two least significant digits, and to temporarily accomodate possible expansion and contraction in the number of digits. Even though the cells were less complex than those for the initial design, each cell still required 13 times more states than would be needed for straight storage of a single binary signed digit. In addition, the formal proof that the counter operated correctly was rather involved.

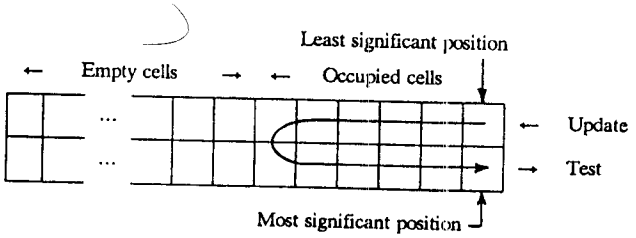

Figure 2. Folded Systolic Binary Counter.

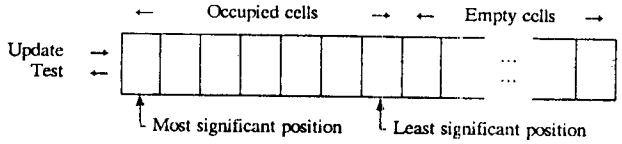

Figure 3. Counting from the Most Significant End.

The design to be presented here is based on conventional binary representation with relatively simple cells. The key to this simplification is the realization that the count changes sign only when its value reaches zero, so that zero detection is all that is needed for proper maintenance of a separate sign. Figure 4 shows the basic arrangement of the counter cells storing the sample count of +18. We assume the existence of a dummy cell to the left of the counter which always stores the same digit value as the most significant counter position and a dummy control cell to the right of the counter which always stores the complement of the least significant count digit. These cells are physically nonexistent and their effects are maintained by proper internal generation of the left and right neighbor state signals respectively.

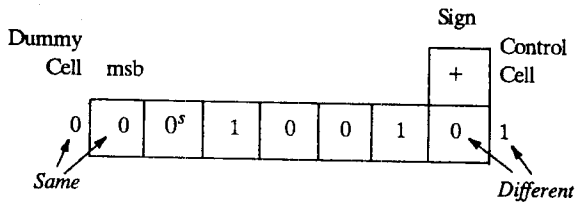

Figure 4. Systolic Binary Counter Design.

The state of each counter cell will be denoted by d_τ^σ where $d \in \{0, 1\}$ is the count digit, the tag $\tau \in \{blank, b, c, r\}$ is used to denote "borrow," "carry," or a special "reset" state, and finally the superscript $\sigma \in \{blank, s\}$ which appears in exactly one counter cell denotes the "sameness" of all the digits to the left of the marked position. The count of zero is indicated by the state 0^s in the least significant position. The all 1s count, which may lead to overflow or underflow in the next operation, is detected by the state 1^s is the least significant position. The sign updating rules are the same as those presented for unary counters in Table II.

Updating of the count corresponds to the following rewriting rules which are applied in two phases:

Phase 1:	$0\,1_\tau^s$	$\rightarrow\ 0^s\,1_\tau$	$(\tau \neq r)$
Superscript	$1\,0_\tau^s$	$\rightarrow\ 1^s\,0_\tau$	$(\tau \neq r)$
Updating	$0^s\,0$	$\rightarrow\ 0\,0^s$	
	$1^s\,1$	$\rightarrow\ 1\,1^s$	
	$0\,d_b$	$\rightarrow\ 1_b\,d$	
Phase 2:	$1\,d_b$	$\rightarrow\ 0\,d$	
Subscript	$0\,d_c$	$\rightarrow\ 1\,d$	
Updating	$1\,d_c$	$\rightarrow\ 0_c\,d$	
	$d'd_r$	$\rightarrow\ d'_r\,0$	

To increment the counter, the control cell at the right end of the counter assumes the state x_c, where x is the complement of the least significant count digit as usual. To reset the counter to zero, the control cell assumes the state x_r^s. The control cell design must be such that the $\tau \neq r$ exception of the first two rewriting rules does not apply to it. This causes the superscript s to move to the least significant counter position and the proper final state of $00 \cdots 000^s$ to eventually prevail.

Modulo-2^a counting is achieved automatically in a counter with n cells. If modulo-n counting for an arbitrary value n is desired, each cell can be provided with a wired-in constant q which denotes the value to be assumed by the cell if the counter is to hold the count of $n-1$. Now, an additional superscript state m enables the counter to count modulo n. Only one cell has the superscript state m which denotes that from the marked position to the most significant position, the count digits match the digits of $n-1$. The maximum count of $n-1$ is reached iff the least significant position contains d^m. In this situation, the next increment will reset the

counter. The following rewriting rules describe the manipulation of m :

$$d^m q \quad \rightarrow \quad d\ q^m$$
$$d\ \bar{q}^m \quad \rightarrow \quad q'^m \bar{q}$$

In other words, when a matching q digit appears, the marker moves right and when a previously matching digit changes to a non-matching digit, the marker moves left. When the count of zero is to be decremented, the control cell assumes the state $\bar{q}^m$, where q is the preset matching bit for the control cell. This forces the m to move to the least significant digit, indicating a count of $n-1$. Even though the rest of the digits are not updated properly, they eventually assume correct values if and when the m moves left.

4. CONCLUSION

We have presented designs for unary and binary systolic counters with zero, sign, overflow, and underflow detection and with the ability to count modulo-n for an arbitrary value of n .

Generalization of the designs to higher radix counters is straightforward. Such counters will be more efficient in the sense of requiring less overhead in the form of tag storage.

ACKNOWLEDGEMENT

The author gratefully acknowledges Dr. Farhad Mavaddat of the University of Waterloo for bringing this problem to his attention. The work reported here was supported in part by the National Science and Engineering Research Council of Canada under grants G1140 and A5515.

REFERENCES

[AVIZ61] Avizienis, A., "Signed-Digit Number Representation for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers,* Vol. EC-10, pp. 389-400, 1961.

[GUIB82] Guibas, L.J. and F.M. Liang, "Systolic Stacks, Queues, and Counters," *Proc. of the 1982 Conf. on Advanced Research in VLSI,* MIT, pp. 155-164.

[KUNG79] Kung, H.T., "Let's Design Algorithms for VLSI Systems," *Proc. of the Caltech Conf. on VLSI,*" Jan. 1979.

[KUNG82] Kung, H.T., "Why Systolic Architectures?," *Computer,* Vol. 15, No. 1, pp. 37-46, Jan. 1982.

[LEIS79] Leiserson, C.E., "Systolic Priority Queues," *Proc. of the Caltech Conf. on VLSI,* Jan. 1979.