Design alternatives for barrel shifters

Matthew R. Pillmeier
Rushmore Processor 2
Unisys Corporation
Blue Bell, PA 19424

Michael J. Schulte and E. George Walters III
Computer Architecture and Arithmetic Laboratory
Computer Science and Engineering Department
Lehigh University
Bethlehem, PA 18015, USA

ABSTRACT

Barrel shifters are often utilized by embedded digital signal processors and general-purpose processors to manipulate data. This paper examines design alternatives for barrel shifters that perform the following functions: shift right logical, shift right arithmetic, rotate right, shift left logical, shift left arithmetic, and rotate left. Four different barrel shifter designs are presented and compared in terms of area and delay for a variety of operand sizes. This paper also examines techniques for detecting results that overflow and results of zero in parallel with the shift or rotate operation. Several Java programs are developed to generate structural VHDL models for each of the barrel shifters. Synthesis results show that data-reversal barrel shifters have less area and mask-based data-reversal barrel shifters have less delay than other designs. Mask-based data-reversal barrel shifters are especially attractive when overflow and zero detection is also required, since the detection is performed in parallel with the shift or rotate operation.

Keywords: barrel shifters, rotators, masks, data-reversal, overflow detection, zero flag, computer arithmetic.

1. INTRODUCTION

Shifting and rotating data is required in several applications including arithmetic operations, variable-length coding, and bit-indexing. Consequently, barrel shifters, which are capable of shifting or rotating data in a single cycle, are commonly found in both digital signal processors and general-purpose processors. Several patents [1–10] and research articles [11–15] have been written on efficient designs and implementations for barrel shifters. In [16], several 32-bit barrel shifters are compared in terms of delay, power, and power-delay product.

This paper examines design alternatives for barrel shifters that perform the following operations: shift right logical, shift right arithmetic, rotate right, shift left logical, shift left arithmetic, and rotate left. These designs are optimized to share hardware for different operations. Techniques are also presented for detecting results that overflow and results of zero in parallel with the shift or rotate operation. Section 2 describes the basic shift and rotate operations and gives examples of each operation. Section 3 presents designs for several types of barrel shifters. Section 4 gives area and delay estimates for each type of barrel shifter as the operand size varies. Section 5 presents conclusions. Further details on the designs presented in this paper are given in [17].

2. SHIFT AND ROTATE OPERATIONS

In this paper, we define A to be the input operand, B to be the shift/rotate amount, and Y to be the shifted/rotated result. We define A to be an n-bit value, where n is an integer power of two. Therefore, B is a $\log_2(n)$-bit integer representing values from 0 to $n-1$. The barrel shifters presented in this paper perform the following six operations: shift right logical, shift right arithmetic, rotate right, shift left logical, shift left arithmetic, and rotate left. Table 1 gives an example for each of these operations. In this table, the bit vector for A is denoted as $a_7a_6a_5a_4a_3a_2a_1a_0$ and the shift/rotate amount, B, is 3 bits. As illustrated in this table:

Operation	Y
3-bit shift right logical	0 0 0 $a_7a_6a_5a_4a_3$
3-bit shift right arithmetic	$a_7a_7a_7a_7a_6a_5a_4a_3$
3-bit rotate right	$a_2a_1a_0a_7a_6a_5a_4a_3$
3-bit shift left logical	$a_4a_3a_2a_1a_0$ 0 0 0
3-bit shift left arithmetic	$a_7a_3a_2a_1a_0$ 0 0 0
3-bit rotate left	$a_4a_3a_2a_1a_0a_7a_6a_5$

Table 1. Shift and rotate examples for $A = a_7a_6a_5a_4a_3a_2a_1a_0$ and $B = 3$.

3-bit Opcode			
left	*rotate*	*arithmetic*	Operation
0	0	0	shift right logical
0	0	1	shift right arithmetic
0	1	X	rotate right
1	0	0	shift left logical
1	0	1	shift left arithmetic
1	1	X	rotate left

Table 2. Operation control bits.

- A B-bit shift right logical operation performs a B-bit right shift and sets the upper B bits of the result to zeros.
- A B-bit shift right arithmetic operation performs a B-bit right shift and sets the upper B bits of the result to a_{n-1}, which corresponds to the sign bit of A.
- A B-bit rotate right operation performs a B-bit right shift and sets the upper B bits of the result to the lower B bits of A.
- A B-bit shift left logical operation performs a B-bit left shift and sets the lower B bits of the result to zeros.
- A B-bit shift left arithmetic operation performs a B-bit left shift and sets the lower B bits of the result to zeros. The sign bit of the result is set to a_{n-1}.
- A B-bit rotate left operation performs a B-bit left shift and sets the lower B bits of the result to the upper B bits of A.

3. BARREL SHIFTER DESIGNS

This section discusses barrel shifter designs. Basic shifter and rotator designs are described first in Section 3.1. Mux-based data-reversal barrel shifters, mask-based data-reversal barrel shifters, mask-based two's complement barrel shifters, and mask-based one's complement barrel shifters are then discussed in Sections 3.2 through 3.5. In the following discussion the term multiplexor refers to a 1-bit 2-to-1 multiplexor, unless otherwise stated. The operation performed by the barrel shifters is controlled by a 3-bit opcode, which consists of the bits *left*, *rotate*, and *arithmetic*, as summarized in Table 2. Additional control signals, *sra* and *sla*, are set to one when performing shift right arithmetic and shift left arithmetic operations, respectively.

3.1. Shifters and Rotators

An n-bit logarithmic barrel shifter uses $\log_2(n)$ stages [1, 2]. Each bit of the shift amount, B, controls a different stage of the shifter. The data into the stage controlled by b_k is shifted by 2^k bits if $b_k = 1$; otherwise it is not

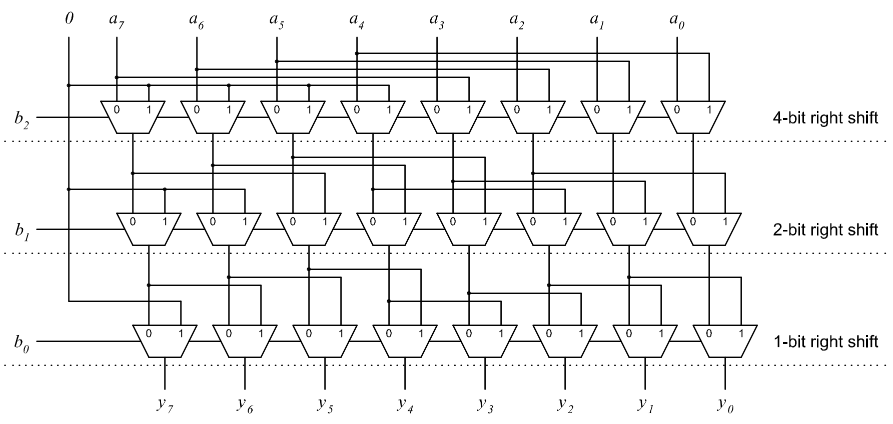

Figure 1. 8-bit logical right shifter.

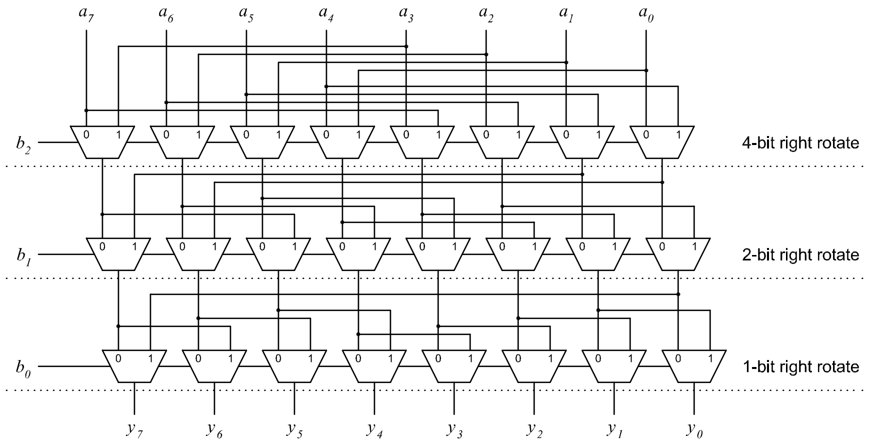

Figure 2. 8-bit right rotator.

shifted. Figure 1 shows the block diagram of an 8-bit logical right shifter, which uses three stages with 4-bit, 2-bit, and 1-bit shifts. To optimize the design, each multiplexor that has '0' for one of its inputs can be replaced by a 2-input AND gate with the data bit and $\overline{b_k}$ as inputs.

A similar unit that performs right rotations, instead of right shifts, can be designed by modifying the connections to the more significant multiplexors. Figure 2 shows the block diagram of an 8-bit right rotator, which uses three stages with 4-bit, 2-bit, and 1-bit rotates. The right rotator and the logical right shifter supply different inputs to the more significant multiplexors. With the rotator, since all of the input bits are routed to the output, there is no longer a need for interconnect lines carrying zeros. Instead, interconnect lines are inserted to enable routing of the 2^k low order data bits to the 2^k high order multiplexors in the stage controlled by b_k. Changing from a non-optimized shifter to a rotator has no impact on the theoretical area or delay. The longer interconnect lines of the rotator, however, can increase both area and delay.

The logical right shifter can be extended to also perform shift right arithmetic and rotate right operations by adding additional multiplexors. This approach is illustrated in Figure 3, for an 8-bit right shifter/rotator with three stages of 4-bit, 2-bit, and 1-bit shifts/rotates. Initially, a single multiplexor selects between '0' for logical right shifting and a_{n-1} for arithmetic right shifting to produce s. In the stage controlled by b_k, 2^k multiplexors select between s for shifting and the 2^k lower bits of the data for rotating.

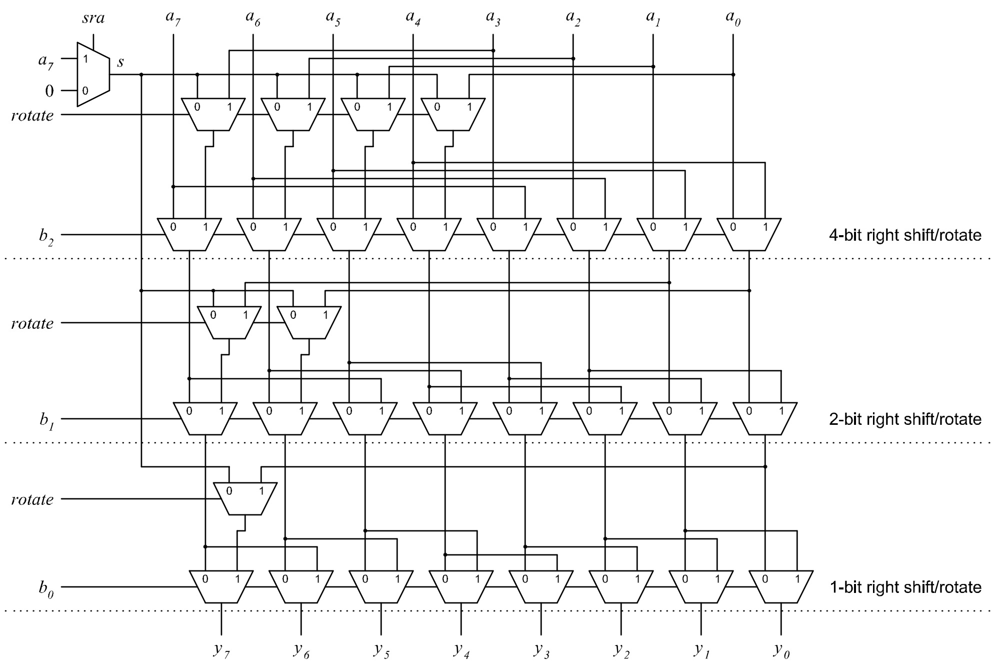

Figure 3. 8-bit mux-based right shifter/rotator.

A right shifter can be extended to also perform left shift operations by adding a row of n multiplexors both before and after the right shifter [4]. When a left shift operation is performed, these multiplexors reverse the data into and out of the right shifter. When a right shift operation is performed, the data into and out of the shifter is not changed. An 8-bit data-reversal logical shifter is shown in Figure 4.

3.2. Mux-based Data-Reversal Barrel Shifters

The techniques described previously can be combined to form a barrel shifter that performs shift right logical, shift right arithmetic, rotate right, shift left logical, shift left arithmetic, and rotate left. This approach is shown in Figure 5. Initially, a row of n multiplexors reverses the order of the data when $left = 1$ to produce $\hat{A}$. Then, an n-bit right shifter/rotator performs the right shift or rotate operation on $\hat{A}$ to produce $\hat{Y}$. Finally, a row of n multiplexors reverses the data when $left = 1$ to produce the final result Y.

The design presented in Figure 5, called a mux-based data-reversal barrel shifter, also detects overflow and results of zero. Overflow only occurs when performing a shift left arithmetic operation and one or more of the shifted-out bits differ from the sign bit. A method for detecting overflow in parallel with the shift operation is shown in Figure 6. In each stage, the bits that are shifted out are XORed with the sign bit; when no bits are shifted out, the sign-bit is XORed with itself*. The outputs of the XOR gates are then ORed together to produce the overflow flag, which is '1' when overflow occurs. An additional multiplexor sets $\hat{y}_0$ to $\hat{a}_0$ when $sla = 1$. The zero flag, which is '1' when Y is zero, is obtained from the logical NOR of all of the bits in $\hat{Y}$. One disadvantage of this mux-based data-reversal barrel shifter is that the zero flag is not computed until $\hat{Y}$ is produced.

3.3. Mask-based Data-Reversal Barrel Shifters

Another approach for designing barrel shifters with the same functionality as the one presented in Figure 5 is to use a mask-based data-reversal approach [6]. With this approach, the primary unit that performs the operations is a right rotator and the data-reversal technique is used to support left shift and rotate operations. In parallel with the data reversal and rotation, masks are computed that allow logical and arithmetic shifting to also be performed. With the mask-based data-reversal approach, the overflow and zero flags are computed

*For shift left arithmetic, the sign-bit corresponds to $\hat{a}_0$, since the data is reversed.

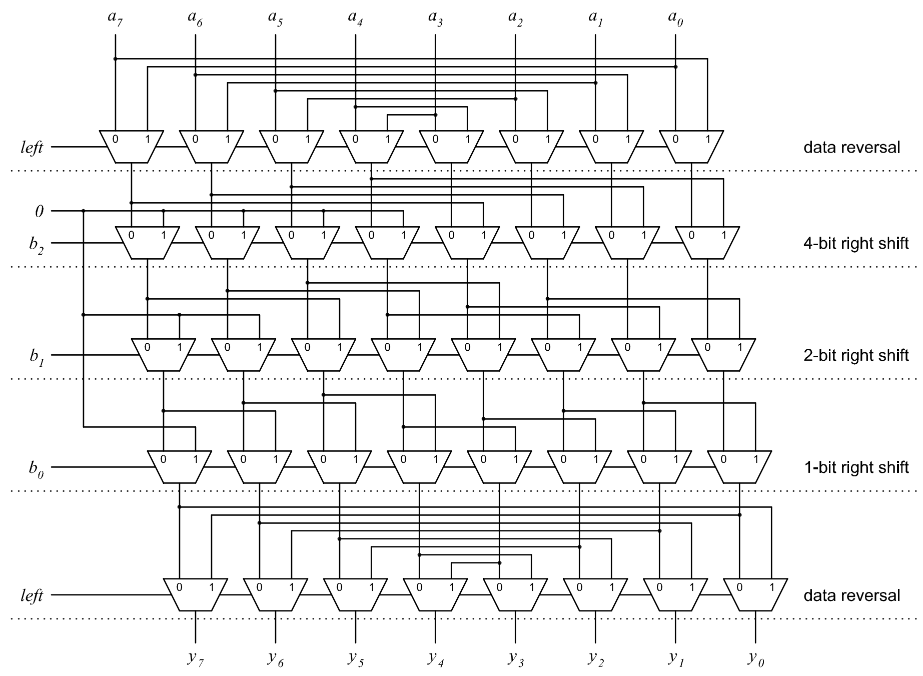

Figure 4. 8-bit data-reversal logical shifter.

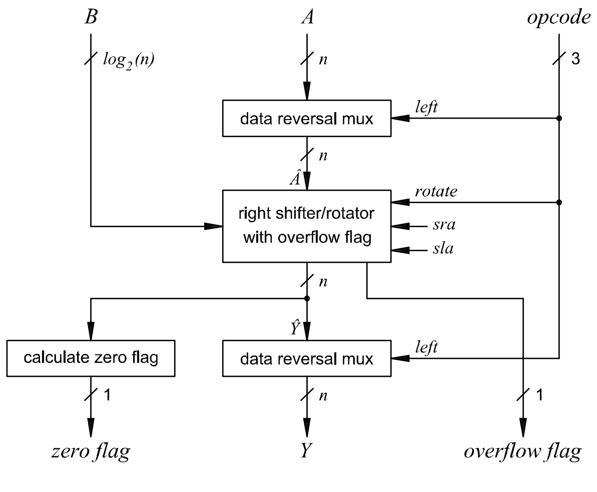

Figure 5. Mux-based data-reversal barrel shifter.

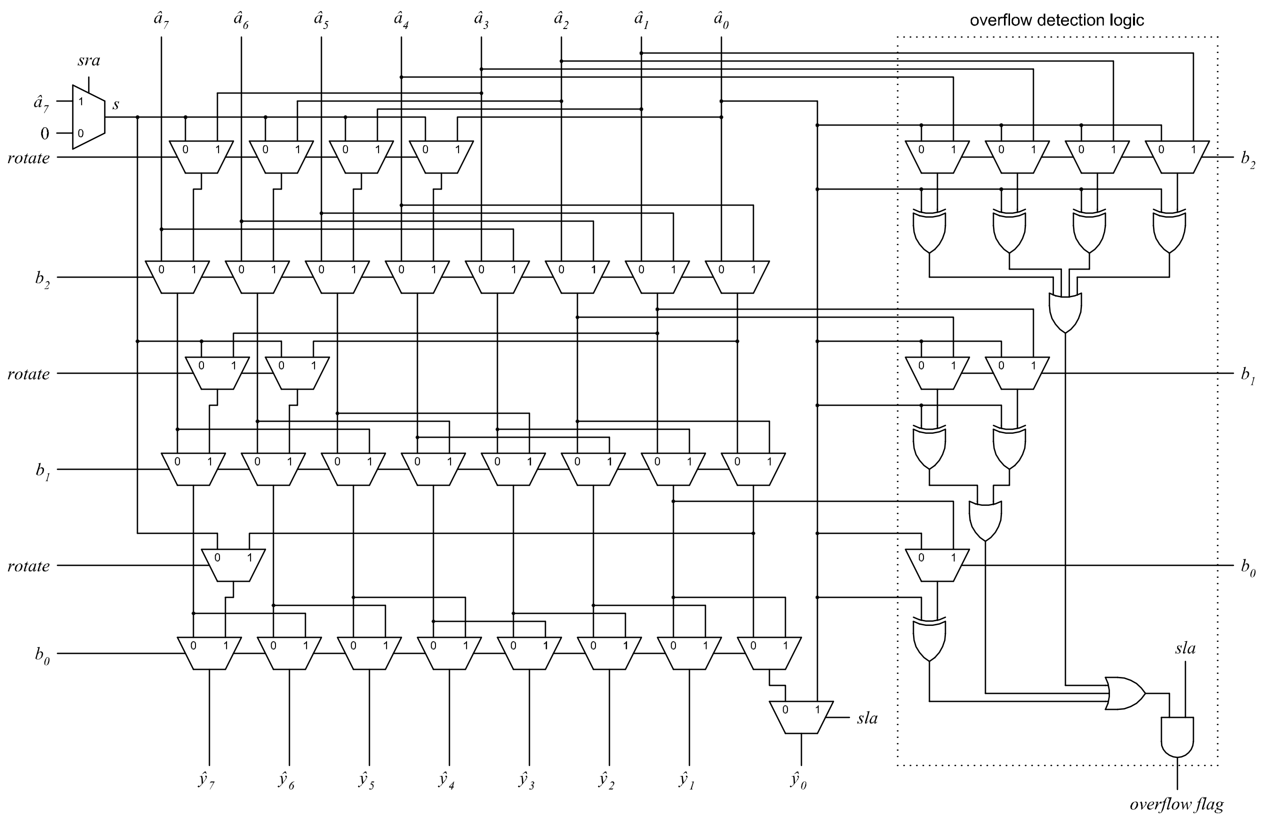

Figure 6. 8-bit right shifter/rotator with overflow flag.

while reversing and rotating the data. A mask-based data-reversal barrel shifter is shown in Figure 7. With this type of barrel shifter, several masks are computed in parallel with the data reversal and rotation. The first mask that is computed is the F mask, which contains B leading zeros and $n - B$ trailing ones. For example, if $B = 3$ and $n = 8$, then $F = 00011111$. The design of an 8-bit F mask generator is shown in Figure 8. When the F mask is ANDed with a rotate right result, a shift right logical result is produced. To facilitate both rotating and shifting, each bit of the F mask is ORed with the *rotate* signal to produce the P mask. Thus,

$$p_i = f_i + rotate \qquad (0 \leq i \leq n-1) \tag{1}$$

which sets P to all ones for rotations. The temporary result, T, before the final data reversal, is computed as:

$$t_0 = \hat{y}_0 \cdot \overline{sla} + sla \cdot a_{n-1} \tag{2}$$

$$t_i = \hat{y}_i \cdot p_i + s \cdot \overline{p_i} \qquad (1 \leq i \leq n-1) \tag{3}$$

where $s = sra \cdot a_{n-1}$.

One advantage of the masked-based data-reversal barrel shifter is that the overflow and zero flags are both computed while the data is being rotated [7]. The zero flag is calculated using an n-bit zero mask, Z, which masks out any bits that do not appear in the final result. Z is computed as

$$z_0 = p_{n-1} + sla \tag{4}$$

$$z_i = \overline{sla} \cdot p_{n-1-i} + sla \cdot p_{n-1} \qquad (1 \leq i \leq n-1) \tag{5}$$

After the Z mask is computed, it is bit-wise ANDed with $\hat{A}$, and the resulting n bits are NORed to compute the zero flag. To calculate the overflow flag, the sign bit, a_{n-1}, is XORed with the $n-1$ remaining bits of A to

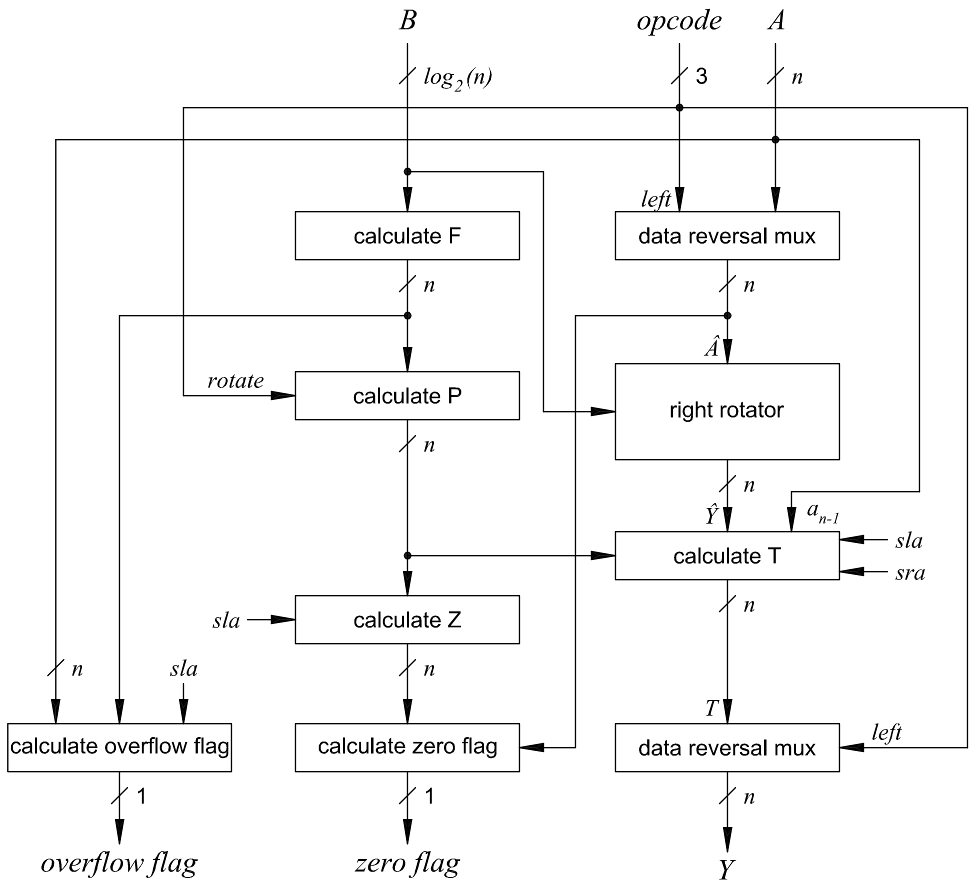

Figure 7. Mask-based data-reversal barrel shifter.

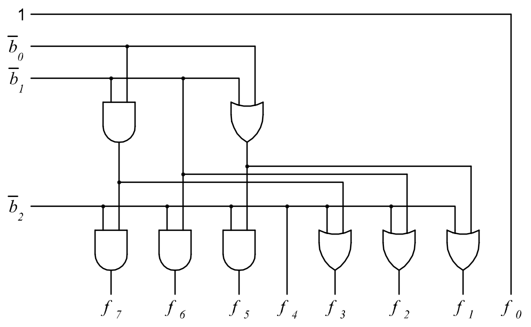

Figure 8. 8-bit F mask generator.

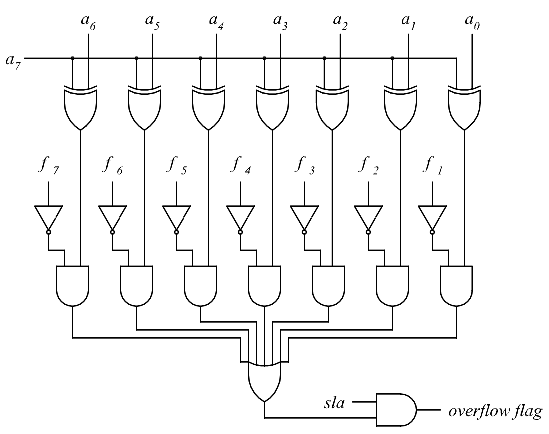

Figure 9. 8-bit overflow calculation.

determine which bits differ from the sign bit [8]. The $(n-1)$-bit result from this operation is then ANDed with the complement of the $(n-1)$ leading bits of the F mask. Finally, these bits are ORed together and the result is ANDed with sla to produce the overflow flag. This approach is shown in Figure 9 for $n = 8$.

3.4. Mask-based Two's Complement Barrel Shifters

Another approach for designing mask-based barrel shifters takes advantage of the fact that a B-bit left rotate is equivalent to an $(n-B)$-bit right rotate [5]. When n is an integer power of two, $n-B$ corresponds to the two's complement of B. Thus, when a left shift or rotate operation is performed, the two's complement of B is used as the shift amount, which eliminates the multiplexors needed to reverse the input and output data.

A masked-based two's complement barrel shifter is shown in Figure 10. The F mask is computed using the technique presented in Section 3.3, except $\hat{B}$ is used instead of B. When right shifts are performed, the F mask is used directly. When left shifts are performed, the complement of the F mask is used. When performing rotate operations or when $\hat{B} = 0$, the P mask is set to all ones. Thus, the P mask is now computed as

$$p_i = left \oplus f_i + rotate + bzero \qquad (0 \le i \le n-2) \tag{6}$$
$$p_{n-1} = left \cdot \overline{f}_{n-1} + \overline{sla} \cdot \overline{left} \cdot f_{n-1} + rotate + bzero \tag{7}$$

where $bzero = 1$ when $\hat{B} = 0$. Y is computed as

$$y_i = \hat{y}_i \cdot p_i + s \cdot \overline{p_i} \qquad (0 \le i \le n-2) \tag{8}$$
$$y_{n-1} = (\hat{y}_{n-1} \cdot p_{n-1} + s \cdot \overline{p_{n-1}}) \cdot \overline{sla} + sla \cdot a_{n-1} \tag{9}$$

where $s = sra \cdot a_{n-1}$. Z is computed as

$$z_i = sla \cdot p_{n-i-2} + \overline{sla} p_{n-i-1} \qquad (0 \le i \le n-1) \tag{10}$$
$$\tag{11}$$

After the Z mask is computed, it is bit-wise ANDed with A, and the resulting n bits are NORed to produce the zero flag. The overflow flag is still computed using the technique shown in Figure 9, with F replaced by P.

3.5. Mask-based One's Complement Barrel Shifters

One disadvantage of mask-based two's complement barrel shifters is the need to perform a carry-propagate addition to obtain the two's complement of B when performing left shifts and rotates. This carry propagation can be eliminated by instead using the one's complement of B for left shifts and rotates, and making minor modifications to the rotator and mask generation logic [3]. This approach is shown in Figure 11. The rotator is

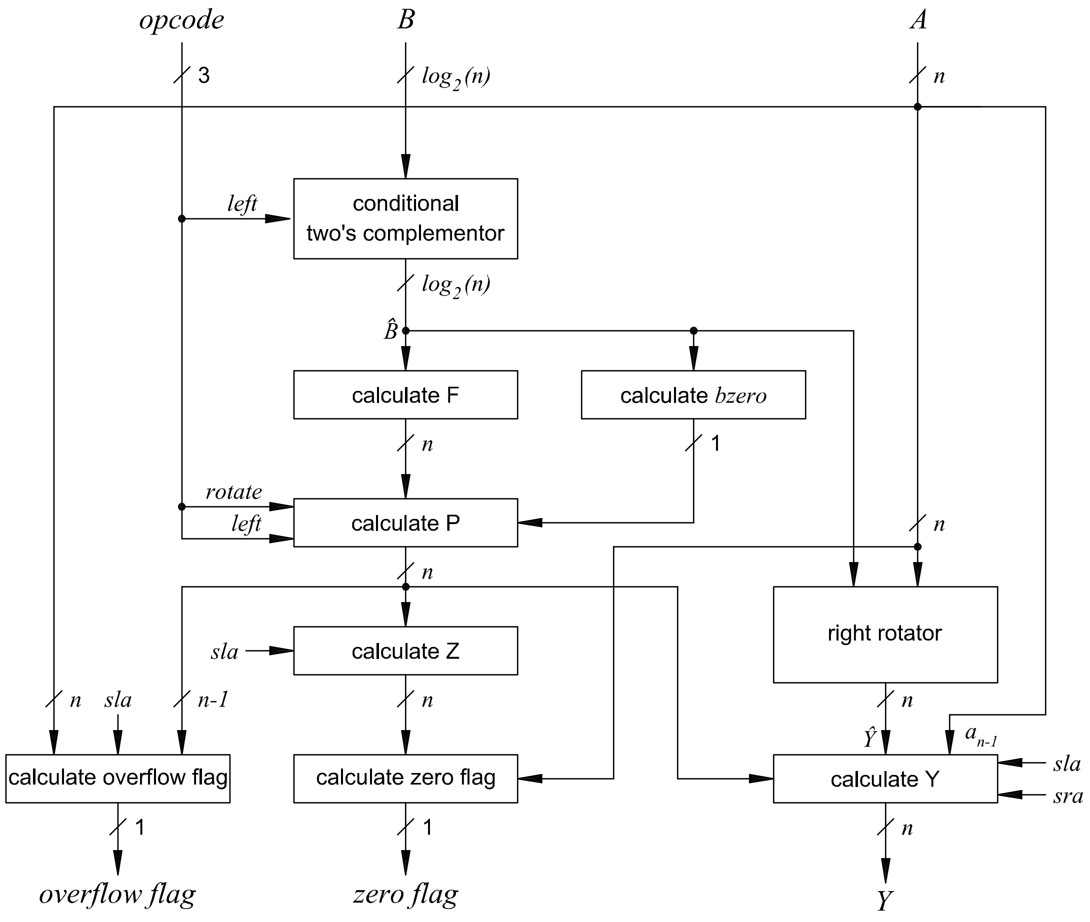

Figure 10. Mask-based two's complement barrel shifter.

modified by adding a row of multiplexors that performs a 1-bit rotate and a row of XOR gates that complements B for left shifts and rotates. Since it is no longer necessary to test if $\hat{B} = 0$, P is computed as

$$p_i = \mathit{left} \oplus f_i + \mathit{rotate} \qquad (0 \le i \le n-1) \tag{12}$$

$$p_{n-1} = \mathit{left} \cdot \overline{f}_{n-1} + \overline{\mathit{sla}} \cdot \overline{\mathit{left}} \cdot f_{n-1} + \mathit{rotate} \tag{13}$$

Z is computed as

$$z_i = \mathit{sla} \cdot p_{n-i-2} + \overline{\mathit{sla}} p_{n-i-1} \qquad (0 \le i \le n-2) \tag{14}$$

$$z_{n-1} = \mathit{sla} + \overline{\mathit{sla}} p_0 \tag{15}$$

The rest of the unit is identical to the mask-based two's complement barrel shifter.

4. AREA AND DELAY ESTIMATES

To obtain area and delay estimates for the designs presented in this paper, several Java programs were written to generate structural VHDL for each barrel shifter, given the input operand size. These programs were used to generate 8, 16, 32, 64, and 128-bit barrel shifters, which were first simulated to verify functionality and then synthesized using IBM's CU-11 0.11 micron CMOS standard cell library, with copper interconnect [18]. Each design was synthesized to optimize for delay.

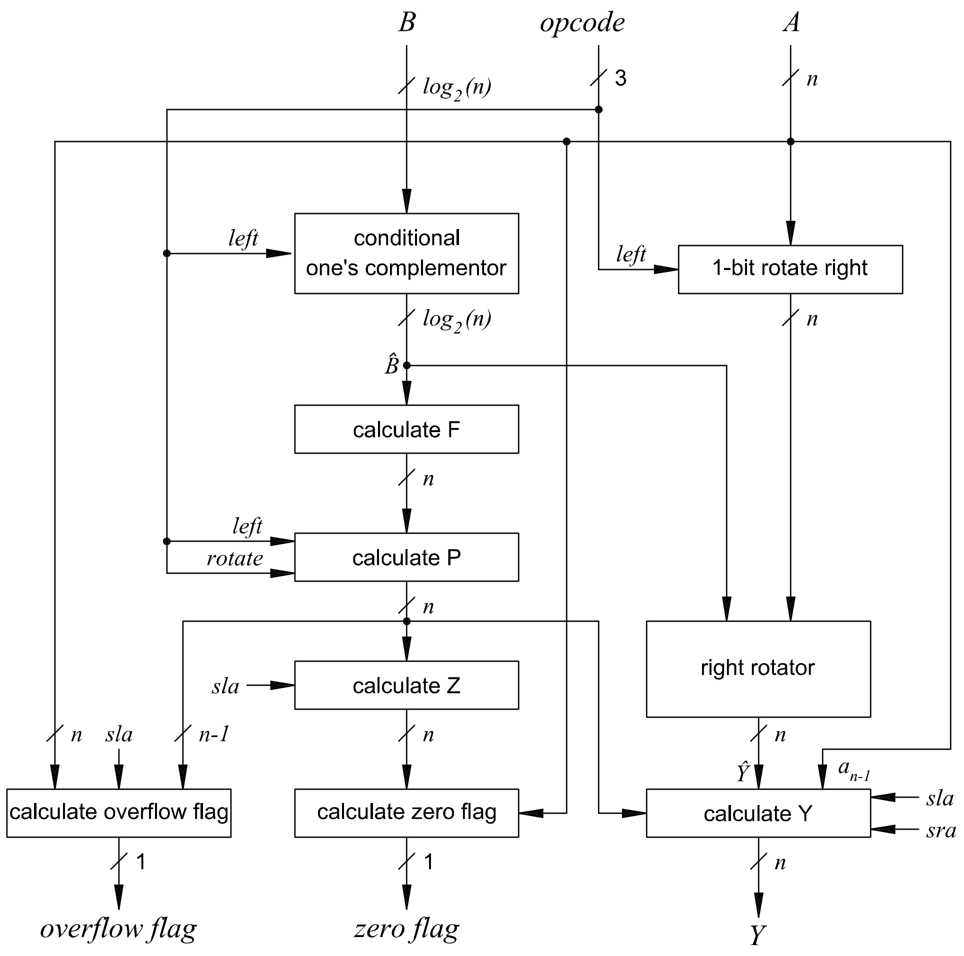

Figure 11. Mask-based one's complement barrel shifter.

	Operand Size				
Barrel Shifter	8	16	32	64	128
Mux-based Data-Reversal	1308	2731	6416	14242	30990
Mask-based Data-Reversal	1226	3180	6141	14488	31424
Mask-based Two's Complement	1958	3908	8827	17657	36592
Mask-based One's Complement	1926	4507	9825	20247	40453

Table 3. Total area estimates in equivalent gates.

Tables 3 and 4 show area estimates in equivalent gates and delay estimates in nanoseconds for each barrel shifter. As illustrated in Table 3, either the mux-based data-reversal or mask-based data-reversal designs have the lowest area, depending on the operand size. Table 4 shows that mask-based data-reversal barrel shifters require less delay than the other barrel shifters for all operand sizes examined. They are faster than the mux-based data-reversal barrel shifters, since they compute the zero and overflow flags in parallel with the rotate operation and they do not have additional multiplexor delays for selecting between shift and rotate operations.

	Operand Size				
Barrel Shifter	8	16	32	64	128
Mux-based Data-Reversal	0.68	0.83	1.08	1.30	1.46
Mask-based Data-Reversal	0.61	0.75	0.94	1.06	1.27
Mask-based Two's Complement	0.89	1.03	1.19	1.41	1.67
Mask-based One's Complement	0.86	1.08	1.22	1.44	1.74

Table 4. Worst case delay estimates in nanoseconds.

5. CONCLUSIONS

This paper has examined four barrel shifter designs: mux-based data-reversal, mask-based data-reversal, mask-based two's complement, and masked-based one's complement. Area and delay estimates, based on synthesis of structural level VHDL, indicate that data-reversal barrel shifters have less area than two's complement or one's complement barrel shifters and that mask-based data-reversal barrel shifters have less delay than the other designs. As the operand size increases, the delay of the shifters increases as $O(\log(n))$ and their area increases as $O(n \log(n))$.

Acknowledgment

This material is based upon work supported by the IBM Faculty Partnership Program and the National Science Foundation under Grant No. 9703421. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of IBM or the National Science Foundation.

REFERENCES

1. M. Seckora, "Barrel Shifter or Multiply/Divide IC Structure," *U.S. Patent 5,465,222*, November 1995.
2. J. Muwafi, G. Fettweis, and H. Neff, "Circuit for Rotating, Left Shifting, or Right Shifting Bits," *U.S. Patent 5.978,822*, December 1995.
3. A. Yamaguchi, "Bidirectional Shifter," *U.S. Patent 5,262,971*, November 1993.
4. A. Ito, "Barrel Shifter," *U.S. Patent 4,829,460*, May 1989.
5. T. Thomson and H. Tam, "Barrel Shifter," *U.S. Patent 5,652,718*, July 1997.
6. F. Worrell, "Microprocessor Shifter using Rotation and Masking Operations," *U.S. Patent 5,729,482*, March 1998.
7. G. F. Burns, "Method for Generating Barrel Shifter Result Flags Directly from Input Data," *U.S. Patent 6,009,451*, December 1999.
8. H. S. Lau and L. T. Ly, "Left Shift Overflow Detection," *U.S. Patent 5,777,906*, July 1998.
9. M. Diamondstein and H. Srinivas, "Fast Conversion Two's Complement Encoded Shift Value for a Barrel Shifter," *U.S. Patent 5,948,050*, September 1999.
10. K. Dang and D. Anderson, "High-Speed Barrel Shifter," *U.S. Patent 5,416,731*, May 1995.
11. R. Pereira, J. A. Mitchell, and J. M. Solana, "Fully Pipelined TSPC Barrel Shifter for High-speed Applications," *IEEE Journal of Solid State Circuits*, vol. 30, pp. 686–690, June 1995.
12. S.-J. Yih, M. Cheng, and W.-S. Feng, "Multilevel Barrel Shifter for CORDIC Design," *Electronics Letters*, vol. 32, pp. 1178–1179, June 1996.
13. P. A. Beerel, S. Kim, P.-C. Yeh, and K. Kim, "Statistically Optimized Asynchronous Barrel Shifters for Variable Length Codecs," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 261–263, 1999.
14. V. Milutinovic, M. Bettinger, and W. Helbig, "Multiplier/Shifter Design Tradeoffs in a 32-bit Microprocessor," *IEEE Transactions on Computers*, vol. 38, pp. 874–880, June 1989.
15. G. M. Tharakan and S. M. Kang, "A New Design of a Fast Barrel Switch Network," *IEEE Journal of Solid-State Circuits*, vol. 28, pp. 217–221, February 1992.

16. K. P. Acken, M. J. Irwin, and R. M. Owens, "Power Comparisons for Barrel Shifters," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 209–212, 1996.
17. M. R. Pillmeier, "Barrel Shifter Design, Optimization, and Analysis," Master's thesis, Lehigh University, January 2002.
18. IBM, *Blue Logic Cu-11 ASIC*, June 2002. Available from http://www-3.ibm.com/chips/products/asics /products/cu-11.html.