Journal of Computer Science 4 (4): 305-308, 2008
ISSN 1549-3636

Performance Analysis of a 32-Bit Multiplier with a Carry-Look-Ahead Adder and a 32-bit Multiplier with a Ripple Adder using VHDL

Hasan Krad and Aws Yousif Al-Taie
Department of Computer Science and Engineering,
College of Engineering, Qatar University, P.O. Box 2713, Doha, Qatar

Abstract: This study presents a performance analysis of two different multipliers for unsigned data, one uses a carry-look-ahead adder and the second one uses a ripple adder. The study's main focus is on the speed of the multiplication operation on these 32-bit multipliers which are modeled using VHDL, A hardware description language. The multiplier with a carry-look-ahead adder has shown a better performance over the multiplier with a ripple adder in terms of gate delays. Under the worst case, the multiplier with the fast adder shows approximately twice the speed of the multiplier with the ripple adder. The multiplier with a ripple adder uses time = 979.056 ns, while the multiplier with the carry-look-ahead adder uses time = 659.292 ns.

Key words: Multiplier, carry-look-ahead adder, ripple adder, VHDL simulation

INTRODUCTION

Multiplication can be considered one of the basic arithmetic operations. However, it is not as simple as addition or subtraction operations, because it takes more time to perform two subtasks, addition and shifting. Typically, a multiplication operation takes between 2 and 8 cycles[2]. Therefore, using high-speed multipliers is a critical requirement for processors with a high performance. The multiplier uses the addition operation for all the partial products. The adder can be a ripple adder, a carry-look-ahead adder, or any other adder[5,8]. However, using a fast adder for the multiplier improves the over all performance of the multiplication operation. Our study is focused on multipliers using unsigned data. VHDL, a Very High Speed Integrated Circuit Hardware Description Language, was used to model our multiplier design.

Several researchers had worked on the performance analysis of adders and other researchers on the performance analysis of multipliers. Sertbas and Selami worked on the performance analysis of classified binary adder architectures. They compared the ripple adder, carry-look-ahead adder, carry select adder and the conditional sum adder. They used VHDL in their comparison. Their study included the unit-gate models for area and delay[1]. Asadi and Navi developed a new 54×54 bit multiplier using a high-speed carry-look-ahead adder. Their proposed multiplier reduced the number of transistors, delay and power consumption[2]. Aziz and Kamruzzaman developed a synthesizable VHDL model for a generalized signed multiplier capable of performing multiplication of both signed-magnitude and two's compliment operands[3].

Ripple carry adder: Ripple carry adders use multiple full adders with the carry ins and carry outs chained together, where the correct value of the carry bit ripples from one bit to the next[4].

The two Boolean functions for the sum and carry are:

$$SUM = A_i \oplus B_i \oplus C_i$$
$$C_{out} = C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i$$

We modeled this module with the following VHDL code:

```
ENTITY R_Adder IS
  PORT (
    a,b: IN BIT_VECTOR (31 DOWNTO 0);
    cin: IN BIT;
    sum: OUT BIT_VECTOR (31 DOWNTO 0);
    cout: OUT BIT
  );
END Entity;

ARCHITECTURE R_Adder_Beh OF R_Adder IS
constant gate_delay2: time: = 10 ns;
constant gate_delay4: time: = 20 ns;
```

Corresponding Author: Hasan Krad, Department of Computer Science and Engineering, College of Engineering, Qatar University, P.O. Box 2713, Doha, Qatar Tel: +974 485-2677 Fax: +974 485-2777

```
SIGNAL car: BIT_VECTOR (32 DOWNTO 0);
BEGIN
  car(0) <= cin;
  PROCESS (a, b, car)
  BEGIN
    FOR i IN 0 to 31 LOOP
      sum(i) <= a(i) XOR b(i) XOR car(i) after
      gate_delay2;
      car(i+1) <= (a(i) AND b(i)) OR (car(i) AND (a(i)
      XOR b(i))) after gate_delay4;
    END LOOP;
   END PROCESS;
   cout <= car(32);
END ARCHITECTURE;
```

Carry look ahead adder: To reduce the delay caused by the effect of carry propagation through the ripple carry adder, we can attempt to evaluate quickly for each stage whether the carry-in from previous stage will have a value of 0 or 1[4].

Given the two Boolean functions for the sum and carry as follows:

$$SUM = A_i \oplus B_i \oplus C_i$$
$$C_{out} = C_{i+1} = A_i \cdot B_i + (A_i \oplus B_i) \cdot C_i$$

If we let:

$G_i = A_i \cdot B_i$ The generate function

$P_i = (A_i \oplus B_i)$ The propagate function

Then

$C_{i+1} = G_i + P_i \cdot C_i$ The Carry Function

Thus, for 4-bit adder, we can extend the carry, as shown below:

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

In general, we can write:

$SUM_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$. The sum function

$$C_{i+1} = \sum_{j=0}^{i} (G_j \prod_{k=j+1}^{i} P_k) + \prod_{k=0}^{i} P_k Cin$$ The carry function

We modeled this module with the following VHDL code:

```
ENTITY CLA_Adder IS
  PORT (
    a,b:IN BIT_VECTOR (31 DOWNTO 0);
    cin: IN BIT;
    sum: OUT BIT_VECTOR (31 DOWNTO 0);
    cout: OUT BIT
  );
END Entity;

ARCHITECTURE CLA_Adder_Beh OF CLA_Adder
IS
constant gate_delay1: time: = 5 ns;
constant gate_delay2: time: = 10 ns;
SIGNAL gen: BIT_VECTOR (31 DOWNTO 0);
SIGNAL pro: BIT_VECTOR (31 DOWNTO 0);
SIGNAL car: BIT_VECTOR (32 DOWNTO 0);
BEGIN
  gen <= a AND b after gate_delay1;
  pro <= a XOR b after gate_delay1;
  car(0) <= cin;
  PROCESS (gen, pro,car)
  BEGIN
    FOR i IN 1 to 32 LOOP
      car(i) <= gen(i-1) OR (pro(i-1) AND car(i-1))
      after gate_delay2;
    END LOOP;
  END PROCESS;
  sum <= pro XOR car(31 DOWNTO 0) after
  gate_delay1;
  cout <= car(32);
END ARCHITECTURE;
```

Unsigned multiplier: Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed up to produce the final product. The multiplication of two n-bit binary integers results in 2n-bit product. We can perform a fast multiplication by the number 2, by simply shifting the number one-bit position to the left. This is called a fast multiplication or bit shifting[5].

VHDL simulation: The VHDL simulation of the two multiplier are presented in this section. The VHDL code for both an unsigned multiplier using a fast carry-look-ahead adder and an unsigned multiplier using a ripple adder are generated. The VHDL model has been developed using the DirectVHDL simulator. The multipliers use 32-bit values. The worst case was applied using the two multipliers, where the gate delay is assumed to be 5 ns. The algorithms for the two multipliers are shown below:

Algorithm for a multiplier with a carry-look-ahead adder:

Begin Program
Multiplier = 32 bits

Multiplicand = 32 bits
Register = 64 bits
Put the multiplier in the least significant half and clear the most significant half
  For i = 1 to 32
  Begin Loop
    If the least significant bit of the 64-bit register contains binary '1'
    Begin If
      Add the Multiplicand to the Most Significant Half using the CLAA
      Begin Adder
        C[0] = '0'
        For j = 0 to 31
        Begin Loop
          Calculate Propagate P[j] = Multiplicand[j] ⊕ Most Significant Half[j]
          Calculate Generate G[j] = Multiplicand[j]·Most Significant Half[j]
          Calculate Carries C[i + 1] = G[i] + P[i] · C[i]
          Calculate Sum S[i] = P[i] ⊕ C[i]
        End Loop
      End Adder
      Shift the 64-bit Register one bit to the right throwing away the least significant bit
    Else
      Only Shift the 64-bit Register one bit to the right throwing away the least significant bit
    End If
  End Loop
  Register = Sum of Partial Products
End Program

Algorithm for a Multiplier with a Ripple Adder:

Begin Program
Multiplier = 32 bits
Multiplicand = 32 bits
Register = 64 bits
Put the multiplier in the least significant half and clear the most significant half
  For i = 1 to 32
  Begin Loop
    If the least significant bit of the 64-bit register contains binary '1'
    Begin If
      Add the Multiplicand to the Most Significant Half using the RA
      Begin Adder
        C[0] = '0'
        For j = 0 to 31
        Begin Loop
          Calculate Sum S[j] = Multiplicand[j] ⊕ Most Significant Half[j] ⊕ C[j]
          Calculate Carries C[j+1] = Multiplicand[j] · Most Significant Half[j] + (Multiplicand[j] ⊕ Most Significant Half[j]) · C[j]
        End Loop
      End Adder
      Shift the 64-bit Register one bit to the right throwing away the least significant bit
    Else
      Only Shift the 64-bit Register one bit to the right throwing away the least significant bit
    End If
  End Loop
  Register = Sum of Partial Products
End Program

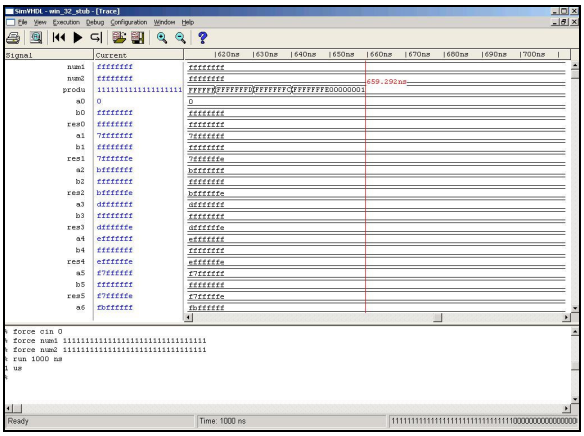

Fig. 1: Waveform and command line window for the 32-bit multiplier using a carry look ahead adder

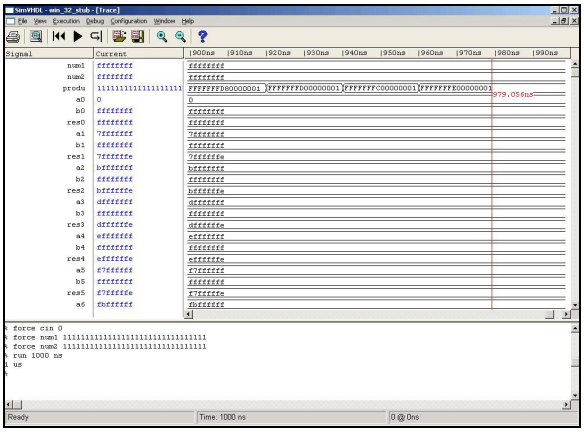

Fig. 2: Waveform and command line window for the 32-bit multiplier using a ripple adder

Future work: This study can be extended to cover signed multipliers using VHDL.

CONCLUSION

Two different multipliers using a fast carry-look-ahead adder and a ripple adder have been modeled and simulated using VHDL. The multiplier with a carry-look-ahead adder has shown a better performance over the multiplier with a ripple adder in terms of gate delays. In other words, the multiplier with the carry-look-ahead adder has approximately twice the speed of the multiplier with the ripple adder, under the worst case. In fact, the multiplier with the carry-look-ahead adder uses time = 659.292 ns, Fig. 1[9], while the multiplier with a ripple adder uses time = 979.056 ns, Fig. 2[9].

ACKNOWLEDGEMENT

We are very grateful to Dr. Belaid Moa for his valuable comments on this study.

REFERENCES

1. Sertbas, A. and R.S. Özbey, 2004. A performance analysis of classified binary adder architectures and the VHDL simulations. J. Elect. Electron. Eng., Istanbul, Turkey, 4: 1025-1030. http://www.istanbul.edu.tr/eng/ee/jeee/main/pages/issues/is41/41005.pdf .
2. Asadi, P. and K. Navi, 2007. A novel high-speed 54-54 bit multiplier. Am. J. Applied Sci., 4 (9): 666-672. http://www.scipub.org/fulltext/ajas/ajas49666-672.pdf .
3. Aziz, S.M., C.N. Basheer and J. Kamruzzaman, 2002. A synthesizable VHDL model for an easily testable generalized multiplier. In: Proceeding of the 1st IEEE International Workshop on Electronic Design, Test Application (DELTA.02), Jan. 29-31, 2002, IEEE Computer Society, Washington, DC, USA, pp: 504-506. http://portal.acm.org/citation.cfm?id=789090.789967&coll=GUIDE&dl=GUIDE&CFID=40436358&CFTOKEN=40390034
4. Stephen Brown and Zvonko Vranesic, 2005. Fundamentals of Digital Logic with VHDL Design. 2nd Edn. McGraw-Hill Higher Education, USA. ISBN: 0072499389.
5. William Stallings, 2006. Computer Organization and Architecture Designing for Performance. 7th Edn. Pearson Prentice Hall, USA. ISBN: 0-13-185644-8.
6. Armstrong, J.R. and F.G. Gray, 2000. VHDL Design Representation and Synthesis. 2nd Edn. Prentice Hall, USA. ISBN: 0-13-021670-4.
7. Zainalabedin Navabi, 2007. VHDL Modular Design and Synthesis of Cores and Systems. 3rd Edn. McGraw-Hill Professional, USA. ISBN: 9780071508926.
8. Wakerly, J.F., 2006. Digital Design-Principles and Practices. 4th Edn. Pearson Prentice Hall, USA. ISBN: 0131733494.
9. Software Simulation Package: DrirectVHDL, Version 1.2, 2007, Green Mounting Computing Systems, Inc., Essex, VT, USA. http://www.gmvhdl.com