International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:3, 2008

Implementation of Adder-Subtracter Design with VerilogHDL

May Phyo Thwal, Khin Htay Kyi, and Kyaw Swar Soe

***Abstract*—**According to the density of the chips, designers are trying to put so any facilities of computational and storage on single chips. Along with the complexity of computational and storage circuits, the designing, testing and debugging become more and more complex and expensive. So, hardware design will be built by using very high speed hardware description language, which is more efficient and cost effective. This paper will focus on the implementation of 32-bit ALU design based on Verilog hardware description language. Adder and subtracter operate correctly on both unsigned and positive numbers. In ALU, addition takes most of the time if it uses the ripple-carry adder. The general strategy for designing fast adders is to reduce the time required to form carry signals. Adders that use this principle are called carry look- ahead adder. The carry look-ahead adder is to be designed with combination of 4-bit adders. The syntax of Verilog HDL is similar to the C programming language. This paper proposes a unified approach to ALU design in which both simulation and formal verification can co-exist.

***Keywords*—**Addition, arithmetic logic unit, carry look-ahead adder, Verilog HDL.

I. Introduction

THIS paper is aimed to study and implement ALU design for 32-bit adder and subtracter based on Verilog hardware description language. The purpose of using Verilog HDL is that it offers many features for hardware design. When the actual addition is performed, there is no delay from waiting by using carry look-ahead adder. For any circuit larger than 4-bit, the carry look-ahead adder, the circuit becomes very complicated. So 32-bit carry look-ahead adder and subtracter will be constructed by combing 4-bit carry look-ahead adders.

II. Overview of ALU Design

The various circuits used to execute data-processing instructions are usually combined in a single circuit called an arithmetic logic unit or ALU. ALU is the part of a computer processor (CPU) that carries out arithmetic and logic operations on the operands in computer instruction words. ALU performs operations such as addition, subtraction and multiplication of integers and bitwise AND, OR, NOT, XOR and other Boolean operations. The CPU's instruction decode logic determine which particular operation the ALU should perform, the source of the operands and the destination of the result [1]. In some processors, the ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Computer arithmetic is commonly performed on two very different types of numbers integer and floating point. Computer programs calculate both positive and negative numbers. So a representation that distinguishes the positive from the negative is required. Every computer today uses twos-complement binary representation for complement binary numbers. In arithmetic unit, 32-bit ALU (addition and subtraction) design will be constructed based on Verilog HDL. The Verilog hardware description language is used to provide a gate level model and simulation of each design. A number of differing configurations of binary adders exist for inclusion into ALU design [1].

One of the most popular methods to reduce delay is to use a carry look-ahead mechanism. By using carry look-ahead mechanism, the propagation delay is reduced to four-gate level irrespective of the number of bits in the adder.

A. Implementation Program

Verilog HDL is a general-purpose hardware description language that is similar syntax to the C programming language. Verilog HDL allows different levels of abstraction to be mixed in the same model. So a hardware model can be defined in terms of switches, gates, RTL, or behavioral code. Today most digital design of processors and related hardware system is done using a hardware description language. Such a language serves two purposes, first it provides on abstract description of the hardware to simulate and debug the design. Second, with use of logic synthesis and hardware compilation tools, this description can be compiled into the hardware implementation. Verilog provides the concept of module. A module is the basic building block in Verilog. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details [2]. These modules can be substituted in place of the 32-bit full adder and subtracter modules described before without changing any other component of the simulation. The simulation results will be unchanged.

III. High Speed Adder

The general strategy for designing fast adders is to reduce the time required to form carry signals. One approach is to compute the input carry needed by stage i directly from carry like signals obtained from all the preceding stages i-1,i-2,……0,rather than waiting for normal carries to ripple slowly from stages to stages. Adders that use this principle are called carry

Manuscript received March 31, 2008. This work was supported in part by the Ministry of Science and Technology, Union of Myanmar.

look-ahead adders. An n-bit carry look-ahead adder is formed from n stages , each of which is basically a full adder modified by replacing its carry output line c_i by two auxiliary signals called g_i and p_i or generate and propagate, respectively ,which are defined by the following logic equation (1);

$$g_i = x_i y_i \qquad p_i = x_i + y_i \tag{1}$$

The name generate comes from the fact that stage i generates a carry of 1(c_i=1) independent of the value of c_{i-1} if both xi and yi are 1; that is, if $x_i y_i$=1.Stage i propagates c_{i-1}; that is makes c_i=1 in response to c_{i-1}=1 if x_i or y_i is 1__in order words, if x_i+y_i=1.Now the usual equation $c_i=x_i y_i + x_i c_{i-1} + y_i c_{i-1}$, denoting the carry signal c_i to be sent to stage i+1, can be rewritten in terms of g_i and p_i.

$$c_i = g_i + p_i c_{i-1} \tag{2}$$

Similarly, c_{i-1} can be expressed in terms of g_{i-1}, p_{i-1} and c_{i-2},

$$c_{i-1} = g_{i-1} + p_{i-1} c_{i-2} \tag{3}$$

On substituting Equation (3) into Equation (2),

$$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-2} \tag{4}$$

Continuing in this way, c_i can be expressed as a sum-of-products function of the p and g outputs of all the preceding stages. For example, the carries in a four-stage carry look-ahead adder are defined as follows:

$$c_0 = g_0 + p_0.c_{in}$$
$$c_1 = g_1 + p_1.g_0 + p_1.p_0.c_{in}$$
$$c_2 = g_2 + p_2.g_1 + p_2.p_1.g_0 + p_2.p_1.p_0.c_{in}$$
$$c_3 = g_3 + p_3.g_2 + p_3.p_2.g_1 + p_3 p_2.p_1.g_0 + p_3 p_2.p_1.p_0.c_{in} \tag{5}$$

Fig. 1 shows the general form of a carry look-ahead adder circuit designed in this way [3].

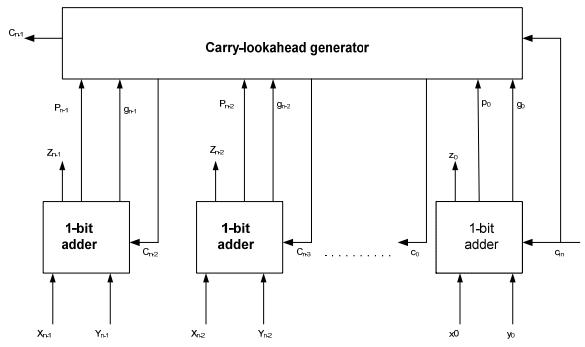

Fig. 1 Overall Structure of Carry Look-ahead Adder

A. Adder Expansion

The methods of handing carry signals in the two main combinational adder designs considered so far, namely, ripple carry propagation and carry look-ahead Fig. 1 can be extended to larger adders of the kind needed to execute add instructions in a 32-bit computer. If the n 1-bit (full) adder's stages are replaced in the n-bit ripple-carry design with n k-bit adders, an nk-bit adder can be obtained. Four 4-bit adders such as the 4-bit carry look-ahead circuit of Fig. 2 can be connected in this way to form the 16-bit adder appearing in Fig. 3.

This design represents a compromise between a 16-bit stage ripple-carry adder, which is cheap but slow, and a single-stage 16-bit carry look-ahead adder, which is fast, expensive, and impractical because of the complexity of its carry-generation logic. The circuit of Fig. 3 effectively combines sets of four $x_i y_i$ inputs into groups that are adder via carry look-ahead; the results computed by the various groups are then linked via ripple carries [3].

The components designed for 1-bit addition have been replaced with similar but larger components intended for 4-bit addition. The expanded design of Fig. 1 can be got. Again 1-bit adders with 4-bit adder are being replaced, but now each adder stage produces a propagate-generate signal pair pg instead of c_{out} and a carry look-ahead generator converts the four sets of pg signals to the carry inputs required by the four stages. The "group" g and p signals produced by each 4-bit stage are defined by [3].

$$g = x_i y_i + x_{i-1}\, y_{i-1}(x_i+y_i) + x_{i-2} y_{i-2}(x_i + y_i)\,(x_{i-1} + y_{i-1}) + x_{i-3}\, y_{i-3}(x_i+y_i)\,(x_{i-1} + y_{i-1})\,(x_{i-2} + y_{i-2}) \tag{6}$$
$$p = (x_i + y_i)\,(x_{i-1} + y_{i-1})\,(x_{i-2} + y_{i-2})\,(x_{i-3} + y_{i-3}) \tag{7}$$

It is not hard to show that the logic to generate the group carry signals cout, c11, c7 and c3 in Fig. 4 is exactly the same as that of the carry look-ahead generator of Fig. 2.

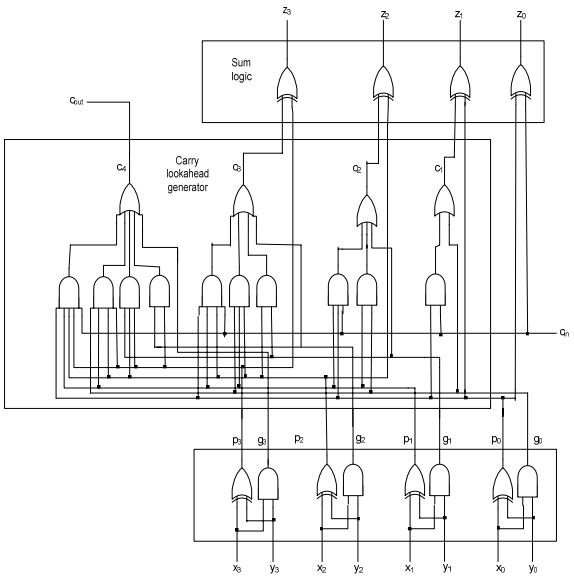

Fig. 2 A 4-Bit Carry Look-ahead Adder

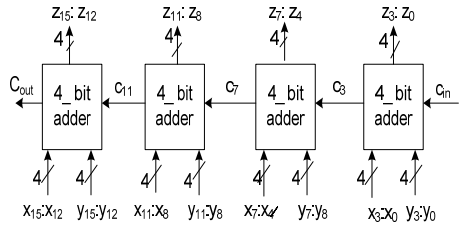

Fig. 3 16-Bit Composed of 4-Bit Adders Linked by Ripple Carry Propagation

IV. Design of Adder-Subtracter

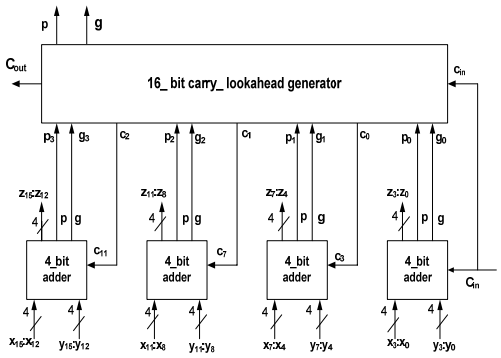

Fig. 4 16-Bit Adder Composed of 4-Bit Adders Linked by Carry Look-ahead

A two-complement adder-subtracter will be designed that computes the three quantities X+Y, X-Y, and Y-X, as well as overflow and zero flags. The design goal is to minimize the number of gates used; operation speed is not of concern. The circuit is required in several versions that handle different data word sizes, including 4, 8 and 16.

Assume that we have standard gate-level and 4-bit register-level components available as building blocks.

The lowest cost adders employ ripple-carry propagation and can easily provide access to the internal signals needed by the flags. Overflow detection uses c_{n-2} the input the sign position. In the first case overflow is indicated by a carry bit into the sign position that is, by $c_{n-2}=1$, since this indicates that the magnitude of the sum exceeds the n-1 bits allocated to it. A little thought shows that overflow from adding two negative numbers is indicated by $c_{n-2}=0$[4].The overflow condition is specified by the logic expression

$$V= \bar{x}_{n-1}\,\bar{y}_{n-1}\,c_{n-2} + x_{n-1}y_{n-1}\,\bar{c}_{n-2} \quad (8)$$

Now c_{n-1}, the carry output signal from the sign position, is defined by $x_{n-1}\,y_{n-1} + x_{n-1}\,c_{n-2} + y_{n-1}\,c_{n-2}$, from which it follows that

$$V=c_{n-1} \oplus c_{n-2}$$

This equation can be used to design overflow detection logic for two-complete addition or subtraction. Zero detection required access to all the sum outputs and poses no special problems. Fig. 5 (a) shows the logic diagram of an appropriate 4-bit ripple-carry adder.

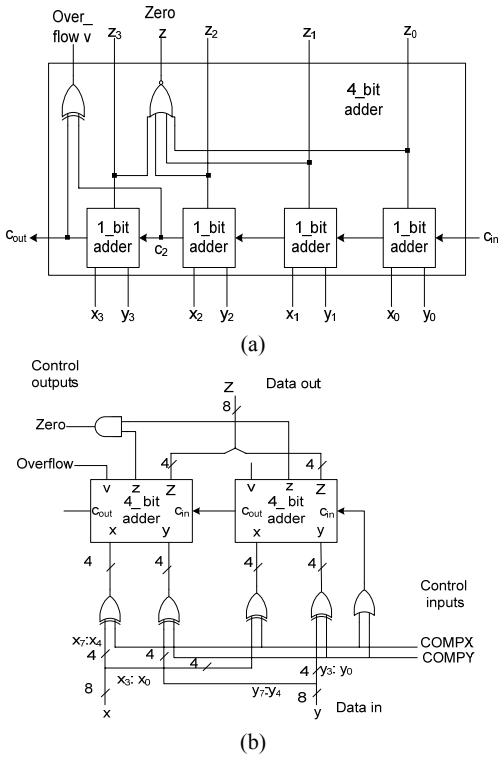

Fig. 5 Low-Cost Addition and Subtraction of Twos-Complement Numbers (a) 4-Bit Adder Module (b) 8-Bit Adder-Subtracter

V. 32-Bit Adder-Subtracter

The overflow flag is defined by Equation as $V= c_{n-1} \oplus c_{n-2}$and is realized here by an XOR gate. The zero flag is defined by Z= z3 +z2+z1+z0 and implemented by a NOR gate. The k copies of this adder can be used to produce a 4k-bit ripple- carry adder in the useful way. The overflow for the entire circuit is taken from the v output of the left-most (most significant) stage, while the z outputs of all the stages are ANDed to produce the zero flag. To extend the adder to an adder-subtracter, the design of Fig. 6 is a good starting point. It uses an XOR word gate to complement the X input, thereby enabling the circuit to compute X+Y and Y-X. To implement the third operation X-Y, a two-way 4-bit multiplexer could be inserted into each of the data -in bused so that both X and Y can be applied to each of the adder-subtracter's data inputs. A cheaper solution is to insert a second XOR word gate into the Y bus, enabling Y to be complemented independently. Then X-Y can be computed in the form $X+\bar{Y}+1$[4].

The complement design of an 8-bit adder-subtracter along the foregoing lines is depicted in Fig. 5 (b). It contains two 4-bit adders of the type in Fig. 5 (a) linked by their carry lines. Two lines COMPX and COMPY control the XOR gates that change X and Y to $\bar{x}$ and - $\bar{y}$, respectively. The OR gates sets the adder's carry_ in line to 1 during subtraction. A two-input AND gate combines the two z outputs to produce the zero flag, which is 1 if and only if the entire 8-bit result Z=0.

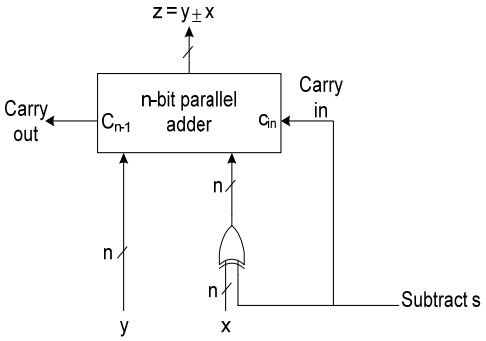

Fig. 6 An N-Bit Two-Complement Adder-Subtracter

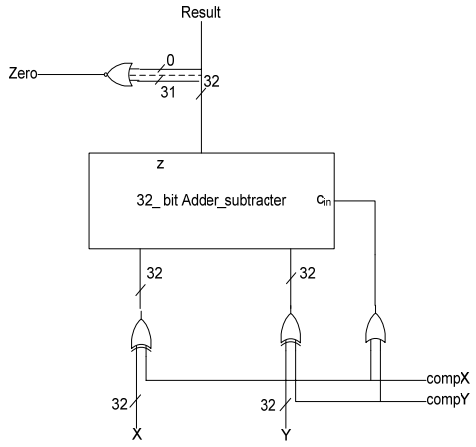

Fig. 7 32-Bit Adder-Subtracter

Three of the four signal combination on COMPX and COMPY control lines implement the desired three arithmetic functions. The fourth combination 11 implements the sum $\overline{x} + \overline{y} + 1$, which is an arithmetic function implemented by our design that has no obvious uses. In this paper, 32-bit adder-subtracter will be designed in Fig. 7.

VI. Implementation of Overflow Condition

Equation for overflow condition of 32-bit adder-Subtracter,

Overflow= (($\overline{X}$ [31]. $\overline{Y}$ [31].Result[31]. $\overline{compX}$. $\overline{compY}$)
+(X [31].Y[31]. $\overline{\mathrm{Re}\,sult}$ [31]. $\overline{compX}$. $\overline{compY}$)
+(X [31]. $\overline{Y}$ [31]. $\overline{\mathrm{Re}\,sult}$ [31]. $\overline{compx}$ .compY) +
($\overline{X}$ [31].Y [31].Result [31]. $\overline{compX}$. compY) +(X
[31]. $\overline{Y}$ [31].Result [31].compX. $\overline{compY}$)+
($\overline{X}$ [31].Y[31]. $\overline{\mathrm{Re}\,sult}$ [31].compX. $\overline{compY}$))

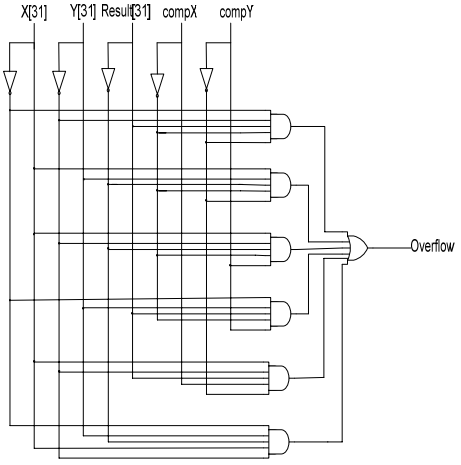

Fig. 8 Overflow Design

VII. Hierarchical Level for Program

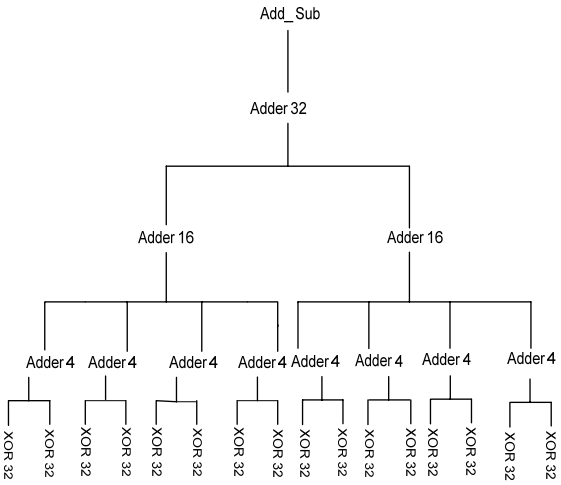

Fig. 9 Hierarchical Level for 32-Bit Carry Look-head Adder

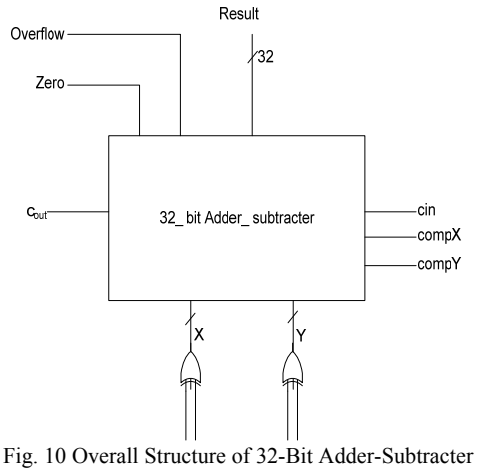

Fig. 10 Overall Structure of 32-Bit Adder-Subtracter

VIII. The Stimulus of Overall System Design

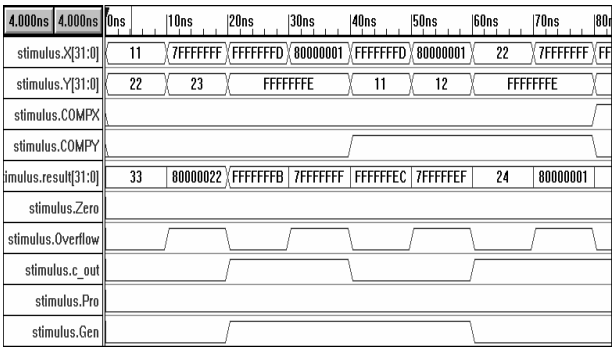

Fig.11 Stimulus of 32-Bit Adder-Subtracter

IX. Conclusion

In paper, 32-bit adder design presented can be implemented and simulated, at the gate level; with Verilog HDL .The first stage of the adder block is getting input data, which is 32-bit wide. After each stages of the adder block some extra bits are added because of the overflow. The final adder stage can give output data of 32-bit wide.

The general strategy for designing fast adders is to reduce the time required to form carry signals. Adders that use this principle are called carry look-ahead adder. The carry look-ahead is a fast adder but extremely large, especially when the operands are big. Carry look-ahead offers a faster path than waiting for the carries to ripple through all 32 1-bit adder. The faster path is paved by two signals, generate and propagate. The former creates a carry regardless of the carry input, and the other passes a carry along.

The disadvantage of carry look- ahead is that the carry logic is getting quite complicated for more than 4_bits.For that reason, carry look-ahead adders are usually implemented as 4_bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.

So 32-bit carry look-ahead adder and Subtracter will be constructed by combing 4-bit carry look-ahead adders.

Acknowledgment

We would like to thank U Kyaw Swar Soe and Dr. Win Aye for their comments on the early drafts of this paper. We would also like to thank the referees for their help in improving the clarity of the presentation.

References

[1] Andrew S. Tanenbaum, Structure Computer Organization, Fourth edition, Prentice_Hall, Inc., 1999.
[2] Samir Palnitkar, Verilog HDL: A Guide to Digital Design and Synpaper, Sun Microsystems, Inc., 1996.
[3] William Stallings, Computer Organization & Architecture, Sixth edition.
[4] David A. Patterson, John L. Hennessy, Computer Organization and Design, (The Hardware /Software Interface), Third edition, Elsevier Inc., 2005.