An adder design optimized for DCS logic

by A. Weinberger

The basic DCS logic gate provides a two-way SELECT function and, with modifications, a two-way XOR, OR, or AND function. Furthermore, outputs of DCS gates can be wired together (dotted) to perform dotted SELECT, XOR, OR, or AND functions. The versatility of this logic is illustrated in the design of a carry-lookahead adder.

Introduction

DCS (differential cascode current switch) logic [1] is based on a cell that provides a two-way SELECT as the primitive logic function. The cell becomes a two-way XOR, OR, or AND with rearranged inputs and/or reduced component use. By contrast, the more familiar ECL (emitter-coupled logic) begins with a simpler cell performing the simpler primitive logic functions OR/NOR. Cells are combined to perform SELECT or XOR functions, and even AND/NAND functions when inputs with proper polarity are unavailable.

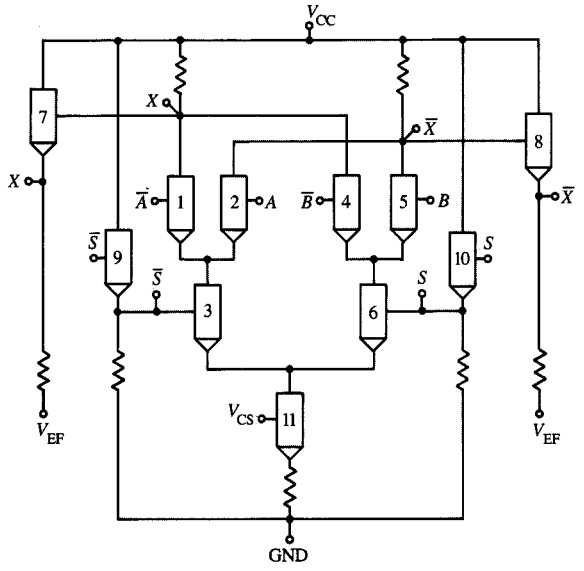
Efficient use of DCS requires exploitation of the SELECT and XOR functions. Complex computer logic functions include among their constituent parts recognizable SELECT and XOR functions. In less obvious cases, AND–OR or OR–AND combinations may be converted to SELECT or XOR functions. Moreover, major logic functions can at times be respecified in terms of SELECT functions for efficient implementation.

In this paper, a design of a carry-lookahead adder illustrates how the DCS functions of SELECT and XOR can be fully exploited. Not only are the obvious XOR functions of the adder used, but the carry-generate functions, which are critical to adder performance, are replaced by simpler pseudo-generate functions, and expressed as SELECT functions.

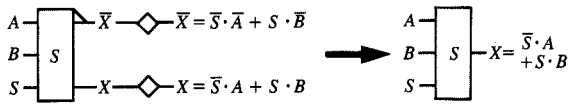
The first part of the paper describes DCS logic and compares it with ECL. The second part describes the carry-lookahead functions of an adder, showing the advantage of using the XOR instead of the inclusive OR as a constituent primitive function, and leading to the more desirable pseudo-generate function. Finally, a long path of a 32-bit adder is described, showing the pervasive use of SELECT and XOR functions in the adder.

DCS logic

DCS logic provides a two-way SELECT function as the basic logic gate. **Figure 1** shows a simplified circuit diagram together with a logic gate representation. The gate can also be used as a two-way XOR, OR, or AND function. If $\overline{B} = A$ and $B = \overline{A}$, $X = S \oplus A$. If transistors 4 and 5 are unused and collectors 6 and 2 are connected, $X = S + A$; if, in addition, inputs to 1 and 2 are exchanged as well as inputs to 3 and 6, $\overline{X} = S \cdot A$.



(a)



(b)

Figure 1

Two-way DCS SELECT cell: (a) circuit and (b) logic symbols.

Exchanging the true logic input with its complement has the effect of complementing the input.

The primitive gating functions SELECT, XOR, OR, and AND can also be implemented with dotting (wiring together of signals to perform logic). Dotting collector outputs ANDs them, while dotting emitter outputs ORs them. Collector dotting is done at the X and $\overline{X}$ outputs preceding the emitter followers, and emitter dotting is done at the outputs of the emitter-follower transistors (7 and 8). A combination of collector and emitter dotting implements the DCS dotting functions of SELECT-dot, XOR-dot, OR-dot, and AND-dot. To AND-dot, the principal outputs are collector-dotted and the complement outputs emitter-dotted, as in **Figure 2**. **Figure 3** shows an OR-dot. **Figure 4** shows an XOR-dot, using two copies of the gates to be dotted, while **Figure 5** shows a SELECT-dot.

The primary advantage of DCS over the more familiar ECL (emitter-coupled logic) is significantly reduced power for average logic functions of comparable performance,

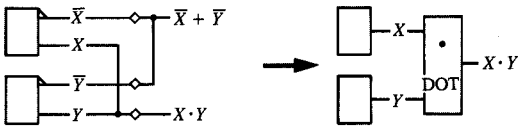


Figure 2

DCS AND-dot.

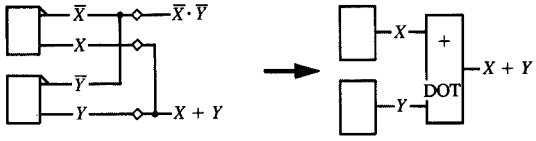


Figure 3

DCS OR-dot.

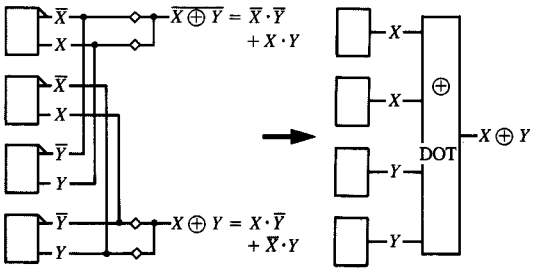


Figure 4

DCS XOR-dot.

particularly for SELECT functions and the derivative XOR functions. It is especially advantageous for latches, noting that a simple latch is a two-way SELECT function in which a clock signal selects the new latch output between a new input and the prior latch output.

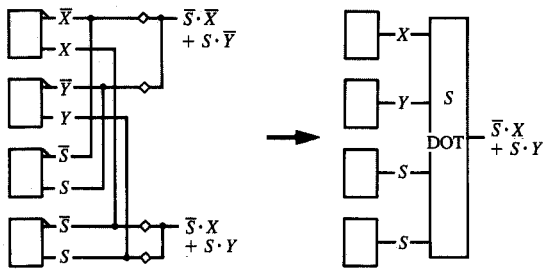


Figure 5

DCS SELECT-dot.

The advantage of DCS over ECL in combinational logic depends on the function. SELECT and XOR functions favor DCS, while AND and OR functions favor ECL. To show this more clearly, the logic differences between DCS and ECL are explained.

Logically, DCS differs from ECL in three important aspects. First, DCS uses differential inputs instead of reference-controlled inputs. In Figure 1, current is switched between transistors 1 and 2 by means of the dual inputs $\overline{A}$ and A. In ECL, one of the inputs, say $\overline{A}$, is replaced by a reference voltage requiring a larger signal swing for A, but obviating the need for dual inputs. In addition, the remaining input, A, can be replaced with a parallel set of inputs to perform an OR: Namely, A is replaced by $A_1 + \cdots + A_N$. It should be pointed out, however, that the dual inputs of DCS avoid the problem with ECL where a signal must pass through an inverter stage if one of its destinations encounters a polarity mismatch.

The second difference is that DCS uses cascoding to perform logic, as in Figure 1, which shows a two-level cascode logic tree comprising transistors 1 through 6. It is converted to ECL logic by eliminating transistors 3 through 6 and level shifters 9 and 10 (including their resistors). Logic is performed using transistors 1 and 2, with $\overline{A}$ replaced by a reference voltage and A expanded to an OR of inputs. X and $\overline{X}$ provide the OR and NOR outputs, respectively.

A third difference between DCS and ECL is in dotting (wired-connection) logic. In ECL, a collector output may be AND-dotted with collector outputs of other ECL gates prior to entering an emitter follower, and emitter-follower outputs may be further OR-dotted with other emitter-follower outputs. Thus, an ECL stage can perform one to three levels of logic: a NOR/OR gate level and up to two levels of dotting, AND dotting followed by OR dotting. A DCS stage can perform one or two levels of logic: a gating level and an optional dotting level. The gate can perform more complex functions than ECL (SELECT, XOR, AND, or OR, instead of only OR/NOR), although the DCS gate is limited to a two-way OR, whereas ECL permits a higher limit. The single dotting level is a versatile logic dot (SELECT-dot, XOR-dot, AND-dot, or OR-dot) implemented with an actual sequence of collector-dot and emitter-dot.

DCS carry-lookahead adder

DCS logic permits a new way of optimizing the design of a parallel adder [2], particularly the parallel carry functions generally known as carry-lookahead and typically defined as

$$G_{i,j} = G_i + \hat{P}_i \cdot G_{i+1} + \cdots + \hat{P}_i \cdot \cdots \cdot \hat{P}_{j-1} \cdot G_j , \quad (1)$$

$$\hat{P}_{i,j} = \hat{P}_i \cdot \hat{P}_{i+1} \cdot \cdots \cdot \hat{P}_j , \quad (2)$$

where

$G_{i,j}$ = carry-generate of bit group i through j, high to low order,
$G_i = A_i \cdot B_i$ = carry-generate from bit i
$\hat{P}_i$ = carry-propagate through bit i
= either H_i (the exclusive propagate = $A_i \oplus B_i$) or P_i (the inclusive propagate = $A_i + B_i$),
A_i, B_i = adder inputs to bit i.

In ECL logic, the inclusive OR, P_i, is used as the propagate signal because it is easier to implement than the exclusive OR. Little difference in ease of implementation exists in DCS, because both are primitive gating functions.

A significant advantage of H_i over P_i is that H_i, and the extended propagate function $H_{i,j}$ (= $H_i \cdot H_{i+1} \cdot \cdots \cdot H_j$), can be used as a select signal to simplify the carry-generate functions. For example,

$$G_{1,2} = G_1 + H_1 \cdot G_2 = \overline{H_1} \cdot G_1 + H_1 \cdot G_2 . \quad (3)$$

In the second expression H_1 selects between G_1 and G_2. Note, however, that G_1 can be replaced by either A_1 or B_1, because $\overline{H}_1 \cdot (A_1 \text{ or } B_1) = (\overline{A}_1 \cdot \overline{B}_1 + A_1 \cdot B_1) \cdot (A_1 \text{ or } B_1)$ $= A_1 \cdot B_1 = G_1$.

Any carry-generate function can be replaced by a simpler function, a pseudo-generate function pG, if it is ANDed with the complement carry-propagate function. As a result, the carry-lookahead can be designed with pG and H functions. For a single bit, $pG_i = (A_i \text{ or } B_i)$, as shown above.

A multibit pseudo-generate function is equal to the carry-generate function except that the low-order subgroup carry-generate function is replaced by the pseudo-generate function. For example,

$$pG_{1,2} = \overline{H_1} \cdot pG_1 + H_1 \cdot pG_2 , \quad (4)$$

where $\overline{H_1} \cdot pG_1 = G_1$.

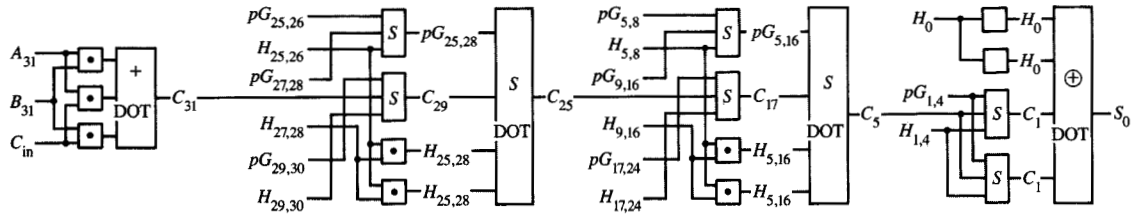


Figure 6

Long path of a DCS four-stage carry-lookahead adder.

If it is implemented directly from the adder inputs,

$$pG_{1,2} = A_1 \cdot B_1 + A_1 \cdot A_2 + B_1 \cdot A_2$$
$$= A_1 \cdot B_1 + A_1 \cdot B_2 + B_1 \cdot B_2 , \quad (5)$$

whereas $G_{1,2} = A_1 \cdot B_1 + A_1 \cdot A_2 \cdot B_2 + B_1 \cdot A_2 \cdot B_2$.

Similarly,

$$pG_{1,4} = \overline{H_{1,2}} \cdot pG_{1,2} + H_{1,2} \cdot pG_{3,4} , \quad (6)$$

where $\overline{H_{1,2}} \cdot pG_{1,2} = G_{1,2}$.

We further take advantage of SELECT-dotting to perform two select functions in one stage. For example,

$$pG_{1,8} = \overline{H_{1,4}} \cdot pG_{1,4} + H_{1,4} \cdot pG_{5,8}$$
$$= \overline{H_{1,4}} \cdot (\overline{H_{1,2}} \cdot pG_{1,2} + H_{1,2} \cdot pG_{3,4})$$
$$+ H_{1,4} \cdot (\overline{H_{5,6}} \cdot pG_{5,6} + H_{5,6} \cdot pG_{7,8}). \quad (7)$$

Either $H_{1,4}$ or $\overline{H_{1,4}}$ can be used in a SELECT-dot to select between the two parenthesized select functions.

A carry C is also generated as a function of pseudo-generate and carry-propagate functions. For example,

$$C_1 = \overline{H_{1,4}} \cdot pG_{1,4} + H_{1,4} \cdot C_5$$
$$= \overline{H_{1,4}} \cdot (\overline{H_{1,2}} \cdot pG_{1,2} + H_{1,2} \cdot pG_{3,4})$$
$$+ H_{1,4} \cdot (\overline{H_{5,6}} \cdot pG_{5,6} + H_{5,6} \cdot C_7). \quad (8)$$

Application to a 32-bit adder

Figure 6 shows a long path of a four-stage 32-bit adder. Inputs consist of addend A ($= A_0, \cdots, A_{31}$, high-to-low order), augend B ($= B_0, \cdots, B_{31}$), and input carry C_{in}. Outputs consist of a sum S ($= S_0, \cdots, S_{31}$) and an output carry $C_{out} = C_0$. In the first stage,

$$C_{31} = A_{31} \cdot B_{31} + A_{31} \cdot C_{in} + B_{31} \cdot C_{in} . \quad (9)$$

In the same stage, the two-bit pseudo-generate and carry-propagate functions are generated, the former as in Equation (5) with appropriate subscripts. For example,

$$pG_{25,26} = A_{25} \cdot B_{25} + A_{25} \cdot A_{26} + B_{25} \cdot A_{26} , \quad (10)$$

$$H_{25,26} = (A_{25} \oplus B_{25}) \cdot (A_{26} \oplus B_{26}), \quad (11)$$

where the OR in Equation (10) is implemented as an OR-dot and the AND in Equation (11) as an AND-dot.

In the second stage, C_{25} is generated according to Equation (8) with appropriate subscripts, together with other pseudo-generate and carry-propagate functions of up to eight bit groups. For example,

$$pG_{17,24} = \overline{H_{17,20}} \cdot (\overline{H_{17,18}} \cdot pG_{17,18} + H_{17,18} \cdot pG_{19,20})$$
$$+ H_{17,20} \cdot (\overline{H_{21,22}} \cdot pG_{21,22} + H_{21,22} \cdot pG_{23,24}), \quad (12)$$

$$H_{17,24} = (H_{17,18} \cdot H_{19,20}) \cdot (H_{21,22} \cdot H_{23,24}). \quad (13)$$

Similarly, C_5 is generated in the third stage as

$$C_5 = \overline{H_{5,16}} \cdot (\overline{H_{5,8}} \cdot pG_{5,8} + H_{5,8} \cdot pG_{9,16})$$
$$+ H_{5,16} \cdot (\overline{H_{17,24}} \cdot pG_{17,24} + H_{17,24} \cdot C_{25}), \quad (14)$$

together with other lower-order carries.

In the last stage, a sum S_i is produced as an XOR-dot of H_i with a select gate that generates C_{i+1}. For example,

$$S_0 = H_0 \oplus C_1 = H_0 \oplus (\overline{H_{1,4}} \cdot pG_{1,4} + H_{1,4} \cdot C_5). \quad (15)$$

Comparison to an ECL adder

The DCS adder described above is designed for maximum performance, achieved with a critical path delay of four stages. It is considerably faster than a performance-optimized ECL adder, also achieved with a four-stage path. (See Eichelberger and Bello [1] for a performance

and power comparison of DCS and ECL gates. DCS power includes one current source, a pair of emitter followers, and a pair of level shifters. ECL power includes one current source and one emitter follower.)

Even at higher performance, the DCS adder retains a significant power advantage. The DCS adder uses 241 current sources, 118 pairs of emitter followers, and 195 pairs of level shifters. (Dotting and shared-level shifters in some fixed complex functions reduce the number of emitter followers and level shifters.) The ECL adder uses 320 current sources and 212 emitter followers. Assuming all high-power circuits (e.g., 5.8 mW for a three-way NOR in ECL and 7.1 mW for a two-way SELECT in DCS*), the DCS adder uses roughly 80% of the power of the ECL adder.

For a DCS adder with performance comparable to that of the fastest ECL adder, the power can be reduced by using lower-power gates or designing the adder with a longer path and, therefore, fewer gates.

Summary and conclusions

DCS logic is based on the SELECT function and the derivative XOR, AND, and OR functions, as well as on the corresponding dotting functions. Its versatility can be used to advantage in a variety of logic applications, of which the carry-lookahead adder has been highlighted.

In general, applications that use or can be defined for use with SELECT and XOR functions are particularly suitable for DCS. This includes combinational logic as well as latches, which are basically SELECT functions.

References

1. E. B. Eichelberger and S. E. Bello, "Differential Current Switch—High Performance at Low Power," *IBM J. Res. Develop.* **35,** 313–320 (1991, this issue).
2. A. Weinberger and J. L. Smith, "A One-Microsecond Adder Using One-Megacycle Circuitry," *IRE Trans. Electron. Computers* **EC-5,** 65–73 (1956).

Received September 7, 1990

Arnold Weinberger *IBM Data Systems Division, P.O. Box 950, Poughkeepsie, New York 12601.* Mr. Weinberger received his B.S.E.E. from City College of New York and pursued graduate work at the University of Maryland. He began his computer career at the National Bureau of Standards and in 1960 joined IBM, first in Yorktown and later in Poughkeepsie. He is a Life Fellow of the IEEE. Mr. Weinberger holds 22 patents and 85 patent publications. For the past several years, his efforts have been directed primarily toward evaluating the relative logic power of new technologies and developing efficient methods for their use.

*V. L. Gani, IBM Data Systems Division, Poughkeepsie, NY, private communication.