DON'T-CARE OPTIMIZATION
William Bricken
October 1995

Some paths within a circuit are never traversed. When these happen to be the topologically longest paths, the timing of the circuit is not optimal. Here is an example of the kind of algebraic transformations we are doing to optimize timing by eliminating don't-care paths.

Example: two-bit carry by-pass adder

The carry-bypass adder is a ripple-carry adder with two extra gates that bypass the ripple-carry chain when the carry bits are 1, thus shortening the most critical path in the circuit.

The circuit algebraically:

inputs: a0 a1 b0 b1 c0
outputs: s0 s1 c1
gates are numbered in PUN format:

```
1 = ( (a0 b0) ((a0)(b0)) )         xor
2 = ((a0)(b0))                     and
3 = ( (a1 b1) ((a1)(b1)) )         xor
4 = ((a1)(b1))                     and
5 = ( (c0 1) ((c0)(1)) )           xor     s0
6 = ((c0)(1))                      and
7 = 2 6                            or
8 = ( (3 7) ((3)(7)) )             xor     s1
9 = ((3)(7))                       and
10= ((1)(3))                       and     bypass
11= 4 9                            or
12= ( (10 11) (c0 (10)) )          mux     c1   bypass
```

What follows is the Losp reduction and restructuring of this circuit.

COALESCE

```
A = (a0 b0)                       nor
B = (a1 b1)                       nor
2 = ((a0)(b0))                    and
4 = ((a1)(b1))                    and
6 = ((c0)(1))                     and

1 = ( A 2 )                       xor
3 = ( B 4 )                       xor
5 = ( (c0 1) 6 )                  xor     s0
7 = 2 6                           or
8 = ( (3 7) ((3)(7)) )            xor     s1
9 = ((3)(7))                      and
10= ((1)(3))                      and     bypass
11= 4 9                           or
12= ( (10 11) (c0 (10)) )         mux     c1   bypass
```

EXPAND (substitute 1,3,7,9,11)

```
A = (a0 b0)                       nor
B = (a1 b1)                       nor
2 = ((a0)(b0))                    and
4 = ((a1)(b1))                    and
6 = ((c0) A 2)                    and

5 = ( (c0 (A 2)) 6 )              xor     s0
8 = ( ((B 4) 2 6) (B 4 (2 6)) )   xor     s1
10= (A 2 B 4)                     and     bypass
12= ((10 4 (B 4 (2 6))) (c0 (10))) mux    c1   bypass
```

EXPAND (substitute 6,10)

```
A = (a0 b0)                       nor
B = (a1 b1)                       nor
2 = ((a0)(b0))                    and
4 = ((a1)(b1))                    and

5 = ( (c0 (A 2)) ((c0) A 2) )                        s0
8 = ( ((B 4) 2 ((c0) A 2)) (B 4 (2 ((c0) A 2))) )    s1
12= (((A 2 B 4) 4 (B 4 (2 ((c0) A 2)))) (c0 A 2 B 4))   c1
```

```
SIMPLIFY (8,12)

8 = ( ((B 4) 2 ((c0) A 2)) ( B 4 (2 ((c0) A 2))) )      s1
8 = ( ((B 4) 2 ((c0) A  )) ( B 4 (2 ((c0) A  ))) )      s1

12= (((A 2 B 4) 4 (B 4 (2 ((c0) A 2)))) (c0 A 2 B 4))   c1
12= (((A 2 B  ) 4 (B   (2 ((c0) A  )))) (c0 A 2 B 4))   c1


COALESCE

A = (a0 b0)                         nor
B = (a1 b1)                         nor
2 = ((a0)(b0))                      and
4 = ((a1)(b1))                      and
C = (A 2)                           nor
D = (A (c0))                        nor

5 = ( (c0 C) ((c0) (C)) )                               s0
8 = ( ((B 4) 2 D) ( B 4 (2 D)) )                        s1
12= ((((C) B) 4 (B (2 D))) (c0 (C) B 4))                c1


COALESCE

A = (a0 b0)                         nor
B = (a1 b1)                         nor
2 = ((a0)(b0))                      and
4 = ((a1)(b1))                      and
C = (A 2)                           nor
D = (A (c0))                        nor
E = (B 4)                           nor
F = (D 2)                           nor

5 = ( (c0 C) ((c0) (C)) )                               s0
8 = ( (E (F)) ((E) F) )                                 s1
12= ((((C) B) 4 (B F)) (c0 (C) (E)))                    c1


FLEX 8

8 = (E F) ((E)(F))
```

FACTOR 12

```
12 = ((((C) B) 4 (B F)) (c0 (C) B 4))
   = 4 ( (((C) B) (B F)) (c0 (C) B) )
   = 4 ( B (C (F)) (c0 (C) B) )
   = 4 ( B (C (F)) (c0 (C)  ) )
   = 4 ( B (C 2 D) (c0 (C)  ) )

   = 4 (B ((A 2) 2 (A (c0))) (c0 A 2) )
   = 4 (B ((A  ) 2 (A (c0))) (c0 A 2) )
   = 4 (B ((A  ) 2         ) (c0 A 2) )
   = 4 (B (2 (A (c0 A))) )
   = 4 (B (2 (A (c0  ))) )
   = 4 (B (2 D) )
   = 4 (B F)
```

READING THE RESULT

```
A = (a0 b0)                        nor
B = (a1 b1)                        nor
2 = ((a0)(b0))                     and
4 = ((a1)(b1))                     and
C = (A 2)                          nor
D = (A (c0))                       nor
E = (B 4)                          nor
F = (D 2)                          nor

5 = ( (c0 C) ((c0)(C)) )                          s0
8 = (E F) ((E)(F))                                s1
12= 4 (B F)                                       c1


A & 2:  a0 = b0
B & 4:  a1 = b1
C:      a0 ≠ b0
E:      a1 ≠ b1
5:      c0 ≠ C
8:       E = F


c1 = 4 (B (2 (A (c0))))      hierarchical dependencies
```