COL215 Digital Logic and System Design

Divider Design

13 September 2017

Division: example

0011	← Q	
0100 | 00001101	← A	
B		
0000000	←	$0 \times B \times 2^3$
00001101		
000000	←	$0 \times B \times 2^2$
00001101		
01000	←	$1 \times B \times 2^1$
00000101		
0100	←	$1 \times B \times 2^0$
00000001	← R	

Unsigned Division

$$A = Q \times B + R$$

R = A; Q = 0; D = B
for i in 0 to n – 1 loop
    if (D x $2^{n-i-1} \leq$ R) then
        R = R - D x 2^{n-i-1}
        Q_{n-i-1} = 1
    else Q_{n-i-1} = 0
    end if
end loop

Unsigned Division

$$A = Q \times B + R$$

$R = A$; $Q = 0$; $D = B$
for i in 0 to $n - 1$ loop
  if ($D \times 2^{n-i-1} \leq R$) then
    $R = R - D \times 2^{n-i-1}$
    $Q_{n-i-1} = 1$
  else $Q_{n-i-1} = 0$
  end if
end loop

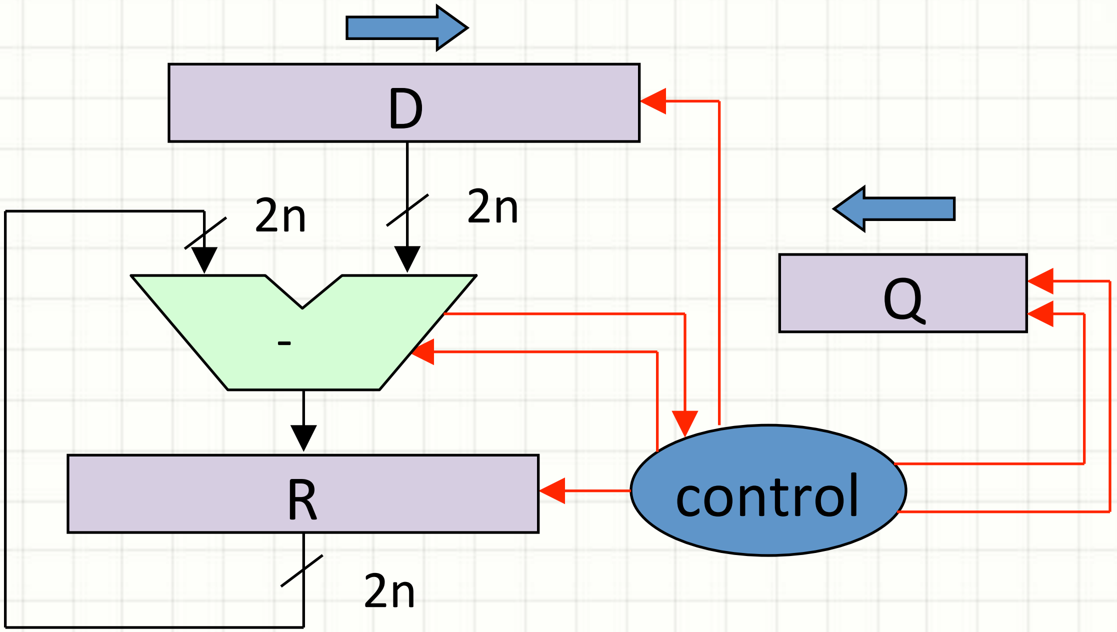
can this be avoided ?

Introducing shift registers

$$A = Q \times B + R$$

```
R = A; Q = 0; D = B x 2^(n-1)
for i in 0 to n – 1 loop
    if (D ≤ R) then
        R = R – D
        Q = 2 x Q + 1
    else Q = 2 x Q
    end if
    D = D / 2
end loop
```

Divider design - 1



Reducing subtractor size

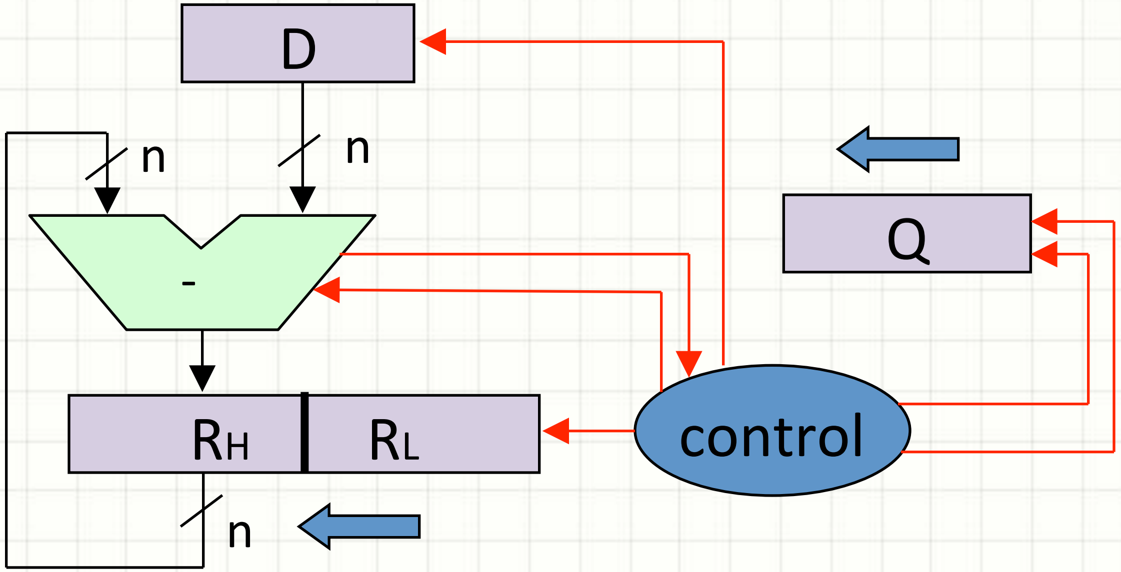
$$A = Q \times B + R$$

```
R = A; Q = 0; D = B
for i in 0 to n – 1 loop
   R = 2 x R
   if (D ≤ R_H) then
      R_H = R_H – D
      Q = 2 x Q + 1
   else Q = 2 x Q
   end if
end loop
```

Division: example

```
             0011     ←  Q
0100 | 00001101       ←  A
  ↑    00011010
  |    0000           ←  0 x B
  B    00011010
       00110100
       0000           ←  0 x B
       00110100
       01101000
       0100           ←  1 x B
       00101000
       01010000
       0100           ←  1 x B
       00010000       ←  R
```

Divider design - 2

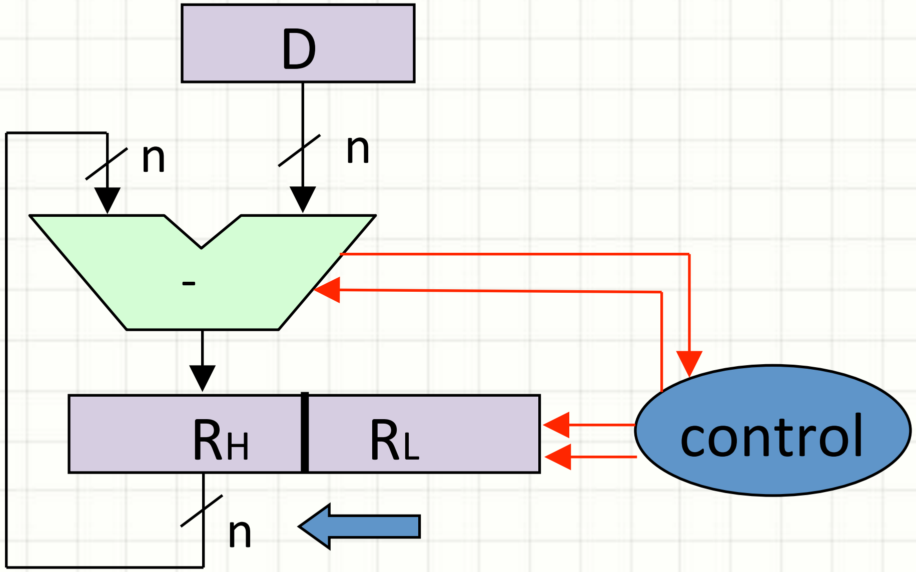


Reducing registers

$$A = Q \times B + R$$

```
R = A; D = B
for i in 0 to n – 1 loop
   R = 2 x R
   if (D ≤ R_H) then
      R_H = R_H – D
      R = R + 1
   end if
end loop
# R_H = remainder, R_L = quotient
```

Divider design - 3



Restoring division

- Non-restoring division
  - First compare the remaining dividend with divisor
  - Subtract only if dividend is large enough
- Restoring division
  - Subtract without comparing
  - Add (restore) the divisor if subtraction result negative

Signed multiplication/division

- Handle sign and magnitude separately
- Directly multiply/divide signed integers

Dividend	Divisor	Quotient	Remainder
+	+	+	+
-	+	-	-
+	-	-	+
-	-	+	-

Direct signed multiplication

- Use a common expression representing the values of positive as well as negative integers

$$B = -B_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} B_i \cdot 2^i$$

Common expr for +/- integers

$$\text{for } B \geq 0, B = \sum_{i=0}^{n-1} B_i \cdot 2^i = -B_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} B_i \cdot 2^i \quad (\because B_{n-1} = 0)$$

$$\text{for } B < 0, B = -|B|$$

$$\text{now } |B| = 2^n - \sum_{i=0}^{n-1} B_i \cdot 2^i$$

$$\therefore B = -2^n + B_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} B_i \cdot 2^i$$

$$= -B_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} B_i \cdot 2^i \quad (\because B_{n-1} = 1)$$

Direct signed multiplication

$$B = -B_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} B_i \cdot 2^i$$

$$= -B_{n-1} \cdot 2^{n-1} + B_{n-2} \cdot 2^{n-2} + \cdots + B_0 \cdot 2^0$$

$$= -B_{n-1} \cdot 2^{n-1} + 2B_{n-2} \cdot 2^{n-2} - B_{n-2} \cdot 2^{n-2} + \cdots + 2B_0 \cdot 2^0 - B_0 \cdot 2^0$$

$$= -B_{n-1} \cdot 2^{n-1} + B_{n-2} \cdot 2^{n-1} - B_{n-2} \cdot 2^{n-2} + \cdots + B_0 \cdot 2^1 - B_0 \cdot 2^0$$

$$= \sum_{i=0}^{n-1} (B_{i-1} - B_i) \cdot 2^i \qquad \text{where } B_{-1} = 0$$

$$\therefore A \cdot B = \sum_{i=0}^{n-1} A \cdot (B_{i-1} - B_i) \cdot 2^i$$

Booth’s algorithm

Comparing with unsigned case

unsigned multiplication

B_i	operation
0	no addition
1	add A

signed multiplication

B_i,	B_{i-1}	operation
0	0	no addtion
0	1	add A
1	0	subtract A
1	1	no addition

Original motivation for Booth’s algorithm

Reduce the number of additions required

. . . . . 0 0 0 1 1 1 1 1 1 0 0 0 . . . . .

add subtract

$$
\begin{array}{rccccccc}
+ & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

What have we learnt?

- Logic design (combinational circuits) [2, 4]
  - truth tables, expressions, circuits, VHDL
- Logic design (sequential circuits) [7, 8]
  - state transition tables, diagrams, circuits, VHDL
- Combinational & sequential modules [6, 7]
  - mux, demux, decoders, encoders, VHDL
  - flip-flops, registers, counters, VHDL
- From logic to arithmetic [5]
  - representations, conversions
  - operations and operators (add, subtract, compare, multiply, divide)

What lies ahead?

- Technology [3]
  - transistor to FPGA and things in between
- A little more theory [4, 8, 9]
  - minimizing logic, minimizing states
- System design [10]
  - from algorithmic description to circuits
  - control-data partition
- Testing [11]
  - testing tools, design for testability

Thanks