

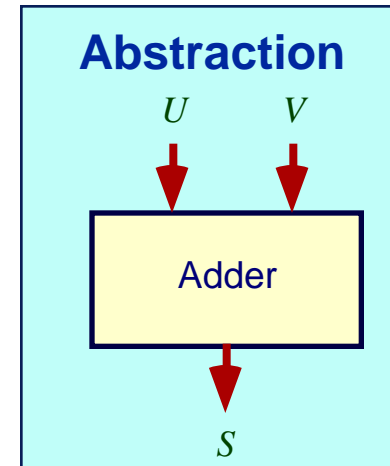
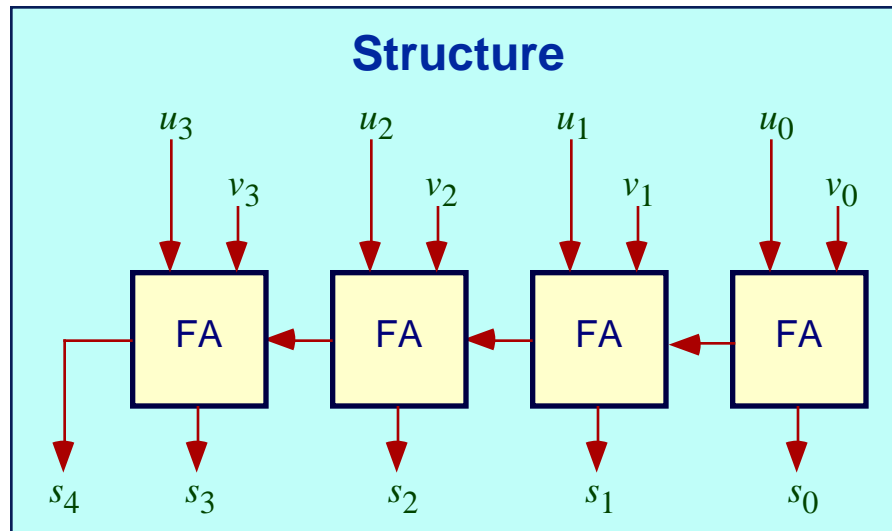
Implementing Computer Arithmetic

Randal E. Bryant

CS 15-347
Jan. 27, 1998

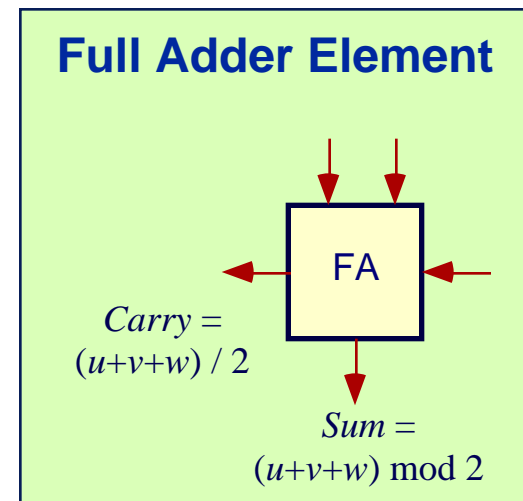
- ◆ Fast Addition
- ◆ Division, Pentium style

Conventional Adder Design

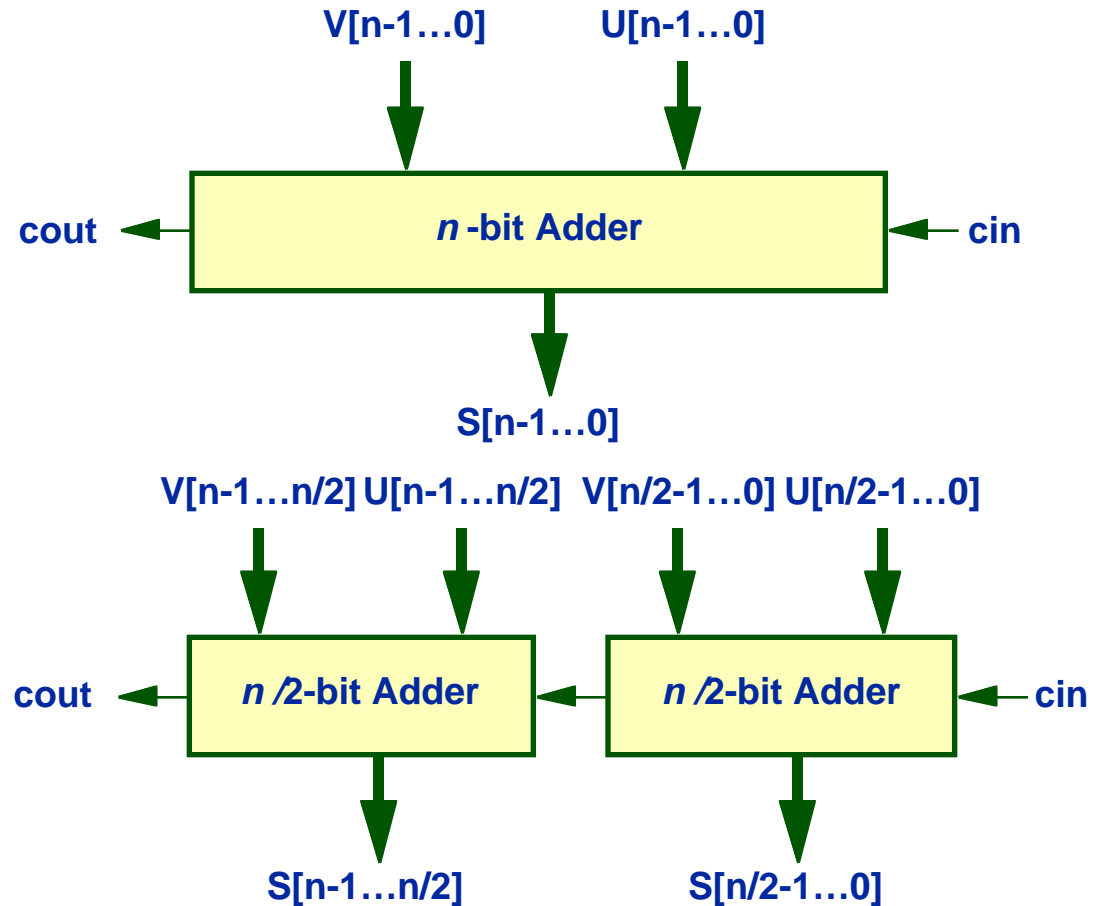


Shortcoming

- ◆ Must propagate carries across all cells
- ◆ Can improve with more costly hardware



Divide & Conquer?



◆ This is not an improvement

Carry Function Reformulation

Carry Function

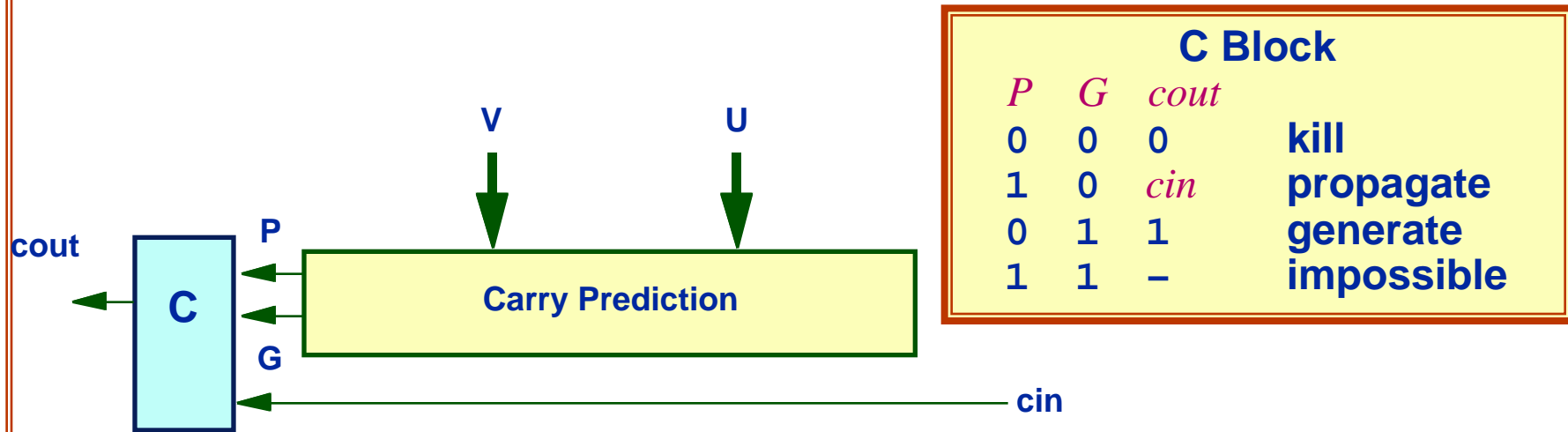
<i>u</i>	<i>v</i>	<i>cin</i>	<i>cout</i>
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Alternate Formulation

<i>u</i>	<i>v</i>	<i>cout</i>	
0	0	0	kill
1	0	<i>cin</i>	propagate
0	1	<i>cin</i>	propagate
1	1	1	generate

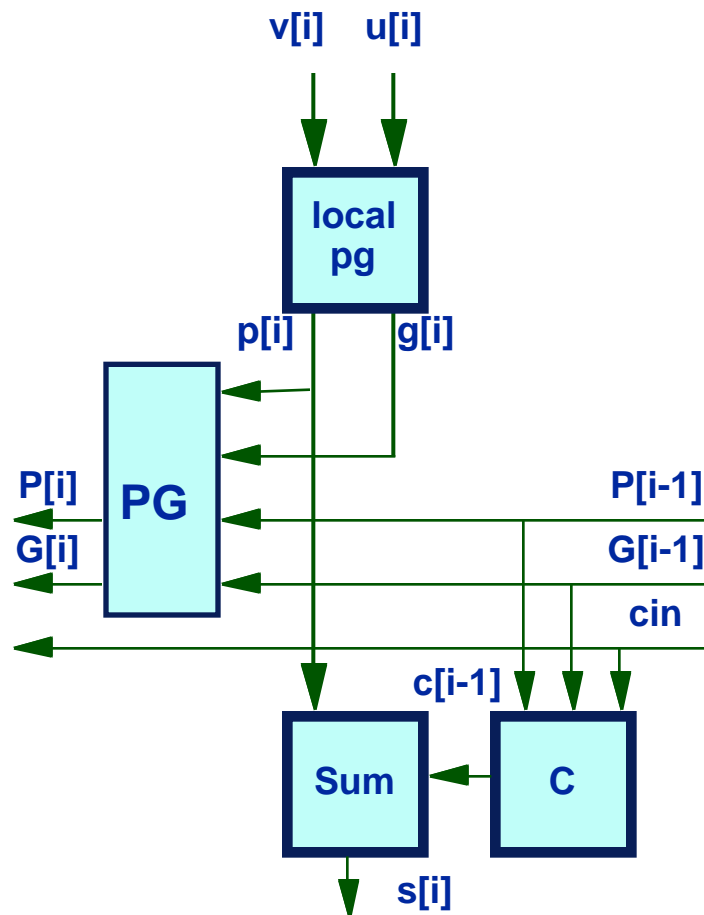
- ◆ When both local inputs 0, no carry
- ◆ When one is 0, the other is 1, propagate carry input
- ◆ When both are 1, then generate a carry

Block Level Carry



- ◆ P indicates that carry propagates through block
- ◆ G indicates that block generates carry

Reformulated Adder Cell



local pg			
$u[i]$	$v[i]$	$p[i]$	$g[i]$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

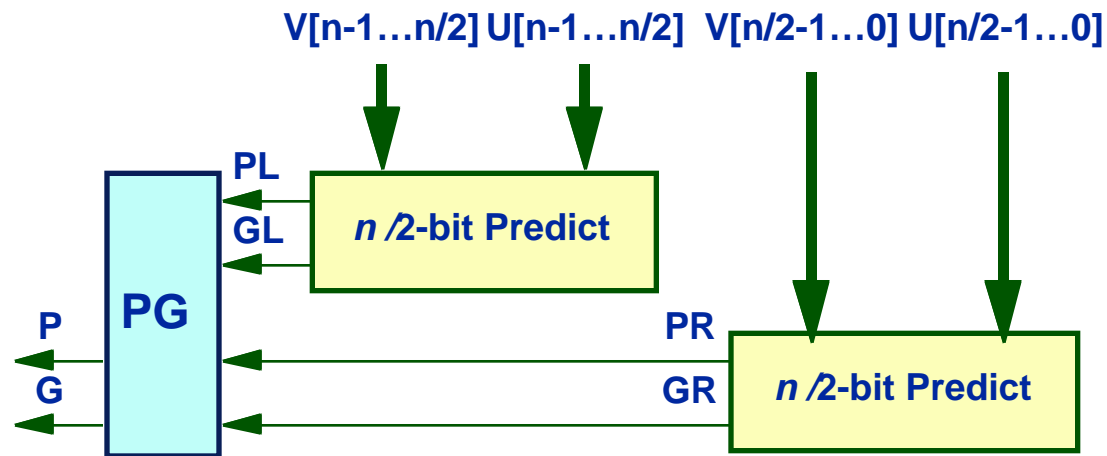
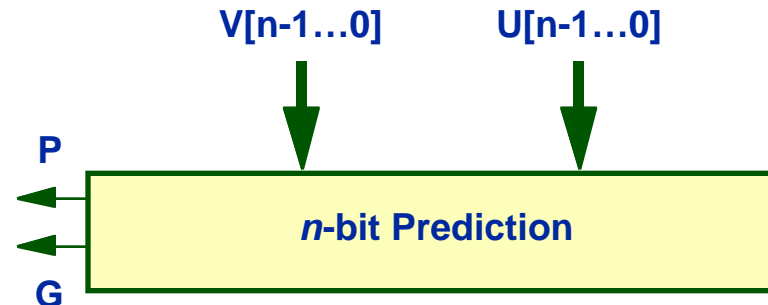
Half Adder

Carry Prediction: PG			
$p[i]$	$g[i]$	$P[i]$	$G[i]$
0	0	0	0
1	0	$P[i-1]$	$G[i-1]$
0	1	0	1

Sum		
$p[i]$	$c[i-1]$	$s[i]$
0	0	0
0	1	1
1	0	1
1	1	0

Xor

D & C Carry Prediction

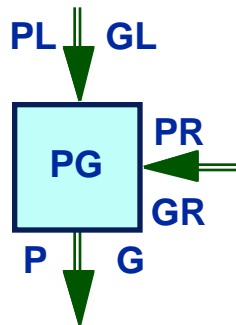


Carry Prediction: PG

PL	GL	P	G
0	0	0	0
1	0	PR	GR
0	1	0	1

◆ Divide & conquer yields shallow circuit

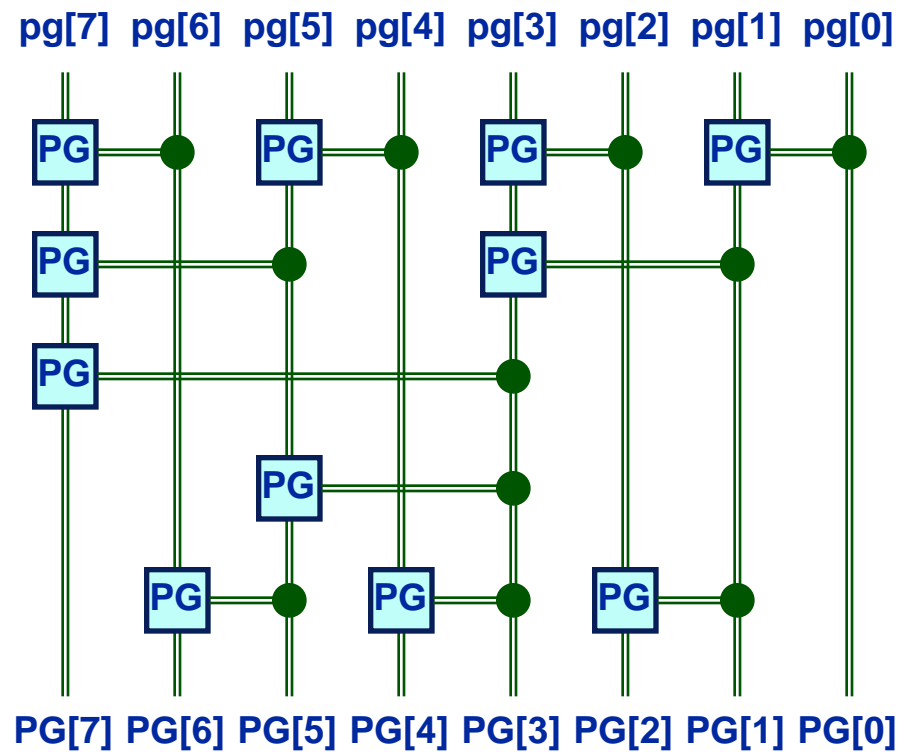
Carry Prediction Tree



- ◆ $O(n)$ complexity
- ◆ $O(\log n)$ depth

Generalization

- ◆ Parallel Prefix
- ◆ Any associative operation



Division, Pentium Style

Summary

- ◆ **Pentium is Intel's mainstream microprocessor**
 - 3.3 Million transistors
 - More advanced PentiumPro fills high end niche
- ◆ **Early versions all had error in floating point division hardware**
 - 5 missing transistors, fixed with change to single mask
- ◆ **Disclosed largely via Internet**
- ◆ **Intel ultimately offered replacements to everyone**
 - \$475 Million charge from 4Q94 revenue

Outline

- ◆ **Brief chronology**
- ◆ **Technical details**
- ◆ **Overall impact**

Discovery and Disclosure

Events

- ◆ **Prof. Thomas Nicely, Lynchburg College, VA**
 - Looking at properties of “twin primes”
 - Incorrect reciprocals for 824633702441 and 824633702443
 - » ~ Single precision accuracy (4×10^{-9})
 - Contacted others on Oct. 30, '94
- ◆ **Spreading of Information on Internet news group `comp.sys.intel`**
 - Terje Mathisen of Norway posts Nicely's findings on Nov. 3
 - Andreas Kaiser of Germany finds 23 bad reciprocals, Nov. 10
- ◆ **Tim Coe, Vitesse Semiconductor, Nov. 16**
 - Created (good enough) software model of flawed divide algorithm
 - Discovered (nonreciprocal) cases with error up to 6×10^{-5}
 - Later showed 1738 cases with less than single precision accuracy

Intel Plays Tough

- ◆ **Claimed had discovered in Summer '94**
 - Logged as minor bug, to be fixed on next revision
- ◆ **Andy Grove “posted” to Internet, Nov. 24**
 - Stating that other posters are overreacting
 - » All chips have flaws
 - Different Intel employee actually did the post
- ◆ **Agrees to replace, but only to those who can justify need**
- ◆ **Nov. 30: Report made available via WWW**
 - Describes algorithm and error
 - Estimates average user will encounter once in 27,000 years

The Uproar

IBM Breaks Ranks

- ◆ Produces report stating error may be encountered as much as once every 24 days, Dec. 12
 - Nonuniform number distribution
 - 4000X more divides per day than Intel estimates
- ◆ Stops shipment of all Pentium-based PCs
- ◆ Some question IBM's motives

Internet

- ◆ Hundreds of posts daily to `comp.sys.intel`
- ◆ Arguments on both sides
- ◆ Angered about Intel's attitude

Popular Press

- ◆ Starts following controversy in November
- ◆ Generally accepts Intel's description as "obscure error"

Resolution

Free Replacement Policy, Dec. 20

- ◆ No need to argue need
- ◆ Complex logistics
 - Many different versions
 - Actual replacement easy

Intel Not Humbled

- ◆ Still state that error is “technically an extremely minor problem”

Actual Replacement Rate Low

- ◆ 10% business users, 2% home users
- ◆ Must guarantee \$500 with credit card
 - Charged only if defective chip not sent back

Numerical Basics

Floating Point Numbers

- ◆ Numerical Form: $-1^s m 2^e$
 - Mantissa m either 24, 53, or 64 bits

Given

- ◆ Dividend mantissa p
- ◆ Divisor mantissa d
- ◆ Normalized forms: $1.\text{xxxxx}_2$
 - Range Constraint: $1 \leq p, d < 2$

Compute

- ◆ Quotient mantissa q
 - To sufficient precision
- ◆ Exponent
 - Based on exponents of dividend & divisor

Radix 4 Division

Conventional “Restoring” Algorithm

- ◆ Select quotient digit q_i
– 0, 1, 2, or 3
- ◆ Subtract $q_i d$ from p
- ◆ Shift p left

	Divisor	d
X 1	1	301
X 2	3	202
X 3	11	103

- ◆ Must consider all digits to select digit q_i
- ◆ Must form “awkward multiple” $3d$

Example			
p_0	1.2130	q_1	0
$-q_1 d$	- 0.		
	<hr/>		
	1.2130		
	←		
p_1	12.1300	q_2	3
$-q_2 d$	-11.1030		
	<hr/>		
	1.0210		
	←		
p_2	10.2100	q_3	2
$-q_3 d$	- 3.2020		
	<hr/>		
	1.0020		
	←		
	10.0200		

SRT Division

History

- ◆ Radix 4 version due to Atkins (1968)
- ◆ Commonly used by others, first time for Intel

Goals

- ◆ Predict quotient digit based on incomplete information about partial remainder and divisor
- ◆ No awkward multiples

Key Ideas of Method

- ◆ Quotient Digits $-2, -1, 0, +1, +2$
 - Redundancy allows imperfect digit prediction
 - Power of two multiples
 - » Implement with shift and/or complement

SRT Iteration Step

Given

◆ **Partial Remainder**

p_i

– Two's complement representation

– Can bound range $-16/3 < p_i < 16/3$

◆ **Divisor**

d

– Range Constraint: $-8/3 d < p_i < 8/3 d$

P

┌───────────┐
SXXX.XXXXXX ... x_2

D

┌───────────┐
1.XXXXXXX ... x_2

Select

◆ **Quotient Digit**

q_{i+1}

– By table lookup

– Using truncated values P and D

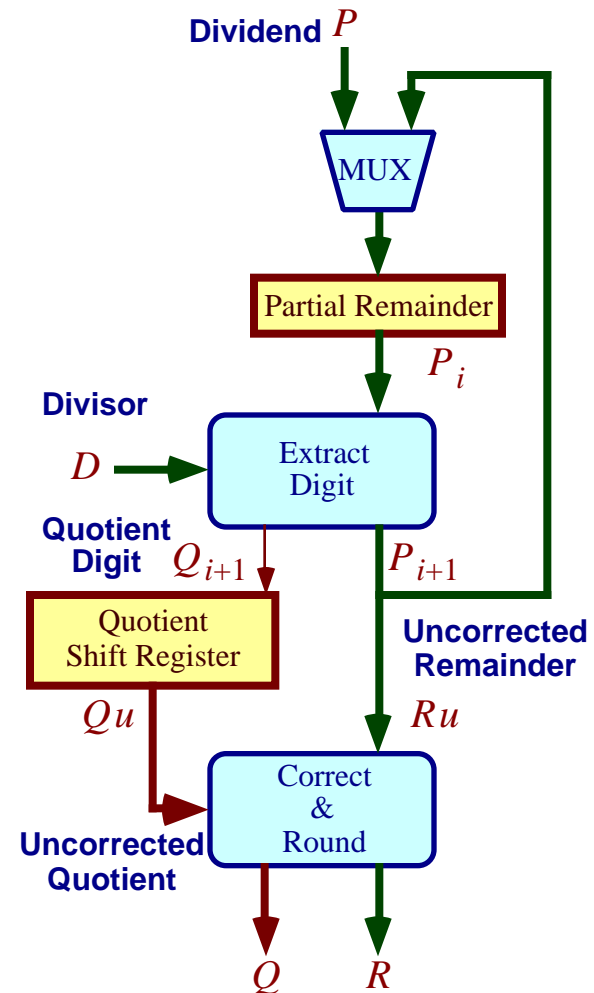
◆ **Compute**

$$p_{i+1} = 4 [p_i - q_{i+1} d]$$

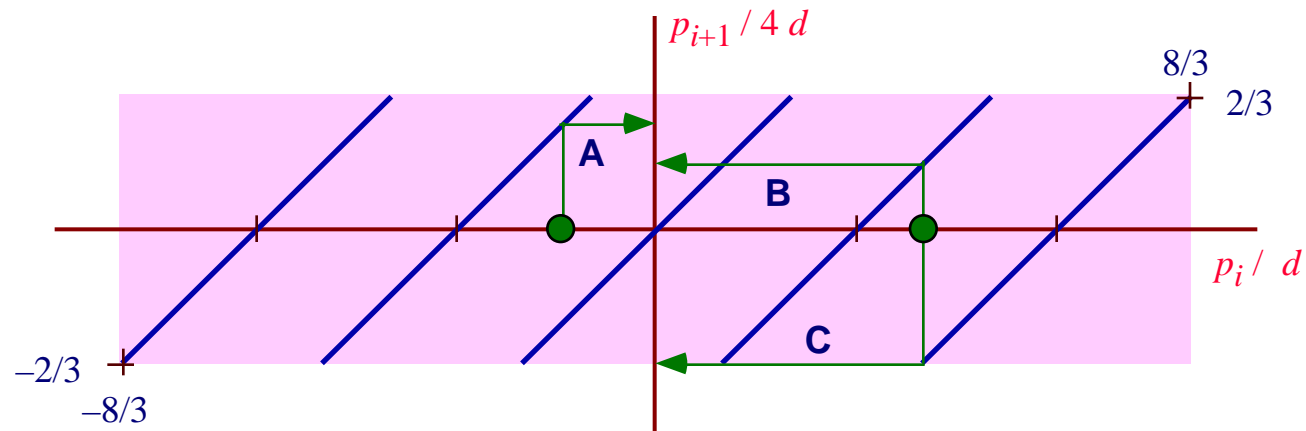
SRT Divider Circuit

Registers

- ◆ **Hold partial remainder**
 - Initially loaded with dividend
- ◆ **Divisor (not shown)**
 - Unchanged for entire computation
- ◆ **Quotient shift register**
 - Accumulates digits from iterations



Quotient Digit Selection



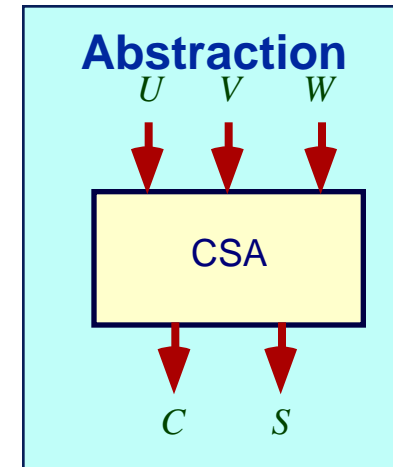
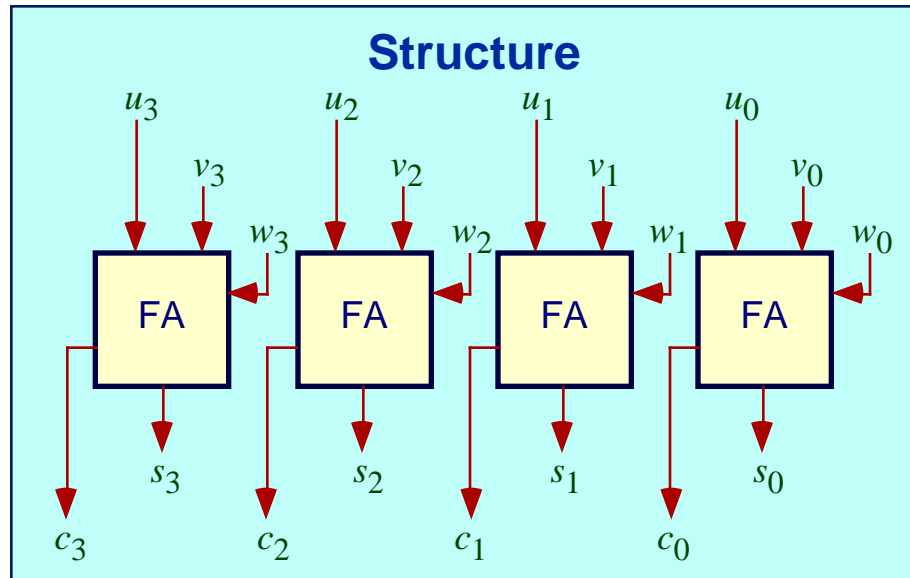
Examples

	p_i	q_{i+1}	p_{i+1}
A	$-7/15 d$	-1	$4 \lceil 8/15 \rceil d = 32/15 d$
B	$4/3 d$	$+1$	$4 \lceil 1/3 \rceil d = 4/3 d$
C	$4/3 d$	$+2$	$4 \lceil -2/3 \rceil d = -8/3 d$

Invariant Preserved

$$-8/3 d \leq p_{i+1} \leq 8/3 d$$

Carry Save Adders



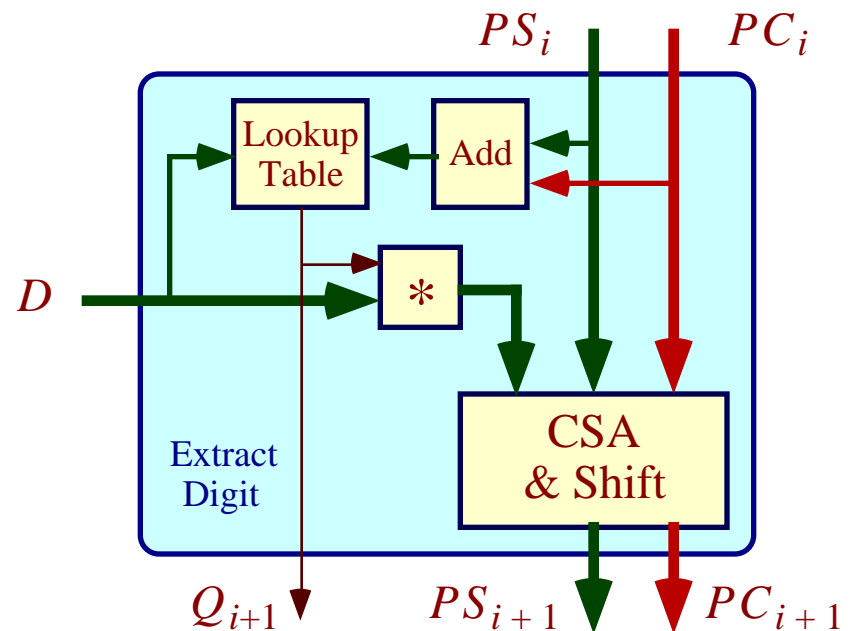
$$S + 2C = U + V + W$$

- ◆ Rearrange elements to eliminate carry chain
- ◆ Three input words
- ◆ Two output words
- ◆ 3 to 2 reducer

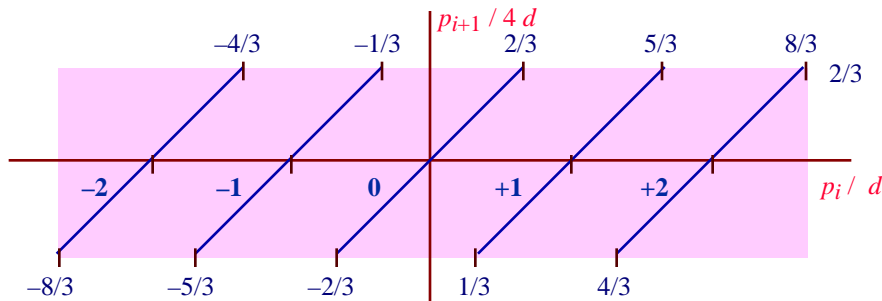
SRT Division Step

Elements

- ◆ Carry Save Adder to subtract weighted divisor
- ◆ 7-bit adder to get estimate of partial remainder
 - High order bits
- ◆ Lookup table to determine quotient digit
 - 4 “interesting” bits of divisor
 - » Leading bit always 1
 - Implemented as PLA
 - Where Intel made mistake
- ◆ Shift/complementer to weight divisor
 - Multiply by quotient digit

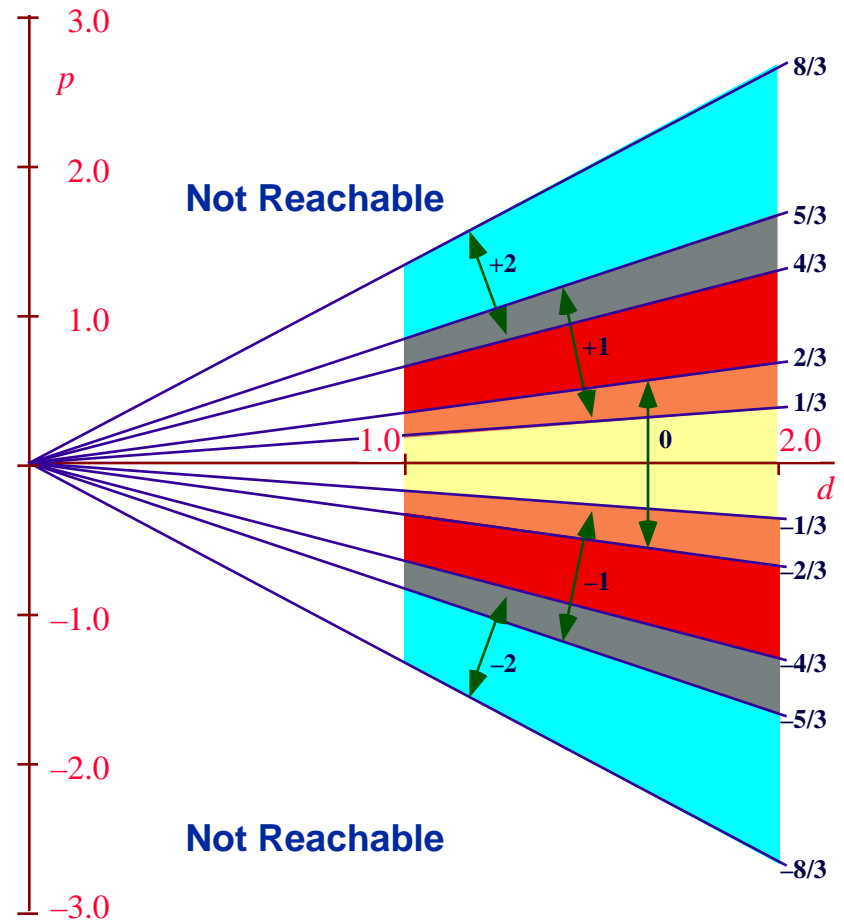


Lookup Table Design



Digit Selection

- ◆ Depends on ratio of d and p
- ◆ Redundancy leads to overlapping regions
 - May choose either digit



Discretization of P & D

Values

- ◆ **Least Resolvable Unit (LRU)**

- Weight of least significant bit

- ◆ **Partial Remainder** p_i

- Approximate by P

- Potentially underestimates by two LRUs

- One each from pc and ps

- ◆ **Divisor** d

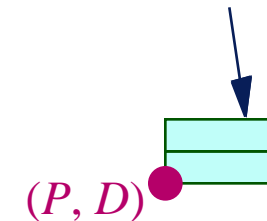
- Approximate by D

- Potentially underestimates by one LRU

P
SXXX.XXXXXX ...

D
1.XXXXXX ...

Range of Values for (p_i, d)

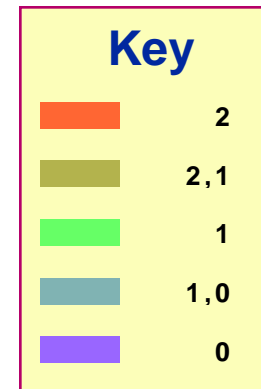
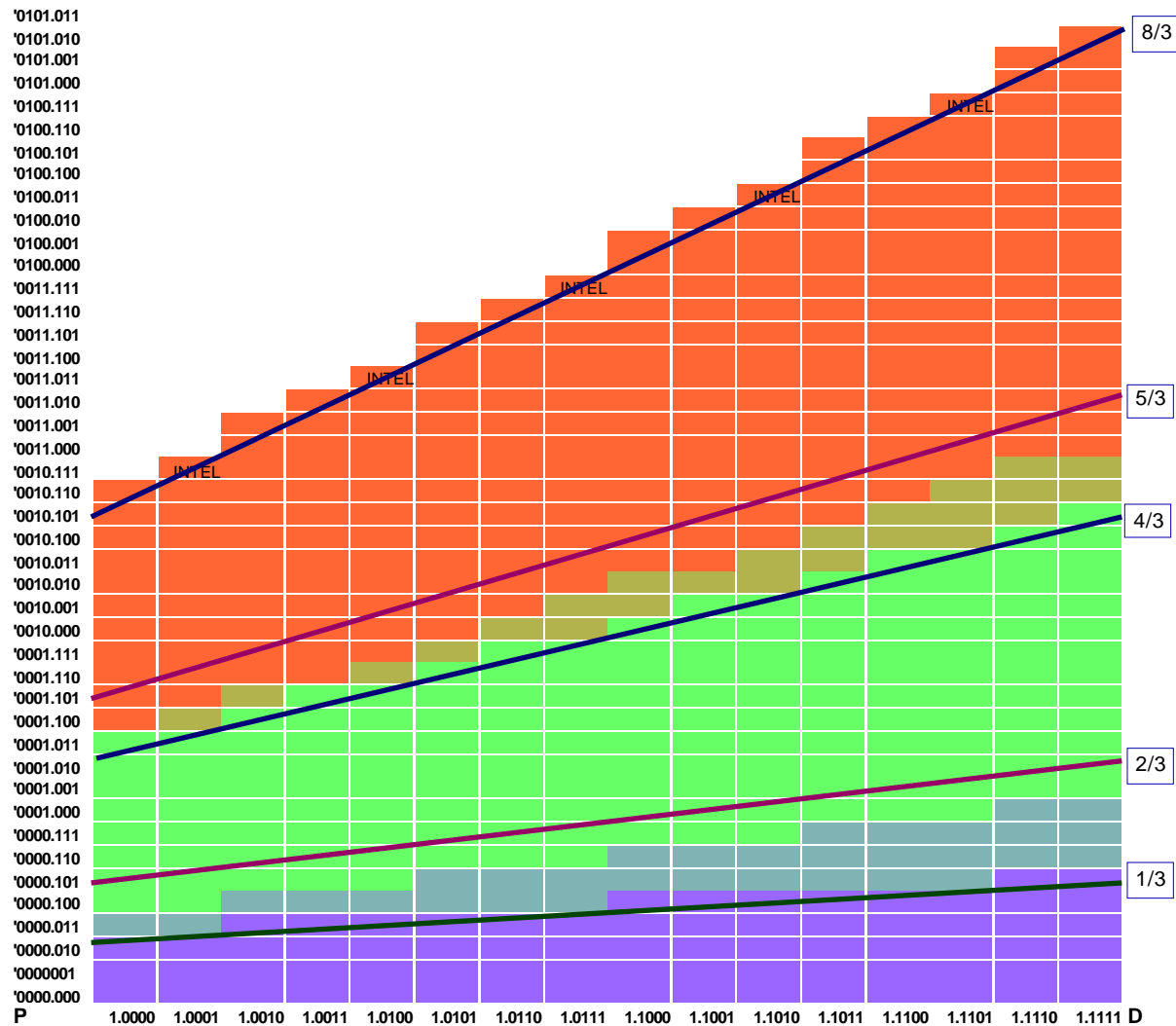


Compensating for Inaccuracy

- ◆ **Redundancy enables valid digit selection for all cases**






- As long as have enough bits of p_i and d

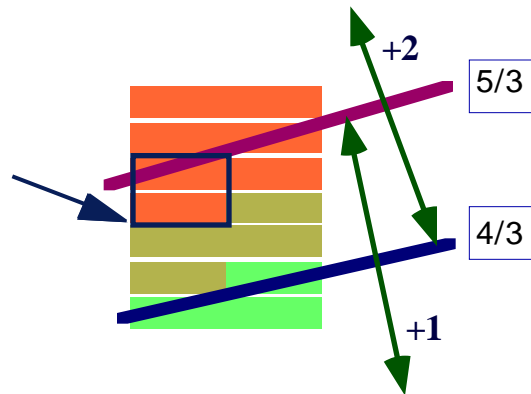
Valid PD Table Entries



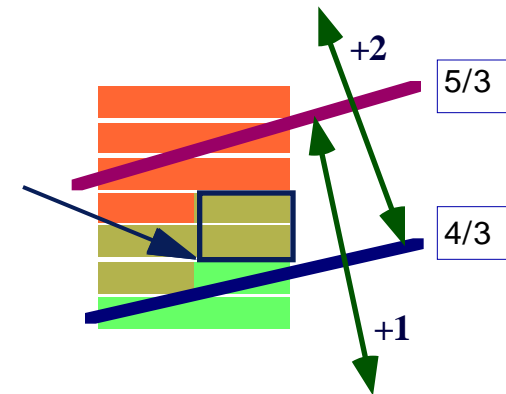
Nuances in PD Table

Unique value required.
Even though choice at (P,D)

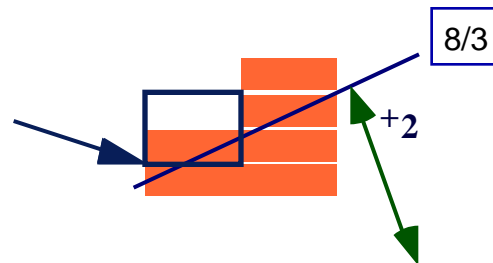
Key	
	2
	2,1
	1
	1,0
	0



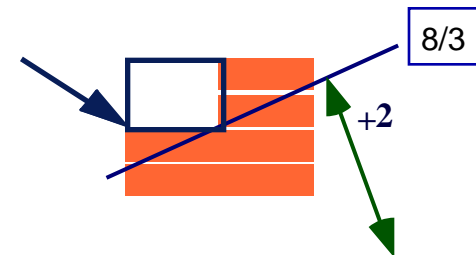
Either 2 or 1 OK



Require table entry
Even though (P,D)
out of range



No table entry required.
All values out of range



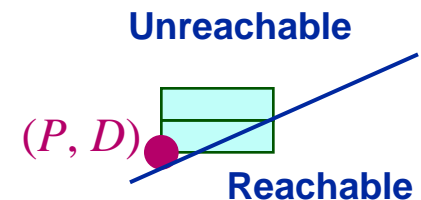
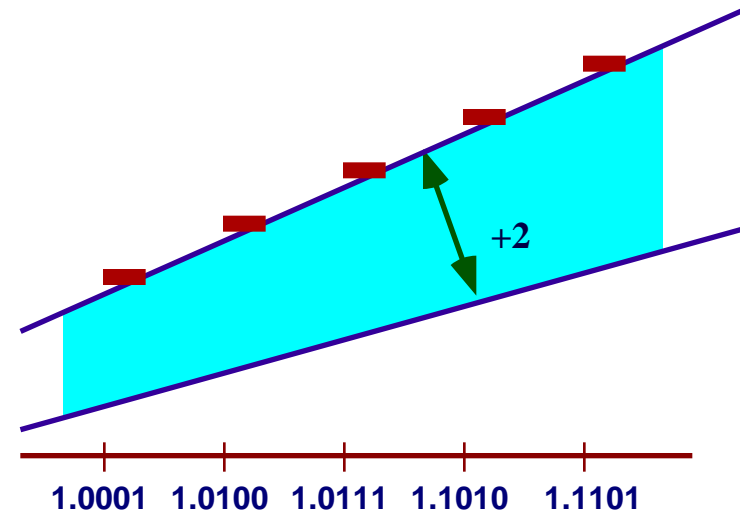
Intel's Table Errors

Actual Bug

- ◆ Erroneous entries for 5 cases
- ◆ Table generates 0 instead of +2

Occurs Only under Marginal Conditions

- ◆ (P, D) appear to be out of range
- ◆ D must under-estimate d
- ◆ P must closely estimate p_i
 - Unlikely with carry-save format



Example of Error

Correct Behavior

$$\begin{array}{r}
 p_0 \quad 0100.111000 \quad q_1 \quad +2 \\
 + - q_1 d \quad \underline{1100.010010} \\
 \quad \quad 0001.001010 \\
 p_1 \quad 0100.101000
 \end{array}$$

Incorrect Behavior

$$\begin{array}{r}
 p_0 \quad 0100.111000 \quad q_1 \quad 0 \\
 + - q_1 d \quad \underline{0000.000000} \\
 \quad \quad 0100.111000 \\
 p_1 \quad 0011.100000
 \end{array}$$

Leading digits drop off end

Divisor d

X	1	0001.110111
X	2	0011.101110
X	-1	1110.001001
X	-2	1100.010010

- ◆ Once hit bad table entry, cannot recover
- ◆ Contrary to some statements in press
 - Not “Self Correcting” !

When Can Error Occur?

Requires at Least 7 Iterations

- ◆ Hard to get to bad positions
- ◆ Worst case error 6×10^{-5}

Only for “At Risk” Divisors

- ◆ Stay in single column for entire division
- ◆ Remaining part of divisor should have lots of 1's
 - To ensure that D underestimates d

◆ Coe’s Example

– $d = 3145727_{10}$

10111111111111111111111111111111



At Risk!

– Correct result: 1.33382044913624100

– Pentium’s result: 1.33373906890203759

◆ Nicely’s Example

– $d = 824633702441_{10}$ 1011111111111111111111111111111100000101001

How Serious is the Error?

Intel's Claims

- ◆ **Most applications don't do much division**
 - 1000 per day
- ◆ **Many applications can tolerate occasional errors**
 - E.g., image generation
- ◆ **Maximum error fairly small**
 - Once every 1.5×10^9 divides
- ◆ **Statistical likelihood remote**
 - Once every 27,000 years

IBM's Counterclaims

- ◆ **Even spreadsheet users do lots of divides**
 - Up to 4.2 Million per day
- ◆ **Error more likely for numbers just below integer**
 - Up to once every 10^8 divides

Other Concerns

Error is not Random

- ◆ Get same result every time on every Pentium

Numbers Used in Practice NOT Uniformly Distributed

- ◆ E.g., $4.999999 / 14.999999$
 - gives 0.33332922 instead of 0.33333329
 - Vaughan Pratt's "bruised integers"

Error is Hard to Work Around

- ◆ Can shut off floating point
 - Machine then runs slower than 486
- ◆ Can recompile with workaround divide subroutine
 - But most users only have binaries

Bad Numerical Properties

- ◆ E.g, $4.999999 / 15.0 > 4.999999 / 14.999999$
 - Problem for iterative methods

Who is Affected?

Average PC User

- ◆ Not likely

Heavy Users of Numerical Software

- ◆ Significant subset of Pentium customers
- ◆ Concern for numerical accuracy very high
 - Single inaccurate result can be magnified by later steps
- ◆ Ethical/legal responsibilities
 - Negligent to knowingly use flawed system
 - Would you buy a bridge from a civil engineer who had used a Pentium?

Why Didn't Intel Discover it?

Standard Steps in Verifying Design

- ◆ **Simulate many cases on high-level software model**
 - Probably had correct P-D table entries
- ◆ **Simulate/emulate final logic design**
 - Hardware emulators costing \$Millions
 - Run complete chip model at ~ 100Hz
 - » $< 10^{-6} \times$ real time
 - Not clear which version of table was used
- ◆ **Run tests on initial production chips**
 - Feasible to run billions of tests
 - Should have caught error here

Observations

- ◆ **Hard to test all aspects of such a complex system**
 - But this is a weak excuse

What about Formal Verification?

Currently Used Tools

- ◆ Most based on Ordered Binary Decision Diagrams (OBDDs)

Other Alternatives

- ◆ Sequential verification by symbolic model checking
 - Used on protocols and small sequential circuits
- ◆ Tools based on automatic theorem provers
 - Unpopular with industry

What About Dividers?

- ◆ Some have been done with theorem prover, but messy
- ◆ BDD-based tools cannot handle complete algorithm
 - But still could be used to verify single iteration
- ◆ New tools based on word-level representations show promise
 - Binary Moment Diagrams

Specifying Single Step

Algebraic Representation

- ◆ Define

$$P_i = PS_i + PC_i$$

$$P_{i+1} = PS_{i+1} + PC_{i+1}$$

- ◆ Input Range Constraint

$$-8/3 D \leq P_i \leq 8/3 D$$

- ◆ I/O Relation

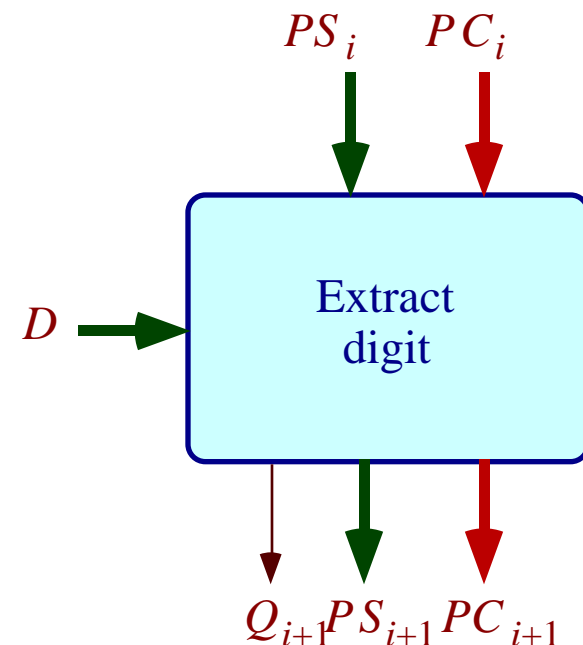
$$P_{i+1} = 4 [P_i - Q_{i+1} D]$$

- ◆ Output Range Constraint

$$-8/3 D \leq P_{i+1} \leq 8/3 D$$

Bit-Level Verification

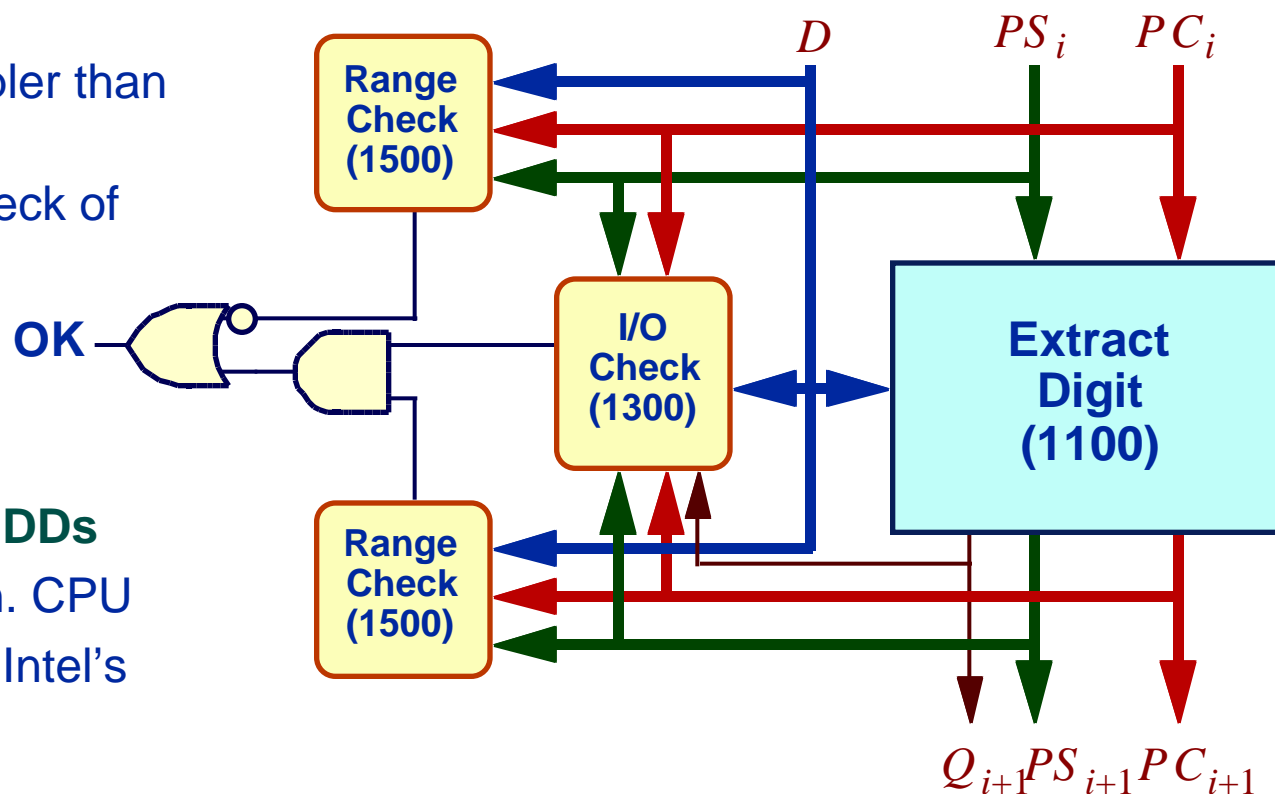
- ◆ BDDs represent Boolean functions
- ◆ Must give specification in terms of individual signal values



Bit-Level Verification of Single Step

◆ Create “checker circuit”

- Gate-level realization of spec.
- Larger, but simpler than actual circuit
- Independent check of design



◆ Evaluate with OBDDs

- 112 MB, 10 min. CPU
- Would uncover Intel’s problem

Bit-Level Analysis Summary

Shortcomings

- ◆ Must generate checker circuits
- ◆ No reliable way to verify checker circuits
- ◆ BDD blowup limits size/class of function

Benefits

- ◆ Tests complete functionality
- ◆ Designing checker circuits not so difficult
 - More conservative design style
 - Can have library of functions
- ◆ Can use to synthesize or optimize design
- ◆ Low cost
 - < \$200,000 test of divider would have saved \$475 million

Long Term Trends

Microprocessor Complexity Increasing at Rapid Pace

- ◆ Pentium has features historically found in high-end mainframes
 - Hardware support for multiplication, division, square root, and transcendental functions
 - Sophisticated instruction pipeline
- ◆ Exercising all possible system behaviors especially difficult
- ◆ Future generations even more complex
 - Speculative execution

Must Get it Right First Time

- ◆ Cannot send field service to replace boards
- ◆ Typically have more people working on verification than on design

Ubiquitous Systems of Unprecedented Complexity

- ◆ Industry's headaches caused by own successes

Impact on Intel

Short Term

- ◆ **Embarassing to make mistake**
- ◆ **Bad PR due to poor crisis management**
- ◆ **\$475 Million is a lot of money**
 - Still had very profitable year, including \$372 Million for 4Q94

Long Term

- ◆ **Intel still way ahead of competition**
 - Pentium class clones just coming out
 - Intel already shipping next generation PentiumPro
 - \$15 Billion of \$16 Billion x86 microprocessor market in 1996
- ◆ **Everyone will be more careful in the future**
 - Good opportunities for research on formal verification
- ◆ **Intel benefits even from this publicity**
 - as playing key role in computer industry

Impact on Society

Reminded of Importance of Computing

- ◆ Have come to expect ever faster & cheaper machines
- ◆ Rely heavily on correctness of results

The Power of the Internet

- ◆ Created *ad-hoc*, international group of collaborators
 - U.S., Germany, Norway, Canada
 - University, industry
 - No external control or authority
- ◆ Rapid dissemination of information and results
- ◆ Caught Intel by surprise
 - Had never dealt with force of this kind
- ◆ System does not scale
 - A few gems among the drivel