

ARITH 2025



El Paso, TX, USA. May 4-7, 2025

Experimental Software and Hardware Evaluation of Ad-Hoc Constant Division Routines

Frédéric Pétrot
Grenoble INP/Université Grenoble Alpes,
CNRS, TIMA Lab,
F-38000 Grenoble, France
✉ frederic.petrot@univ-grenoble-alpes.fr



Why divide by integer constants?

Initially Motivated by AI Computations

Average Pooling

Kernel shapes known beforehand for hw implementations

Kernel usually square $k \times k$, with k odd and "small"

Look for efficient $\frac{X}{k^2}$ implementation

Many algorithms need that!

Printing decimal numbers, filters in signal processing, ...

Lots of prior art, summarized in:

- ▶ Hacker's Delight, Warren, Chapter 10
- ▶ Application-Specific Arithmetic, De Dinechin & Kumm, Chapter 13
- ▶ **Focus of this talk: Ad-Hoc Cookbook proposed by Li in 1985**

General approach: multiply by the reciprocal

Compute $\frac{1}{p}$

$$\frac{1}{p} = 0.(0b_1b_2b_3 \dots b_{n-1}b_1b_2b_3 \dots b_{n-1})$$

$$\frac{x}{p} = \frac{x}{b_1 2^1} + \frac{x}{b_2 2^2} + \frac{x}{b_3 2^3} + \frac{x}{b_4 2^4} + \dots$$

Note that binary pattern is repeating ad infinitum:

Quickly for some numbers, not so much for others

$$\begin{aligned}\frac{1}{3} &= 0.0101010101010101010101010101010\dots = 0.(01)* \\ \frac{1}{23} &= 0.000010110010000101100100001011001\dots = 0.(00001011001001)* \\ \frac{1}{47} &= 0.000001010111001001100010000010101\dots = 0.(00000101011100100110001)* \\ \frac{1}{63} &= 0.000001000001000001000001000001000\dots = 0.(000001)*\end{aligned}$$

Use infinite product rather than infinite series

Finding the pattern and its period

Li's first proposal based on Fermat Euler's theorem

Assuming $p > 1$ odd, find smallest n such that p divides $2^n - 1$

$$\frac{2^n - 1}{p} = b_1 b_2 b_3 \dots b_{n-1}$$

$$\frac{1}{p} = 0.(0b_1 b_2 b_3 \dots b_{n-1}) \prod_{i=0}^{\infty} \left(1 + \frac{1}{2^{n \times 2^i}}\right)$$

Example: $\frac{x}{23}$

Smallest integer n such that 23 divides $2^n - 1$ is 11

$$\frac{2047}{23} = 89 \text{ or } 1011001_2$$

Must be written on 11 bits to have the proper magnitude 00001011001_2

$$\frac{x}{23} = \left(\frac{x}{2^5} + \frac{x}{2^7} + \frac{x}{2^8} + \frac{x}{2^{11}}\right) \left(1 + \frac{1}{2^{11}}\right) \left(1 + \frac{1}{2^{22}}\right) \dots$$

Computing the division

Continuing with $\frac{x}{23}$

```
uint32_t bad_divu23(uint32_t x)
{
    /* needs 6 shifts and 5 adds vs 12 and 11 using raw decomposition */
    x = (x >> 5) + (x >> 7) + (x >> 8) + (x >> 11);
    x = (x >> 11) + x;
    x = (x >> 22) + x;
    return x;
}
```

Incorrect, rounding is very soon an issue

Usual approach computes remainder and corrects:

```
uint32_t good_divu23(uint32_t n)
{
    /* but also mult and test to be correct! */
    uint32_t x = n, r;
    x = (x >> 1) + (x >> 3) + (x >> 4) + (x >> 7);
    x = (x >> 11) + x;
    x = (x >> 22) + x;
    x = x >> 4;
    r = n - x * 23;
    return x + (r > 22);
}
```

Li's Routines Sample

Handmade routines for unsigned division

Playing with the binary decomposition, its complement, and experimenting

Divisor	Division Procedure		
3	X - XL2+X+5 X - XR16+X	X - XR4+X X - XR4	X - XR8+X
5	X - XL1+X+3 X - XR16+X	X - XR4+X X - XR4	X - XR8+X
7	X - X+1 X - XR24+X	X - XL2+XR1 X - XR5	X - XR6+X
9	X - X+1 X - XR12+X	X - XL1+X+XR1 X - XR24+X	X - XR6+X X - XR5
11	X - X+1 X - XR10+X	Y - XL2+X X - XR20+X	X - Y+YR4+XR1 X - XR6
13	X - X+1 X - XR12+X	X - XL2+XR1 X - XR24+X	X - X+(XR1+X)R4 X - XR6
15	X - X+1 X - XR16+X	X - XL2+XR2 X - XR6	X - XR8+X
17	X - XL2+X+5 X - XR16+X	X - XR1+X X - XR7	X - XR8+X
19	X - XL1+X+2 X - XR18+X	X - XR3+X X - XR6	X - X-XR9
21	X - XL1+X+3 X - XR24+X	X - XR6+X X - XR6	X - XR12+X
23	X - X+1 X - XR22+X	X - (XR3+X)R1+X+XL2 X - XR7	X - XR11+X

```
uint32_t <ug?>li_divu23(uint32_t n)
{
    /* 6 shifts and 6 adds, magic ! */
    uint32_t x = n + 1;
    x = (((x >> 3) + x) >> 1) + x + (x << 2);
    x = (x >> 11) + x;
    x = (x >> 22) + x;
    x = (x >> 7);
    return x;
}
```

Quite mind blowing!

But there is not free lunch!

Erroneous from 0x2e000000 on

Li's Routines Sample

Handmade routines for unsigned division

Playing with the binary decomposition, its complement, and experimenting

Divisor	Division Procedure		
3	X - XL2+X+5 X - XR16+X	X - XR4+X X - XR4	X - XR8+X
5	X - XL1+X+3 X - XR16+X	X - XR4+X X - XR4	X - XR8+X
7	X - X+1 X - XR24+X	X - XL2+XR1 X - XR5	X - XR6+X
9	X - X+1 X - XR12+X	X - XL1+X+XR1 X - XR24+X	X - XR6+X X - XR5
11	X - X+1 X - XR10+X	Y - XL2+X X - XR20+X	X - Y+YR4+XR1 X - XR6
13	X - X+1 X - XR12+X	X - XL2+XR1 X - XR24+X	X - X+(XR1+X)R4 X - XR6
15	X - X+1 X - XR16+X	X - XL2+XR2 X - XR6	X - XR8+X
17	X - XL2+X+5 X - XR16+X	X - XR1+X X - XR7	X - XR8+X
19	X - XL1+X+2 X - XR18+X	X - XR3+X X - XR6	X - X-XR9
21	X - XL1+X+3 X - XR24+X	X - XR6+X X - XR6	X - XR12+X
23	X - X+1 X - XR22+X	X - (XR3+X)R1+X+XL2 X - XR7	X - XR11+X

```
template<unsigned int S = 32>
sc_uint<S> divu23(sc_uint<S> n)
{
    sc_uint<S + 3> x = n + 1;
    x = (((x >> 3) + x) >> 1) + x + (x << 2);
    x = (x >> 11) + x;
    x = (x >> 22) + x;
    x = (x >> 7);
    return x;
}
```

Works for all 32-bit values,
at the cost of "just" 3 more bits
⇒ Expensive in software,
⇒ Cheap in hardware

Li's Routines

Unsigned division for odd numbers between 3 and 55 included

Not all are correct

5 erroneous routines, among which:

- ▶ 3 transcription errors: 7, 27, 39,
- ▶ 2 seemingly just wrong: 49, 53

```
static inline uint32_t divu49(uint32_t n)
{ /* 1 more shift than Li's */
    uint32_t x = n;
    x = (x << 2) - (x >> 5) + 2;
    x = x + (x >> 2) + (((x >> 4) + x) >> 4);
    x = (x >> 21) + x;
    x = x >> 8;
    return x;
}
```

```
static inline uint32_t divu53(uint32_t n)
{ /* 2 more shifts and adds than Li's */
    uint32_t x = n + 1, y;
    y = (x << 1) + (x >> 2);
    y = x + y + (y >> 5);
    y = y + ((x - (x >> 3)) >> 11)
        + ((x - (x >> 5)) >> 16)
        + (x >> 23) + (x >> 24);
    x = (x << 2) + (y >> 2);
    x = (x >> 8);
    return x;
}
```


Granlund & Montgomery

Variations used by gcc and clang

Get away with a mult by a magic constant followed by a right shift

```
movl    $2987803337, %edx
imulq   %rdx, %rax
shrq    $36, %rax
```

Table: Number of x86_64 instructions for SA and MH divisions (SA/MH)

divisor	3	5	7	9	11	13	15	17	19
nb insns	10/3	10/3	13/6	13/2	12/3	15/2	10/3	10/3	10/7
divisor	21	23	25	27	29	31	33	35	37
nb insns	10/7	13/3	12/2	9/6	13/3	10/6	12/2	13/7	11/7
divisor	39	41	43	45	47	49	51	53	55
nb insns	13/7	10/3	10/2	12/6	17/3	14/2	7/3	27/6	16/6

Granlund & Montgomery

Variations used by gcc and clang

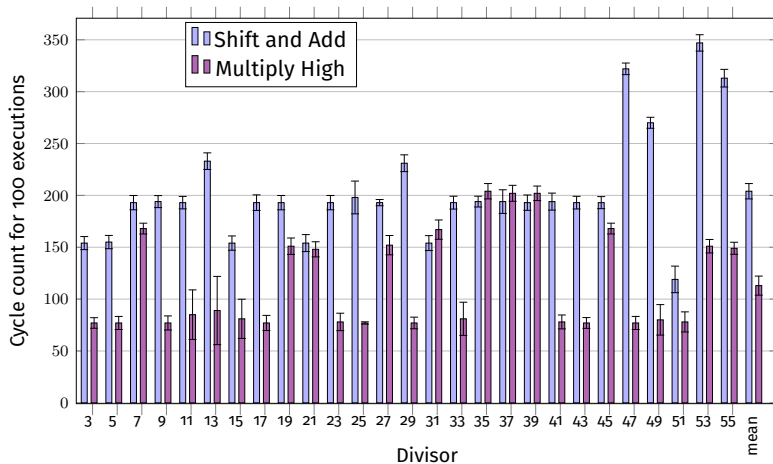
Get away with a mult by a magic constant followed by a right shift

```
movl    $2987803337, %edx
imulq   %rdx, %rax
shrq    $36, %rax
```

Table: Number of x86_64 instructions for SA and MH divisions (SA/MH)

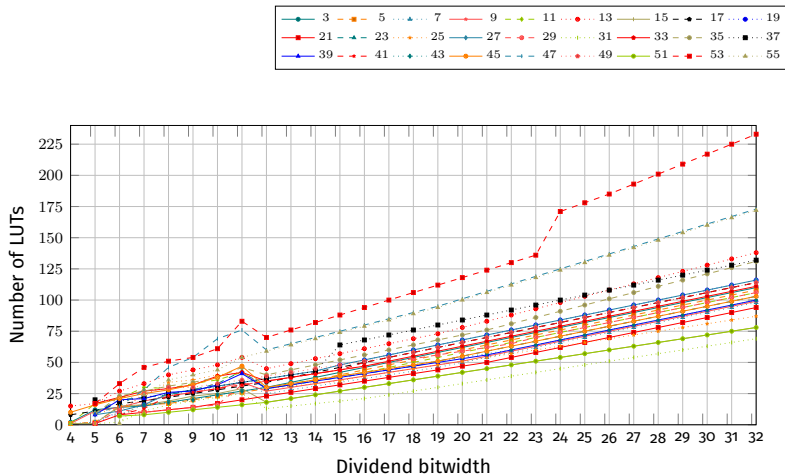
divisor	3	5	7	9	11	13	15	17	19
nb insns	10/3	10/3	13/6	13/2	12/3	15/2	10/3	10/3	10/7
divisor	21	23	25	27	29	31	33	35	37
nb insns	10/7	13/3	12/2	9/6	13/3	10/6	12/2	13/7	11/7
divisor	39	41	43	45	47	49	51	53	55
nb insns	13/7	10/3	10/2	12/6	17/3	14/2	7/3	27/6	16/6

Software evaluation



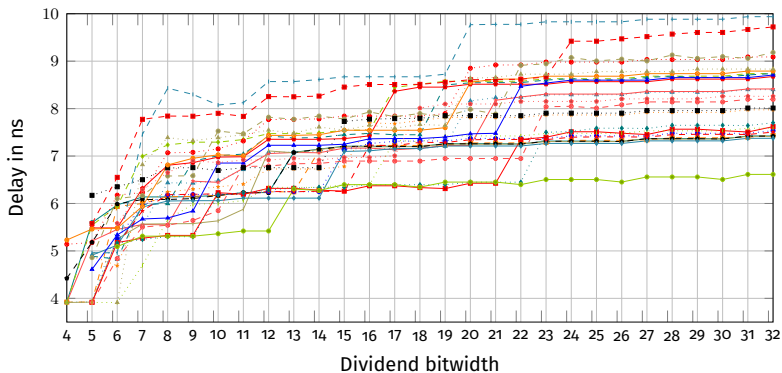
Run times of Shift-and-Add and Multiply High, 32-bit dividend
⇒ *SA useful only on processors without Multiply-High, e.g. Cortex-M0/M1*

Hardware Evaluation : Li's Resources on FPGA



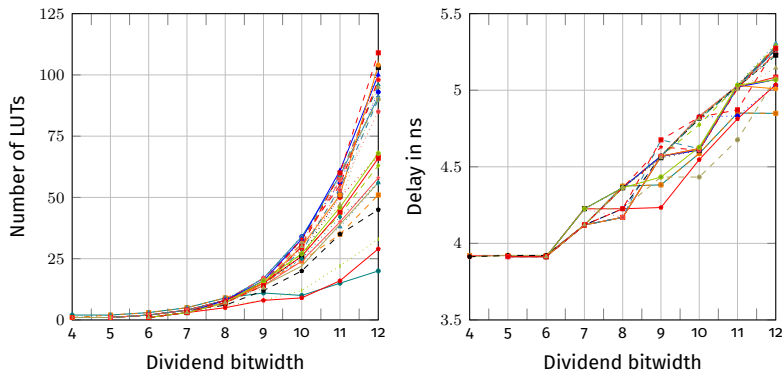
Li: Number of LUTs on Virtex 7.

Hardware Evaluation : Li's Timings on FPGA



Li: Circuit delay in ns on Virtex 7.

Hardware Evaluation : Brute-Force Approach on FPGA



Brute-force: Number of LUTs and delay on Virtex 7.

Hardware Evaluation : Comparison with SoA on FPGA

Delay and area after FPGA synthesis (BR: Block Ram)

d	n	This work		SC (Arith 2023)		LinArch (TC 2017)		BTCD (TC 2017)	
		Delay ns	Area LUTs	Delay ns	Area LUTs	Delay ns	Area LUTs	Delay ns	Area LUTs + BR
3	16	7.2	46	4.1	40	3.6	17	3.7	37
	32	8.6	114	11.4	98	6.0	32	4.8	95
	64	9.0	277	27.5	379	13.5	63	6.2	225
5	16	7.2	44	4.0	52	4.4	21	3.8	44
	32	8.6	111	10.6	123	9.3	45	4.7	109
	64	9.0	274	27.3	386	20.1	93	6.7	270
11	16	7.5	42	3.9	53	8.0	39	3.8	79
	32	8.7	106	10.7	159	17.9	87	6.1	212
	64	10.4	265	26.9	436	39.0	183	8.8	526
23	16	7.5	41	3.9	52	7.4	69	5.6	197
	32	8.7	103	10.4	187	18.5	165	6.8	436 + 1BR
	64	10.4	256	26.1	493	36.6	357	6.5	959 + 2.5BR

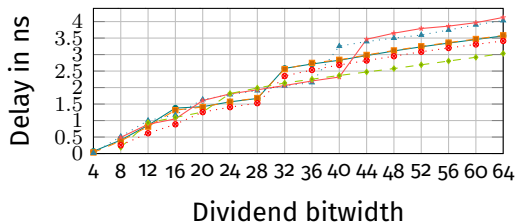
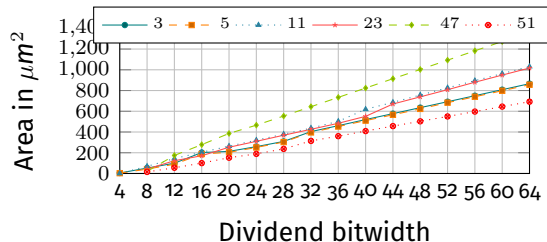
Hardware Evaluation : Comparison with SoA on FPGA

Delay and area after FPGA synthesis (BR: Block Ram)

d n	This work		SC (Arith 2023)		LinArch (TC 2017)		BTCD (TC 2017)	
	Delay ns	Area LUTs	Delay ns	Area LUTs	Delay ns	Area LUTs	Delay ns	Area LUTs + BR
3	16	7.2 46	4.1 40	3.6 17	3.7	37		
	32	8.6 114	11.4 98	6.0 32	4.8	95		
	64	9.0 277	27.5 379	13.5 63	6.2	225		
5	16	7.2 44	4.0 52	4.4 21	3.8	44		
	32	8.6 111	10.6 123	9.3 45	4.7	109		
	64	9.0 274	27.3 386	20.1 93	6.7	270		
11	16	7.5 42	3.9 53	8.0 39	3.8	79		
	32	8.7 106	10.7 159	17.9 87	6.1	212		
	64	10.4 265	26.9 436	39.0 183	8.8	526		
23	16	7.5 41	3.9 52	7.4 69	5.6	197		
	32	8.7 103	10.4 187	18.5 165	6.8	436 + 1BR		
	64	10.4 256	26.1 493	36.6 357	6.5	959 + 2.5BR		

Hardware Evaluation: ASIC

STMicroelectronics 28 nm FDSOI technology



Comparison with SOA (TC 2017)

- ▶ Slower than SOA for small to medium bit sizes, but SOA slope steeper
Easily pipelineable for high-throughput
- ▶ Area about half the size of SOA solutions

Thorough validation and evaluation of Li's Constant Division routines

Validation:

- ▶ Brute force validation for 32 bits quick enough on today's computers
- ▶ Found 5 erroneous routines: 3 typos, 2 wrong
- ▶ Two corrected routines slightly more complex but correct on 32 bits using 35

Evaluation:

- ▶ Software:
Algorithms using multiply-high faster
⇒ Routines useful only on (very) low-end processors
- ▶ Hardware:
Provides a different trade-off compared to existing approaches
Very dependent on the value of the divisor and the target technology
⇒ Can be useful on a case-by-case basis

Take away

Li's approach hard to scale

- ▶ Erroneous above "some" point
Not far above 32 for quite a few, much farther for others
- ▶ Reason for that and mitigation yet unclear

Li's approach hard to generalize

Special case $\frac{n}{R^2}$ doesn't seem so special
No nice formulation found for division with SA
Not even some kind of algorithm
⇒ Trial and error approach not satisfactory

Code, of the PoC kind, ...

<https://github.com/fpetrot/divbysmallcst>

Thanks for listening

Thanks to:

- ▶ MIAI Institute
- ▶ ANR that is financing the Holigrail project
- ▶ EU and French DGE that is financing the EdgeAI project

Thanks for listening

Thanks to:

- ▶ MIAI Institute
- ▶ ANR that is financing the Holigrail project
- ▶ EU and French DGE that is financing the EdgeAI project

And don't forget to go to Arith one of these days, ...

