Designing and testing an ALU for multiprecision arithmetic

Dorian Lesbre

September 10th, 2021

Outline

1. Introduction
2. Hardware design tools
3. The ALU
4. Compiling to Verilog

Introduction

The Cephalopod processor

Designed to limit main sources of error in IoT :

- Garbage collection errors
- Integer overflow

ALU Goals

- Avoid overflow by using multiprecision integers
- Be efficient on single chunk integers

The ALU

Integer representation

- linked list in RAM
- signed by topmost chunk
- remove leading sign chunks

The ALU

ALU components

- Logical unit (&&, ||, $\neg$, `if-then-else`)
- Comparator unit ($==$, $!=$, $>$, $>=$)
- Arithmetic unit ($+$, $-$, $\times$, $/$, %, $\sqrt{\ }$)
- Cleaning unit

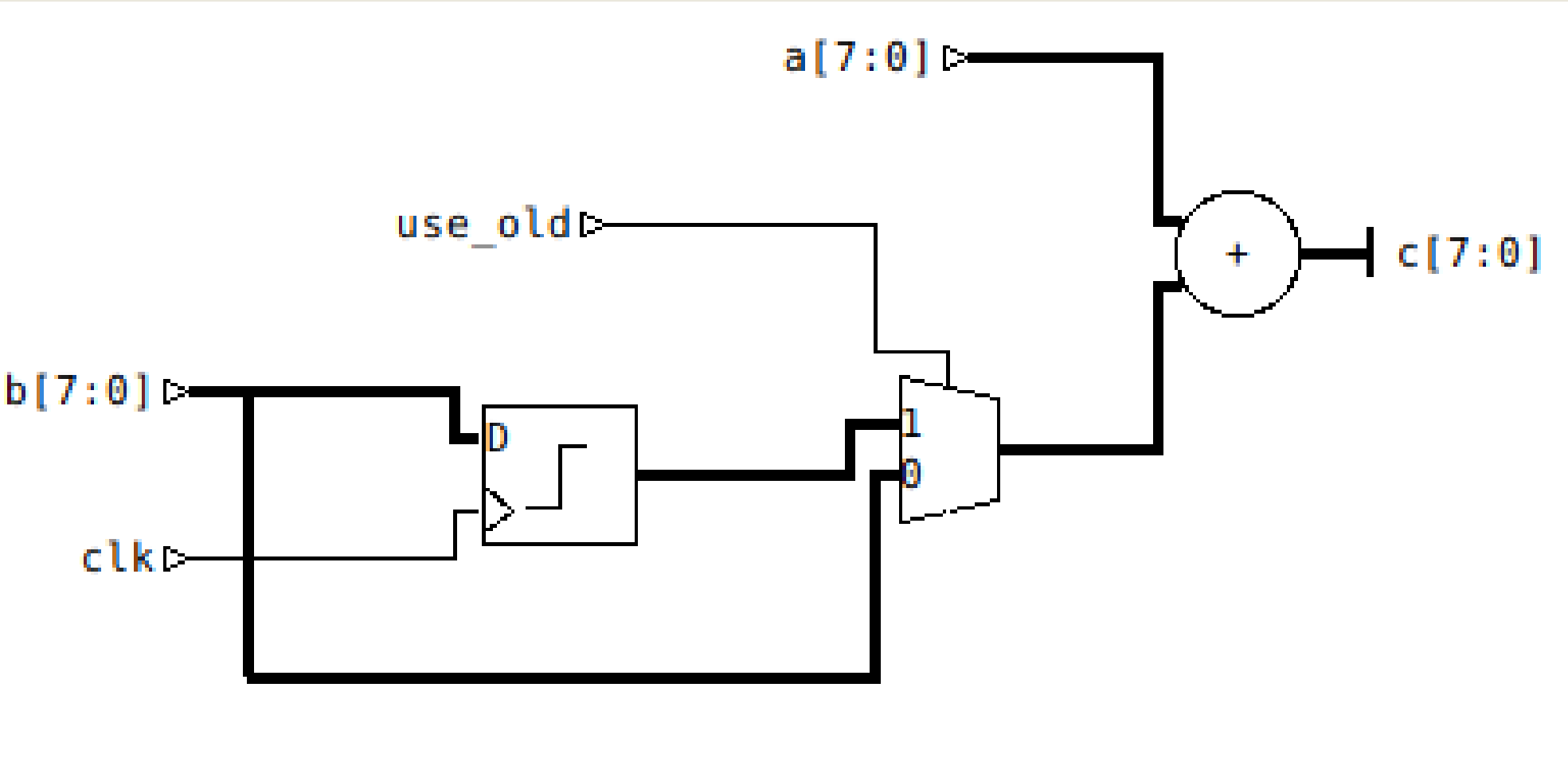
Hardware design tools

Voss II : hardware description in HFL

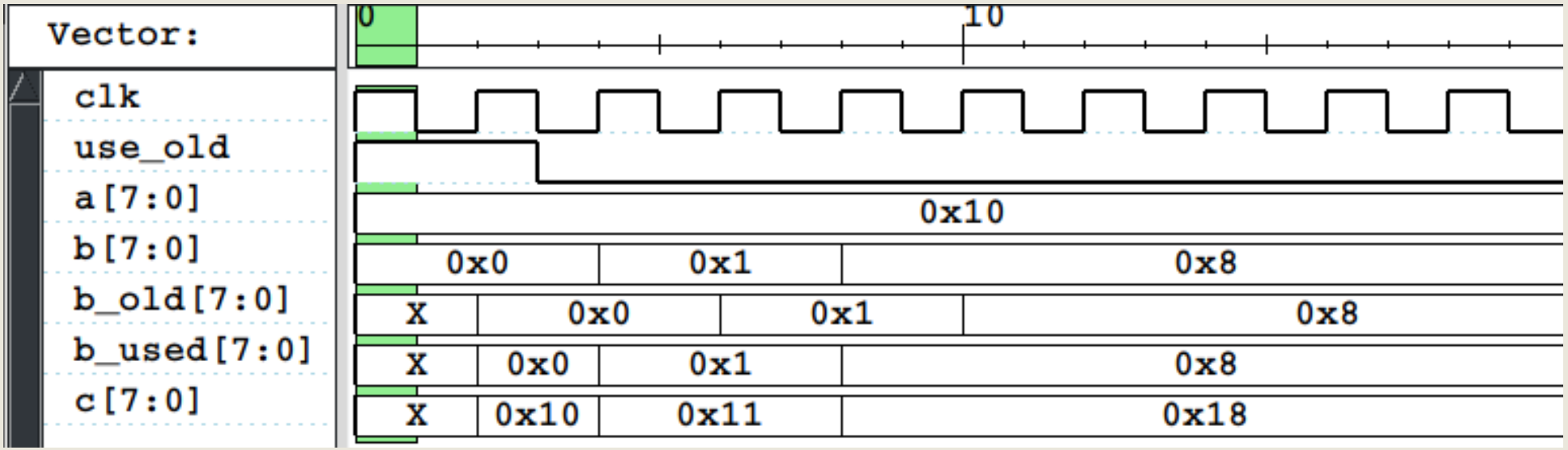
```
TYPE "my_int" 8;

let example =
  bit_input clk use_old.
  my_int_input a b.
  my_int_output c.
  my_int_internal b_old b_used.
  CELL "example" [
    re_ff clk b b_old,
    b_used <- (IF use_old THEN b_old ELSE b),
    c <- a '+' b_used
  ];
```

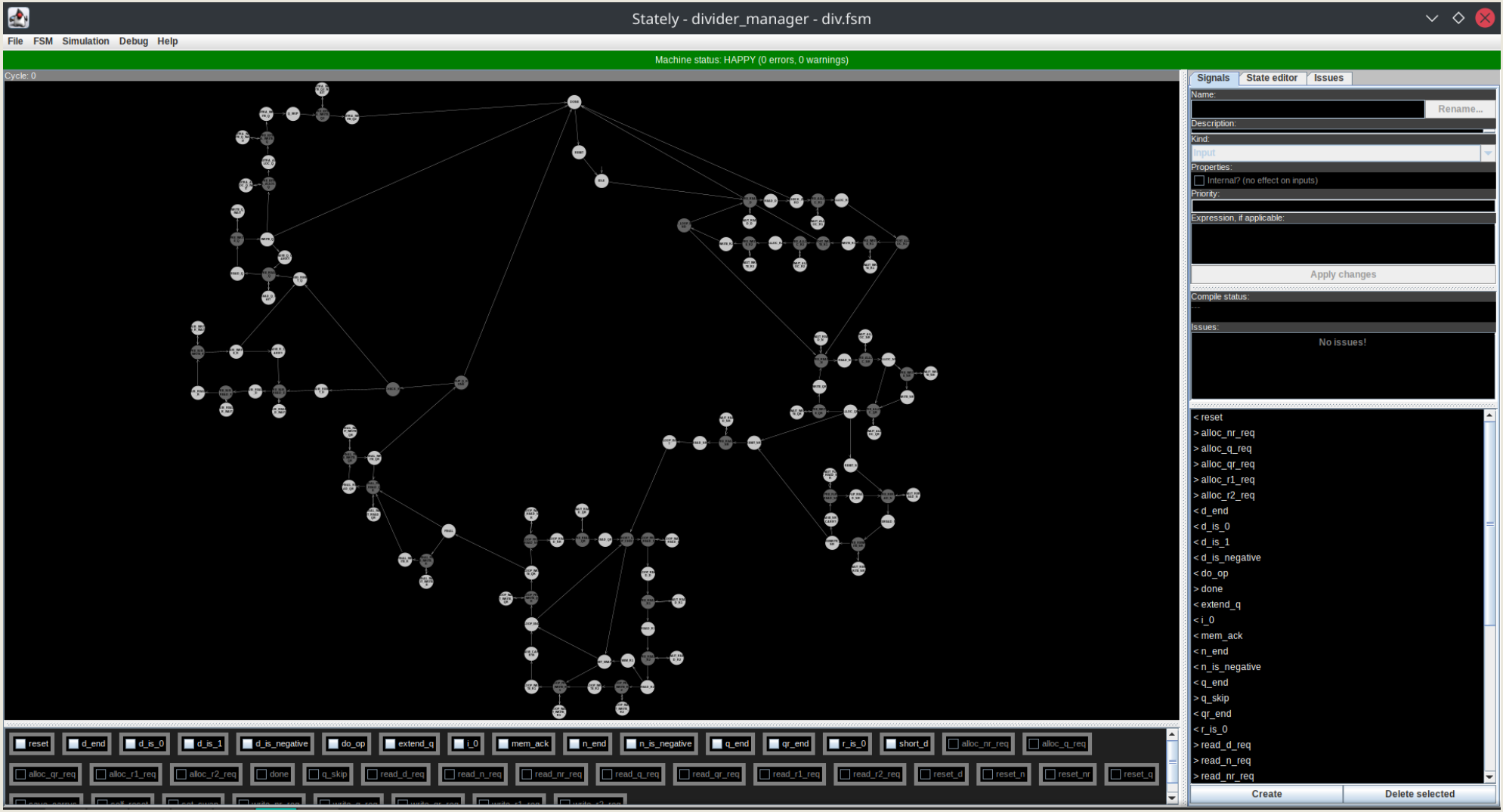
Voss II : circuit visualization



Voss II : circuit simulation



Stately : an FSM editor



Bifrost : a higher level language

```
read_d:
  node_d = do mem_read(addr_d_cur);
  t_d_end = node_d.laddr == ADDR_END;
  short_d = t_d_end & addr_d_cur == addr_d;
  d_is_negative = t_d_end & get_bit(node_d.int, TOP_BIT_POS);
  if (node_d.int == 0 & short_d) {
    goto zero_division;
  }
  addr_r1_cur = do mem_alloc();
  if (node_d.int != 1 & short_d) {
    // for short d only allocate one R
    node_d.int = if d_is_negative then (~node_d.int+1) else node_d.int;
    addr_r1 = addr_r1_cur;
    goto read_n;
  }
```

The ALU

The multiplier

- primary school multiplication
- uses a single cummulative sum
- Sign extend topmost bits

Unsigned division

Definition

For two integers N, $D \in \mathbb{N}^2$ with $D \neq 0$, find Q, $R \in \mathbb{N}^2$ such that :

$$\begin{cases} N = QD + R \\ 0 \leqslant R < D \end{cases}$$

Long division

74	6
-6	12
14	
-12	
2	

Algorithm : While $N \geqslant Q$

1. Find a such that $10^a D \leqslant N < 10^{a+1} D$
2. Find the largest $b \in [\![1,\ 9]\!]$ such that $10^a bD \leqslant N$
3. Set b as the a-th digit of Q
4. subtract $10^a bD$ from N
5. repeat

Binary long division

```
  1010 | 11
-------|----
 -0    | 0011
  1010 |
  -0   |
  1010 |
  -11  |
   100 |
   -11 |
     1 |
```

Algorithm :

Set $Q = 0$ and $R = 0$.

For every bit i in N descending

1. set $R = R \ll 1 + N[i]$
2. if $R > D$
   set $R = R - D$
   set $Q[i] = 1$

Mutliprecision long division

- Descending through N and Q easy if big-endian.
- Naive implementation leads to :
  - 3-5 reads and 1-2 writes every iteration
  - One extra chunk used for R as R can be greater then D.
- Can be slightly speed up by combining operations :
  - 3 reads and 2 writes every iteration (-2 chunks)
  - Needs to store 2 copies of R

Mutliprecision long division

R_1	010	001	111	(143)
D	011	101	011	(235)
$R_1 \ll 1 + 0$			110	$(c_1 = 1)$
$R_1 \ll 1 + 0 - D$			011	$(c_2 = 0)$
$R_1 \ll 1 + 0$		011		$(c_1 = 0)$
$R_1 \ll 1 + 0 - D$		110		$(c_2 = 1)$
$R_1 \ll 1 + 0$	100			$(c_1 = 0)$
$R_1 \ll 1 + 0 - D$	000			$(c_2 = 0)$
R'_1	100	011	110	(286)
R'_2	000	110	100	(52)

Mutliprecision long division

Algorithm : for every bit i of N descending

1. set $c_1 = N[i]$ and $c_2 = 0$
2. for every chunk j of D, R_1 and R_2
   - 2.1 set $T = R_1[j] \ll 1 + c_1$
   - 2.2 write $T[2:0]$ in $R_1[j]$
   - 2.3 set c_1 to $T[3]$
   - 2.4 write $(T - D[j])[2:0]$ in $R_2[j]$
   - 2.5 set c_2 to $(T - D)[3]$
3. if c_1 or not c_2 swap R_1 and R_2

Signed division

- Flip the sign of N or D to be positive
- Some extra operations needed on R and Q

Testing the ALU

HFL allows symbolic evaluation of circuits :

- Can test on fixed number of chunks with variables as integer
- Can add constraints to variables
- Tested up to 2-3 chunks of 6 bits.

Compiling to Verilog

Compiling to Verilog

Goal

Convert HFL's circuit representation (pexlif) to standard verilog

- Mostly straigtforward
- Recognize flip-flop to avoid phase delays
- Some fiddling with slices
- Some inlining to simplify generated code

Thank you for your attention !