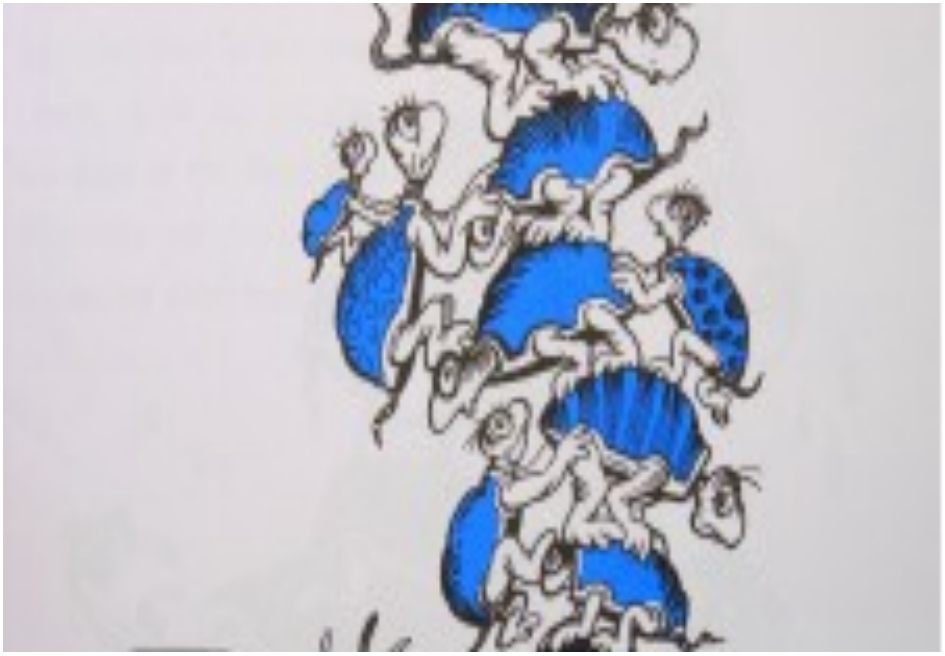
Turtles all the Way Down:

Running Linux on Open Hardware



Q: What parts of Linux Systems are Open and Under your Control?

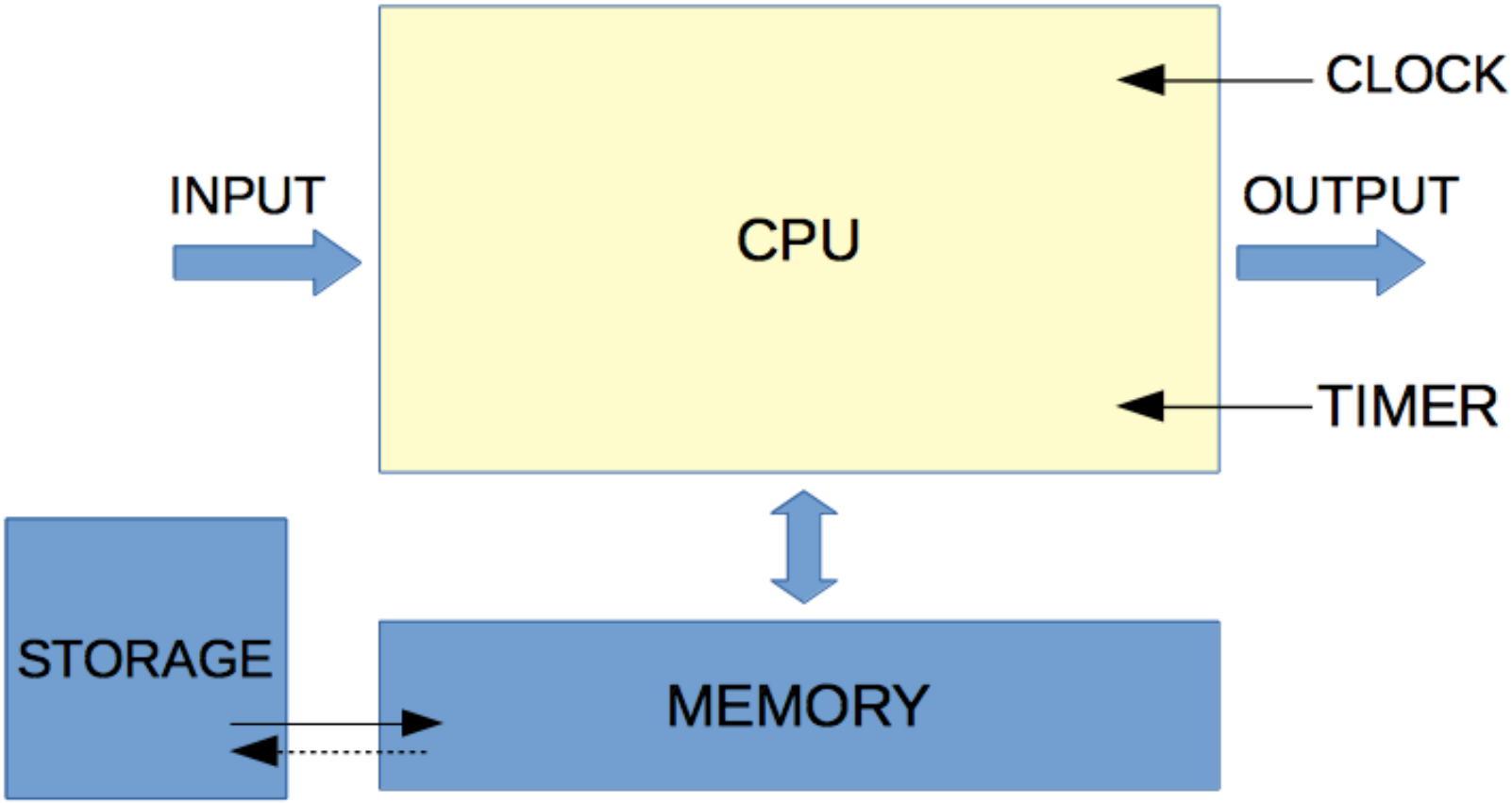
- Modern laptops have more arm/mips processors than x86
  - USB controller? Exploitable:

    http://arstechnica.com/security/2014/07/this-thumbdrive-hacks-computers-badusb-exploit-makes-devices-turn-evil/
  - Hard drive controller? Exploitable:

    http://hackaday.com/2013/08/02/sprite_tm-ohm2013-talk-hacking-hard-drive-controller-chips/
- System management mode, ACPI…

Taking Back Control of HW and SW

What is the minimum system that can run Linux?



The definition of a ‘CPU’ (for the purposes of this talk)

- Has a Flat 32bit memory space
  - All pointers ‘just work’ (no separate spaces)
- A port of GCC
  - Regular, efficient 32-16-8 bit int-short-char
- Executes instructions ‘fast enough’
  - Servicing streaming Ethernet traffic: about 30MIPs

System Requirements

- CPU (of course)
- Memory
  - 8 megs RAM runs practical Linux workloads
  - less is possible, but awkward
- Storage (load kernel+initramfs into memory)
- Some form of I/O
  - Usually, and at first, just a UART
- A Timer (interrupt).

Things you don’t need

- Anything else, actually
  - Video, audio, keyboard, persistent storage
- An MMU, or any ‘fancy’ CPU features.
  - FPU, SMP, even cache is optional

A History Lesson

The Linux/Microcontroller project is a port of Linux to systems without a Memory Management Unit (MMU).

uClinux first ported to the Motorola MC68328: DragonBall Integrated Microprocessor. The first target system to successfully boot is the PalmPilot using a TRG SuperPilot Board with a custom boot-loader created specifically for our Linux/PalmPilot port.

(January 1998)

Why recreate existing architecture?

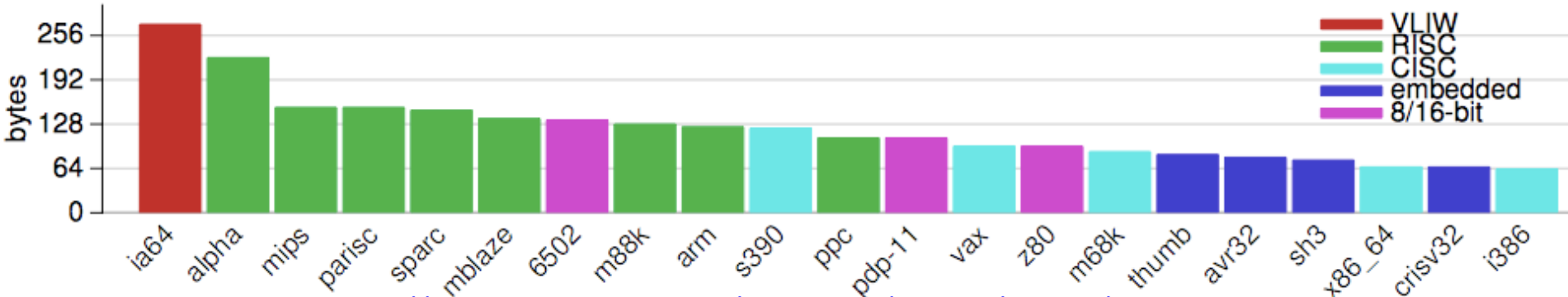
- Existing compiler, kernel, debugger, strace...
- Leverage massive R&D outlay in SuperH
    - 5 stage RISC (full Harvard architecture)
    - instruction set density (16 bit fixed length)
    - simple highly optimized design
    - original designers/implementers still around
- Old chips are prior art vs. "breathing is patented”

Patent Expiration

- SuperH ISA had a huge effort put into it
  - See above about efficient GCC.
- The SuperH architecture was developed by Hitachi a quarter century ago.
- Last patent on SH2 (Sega Saturn) expired in October 2014.
  - That's why we can release this now
  - last SH4 (Dreamcast) patent expires in 2016.
- SuperH ISA was the blueprint for ARM Thumb

Code Density : Efficiency

Fig. 2. Total size of benchmarks (includes some platform-specific code, so does not strictly reflect code density)



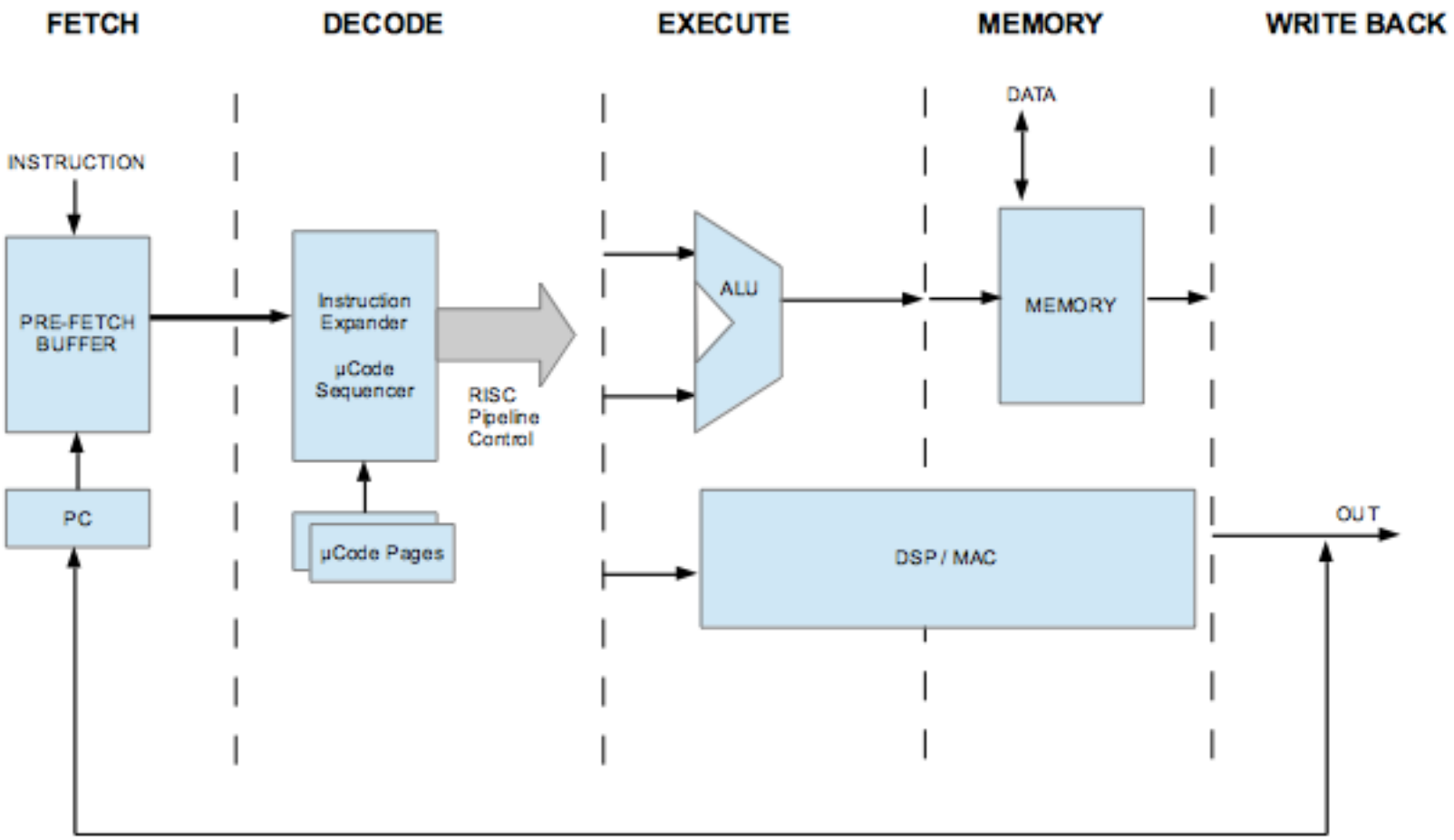
Source: http://web.eece.maine.edu/~vweaver/papers/iccd09/iccd09_density.pdf

- There *Are* other metrics, bit none actually matter more (unless something is broken)

  - Note: ARM paid millions to Hitachi to use SuperH patents in Thumb instruction set… which just expired.

The basic SuperH design:

5 stage ‘classical’ RISC pipeline, with some additions

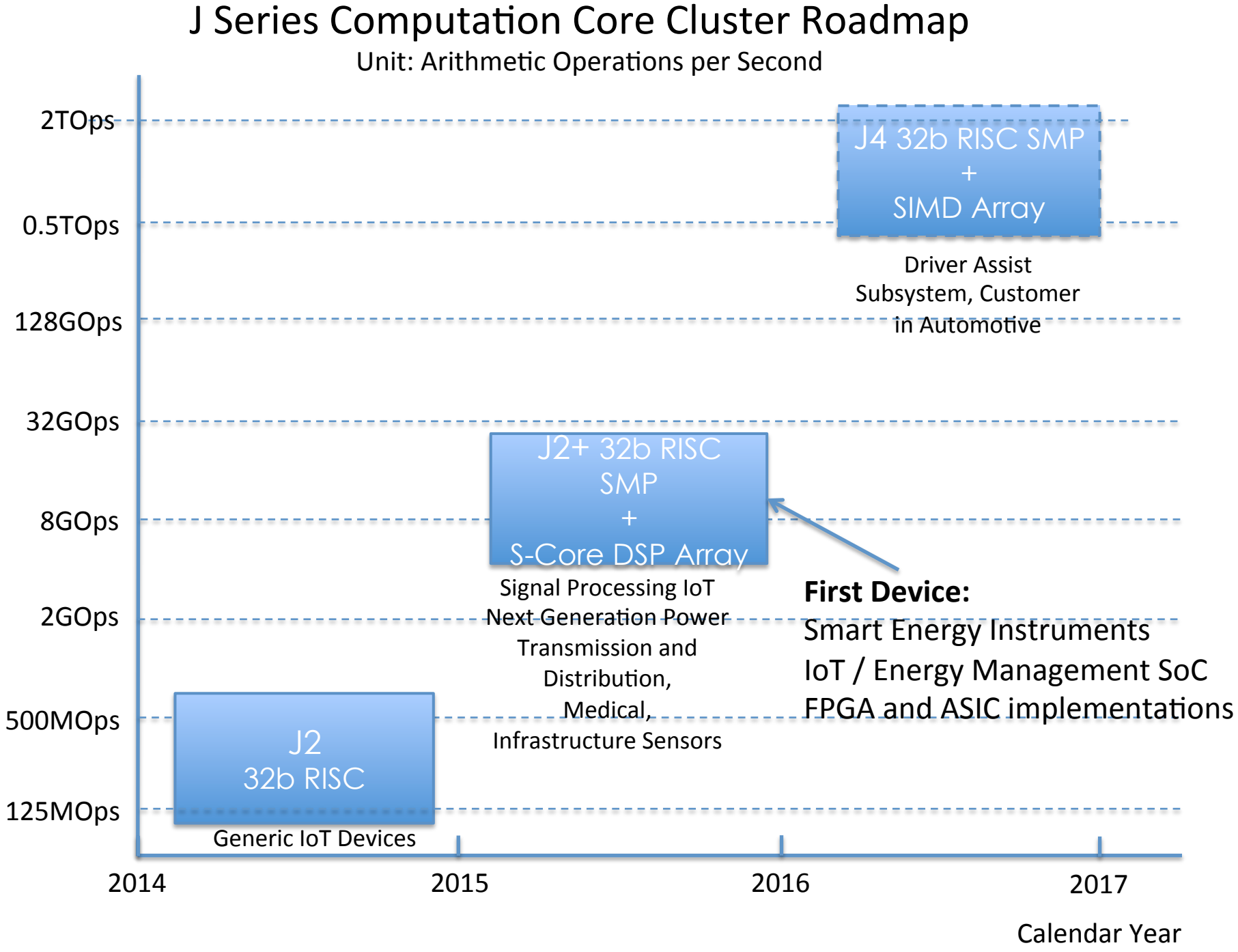


Pretty simple to implement, except the Instruction Decoder / Expander

What did we do?

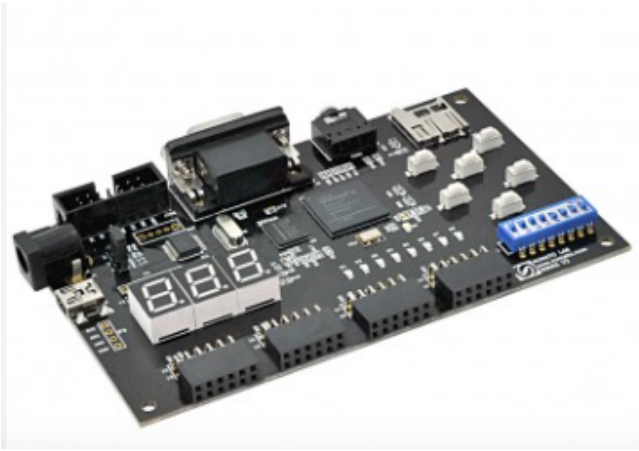
- New SH2 instruction set compatible core design kit
  - called "j2" because trademarks haven't expired.
  - clean-room implementation, initially by Canadian engineers
    - built for design reuse
    - Then hired SuperH architects to work on it afterwards
  - Source is VHDL using programming model developed by European Space Agency
    - Verilog is low level like assembly, VHDL is a HLL
    - Design Kit contains high level abstractions for future cores

- SOC builder system
  - Links together peripherals automatically
    - E.g. Serial, mmc, Ethernet
    - Can produce FPGA bitstream, ASIC RTL, C emulator source
    - SMP capable but not finished yet


J Series Computation Core Cluster Roadmap
Unit: Arithmetic Operations per Second
2TOps
0.5TOps
128GOps
32GOps
8GOps
2GOps
500MOps
125MOps
J4 32b RISC SMP + SIMD Array
Driver Assist Subsystem, Customer in Automotive
J2+ 32b RISC SMP + S-Core DSP Array
Signal Processing IoT Next Generation Power Transmission and Distribution, Medical, Infrastructure Sensors
First Device:
Smart Energy Instruments
IoT / Energy Management SoC
FPGA and ASIC implementations
J2 32b RISC
Generic IoT Devices
2014
2015
2016
2017
Calendar Year

Demo Platforms

- Our j2 processor core can run linux on low-cost FPGA Spartan6 platforms such as Numato Labs' mimas2, or Avnet's Lx9 microboard



- We are currently working to develop a custom development board with the same form factor as Raspberry Pi

So, how do you use it for anything?

- Releasing VHDL and build system under BSD license
- Make any chip you want
  - Royalty free
    - 180nm ASIC of SOC we're demoing costs <10 yen
      - Processor only, about 2 and a half cents
    - Disposable computing at "free toy inside" level
    - Think IoT : ‘Trillion Sensor Network’ economics, but running Linux
- nommu.org (uclinux-ng), 0pf.org (zero-p-f)
  - Source, documentation, tutorials, mailing lists
    - We assume you've never done hardware before.
  - Still a bit sparse but filling sites out as we go
    - Patches welcome. No question too stupid.

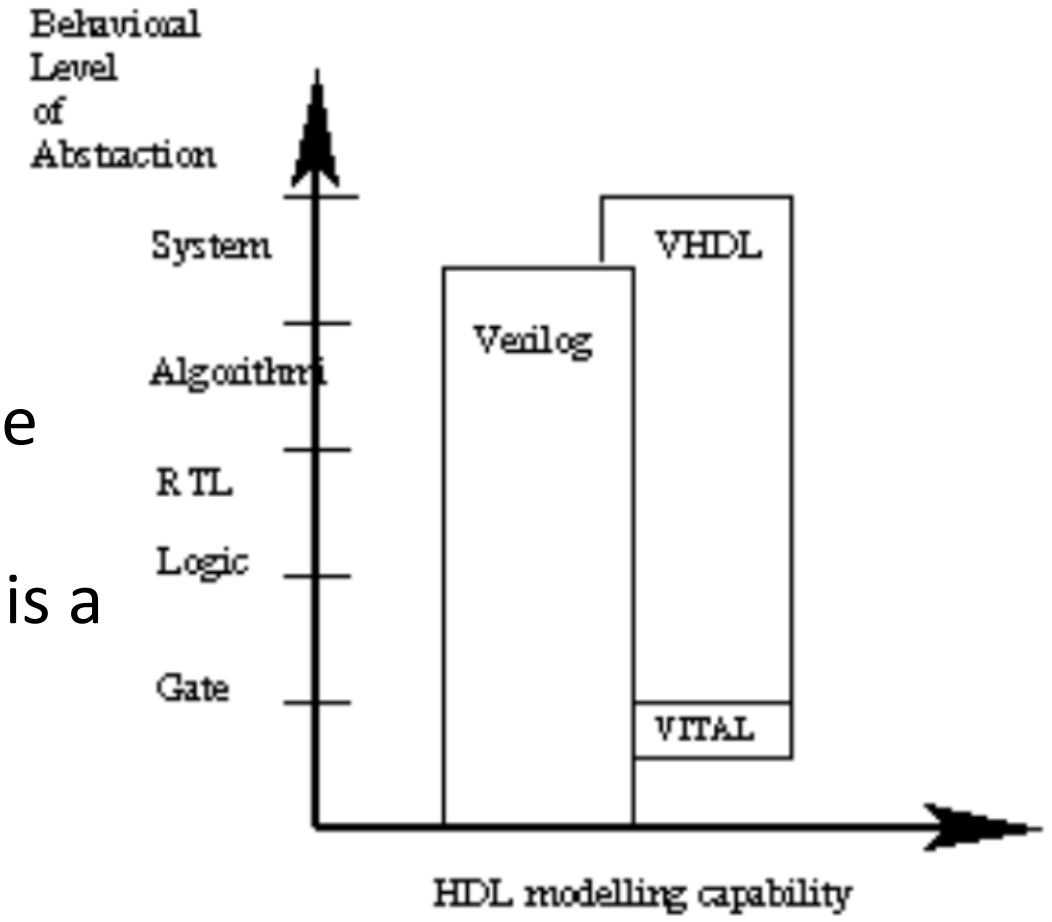
How: VHDL, not Verilog

- Although Verilog is more commonly used in certain geographies..
- VHDL is the preferred language in developments initiated or led by the European Space Agency.
- The VHSIC Hardware Description Language (VHDL) is a formal notation intended for use in all phases of the creation of electronic systems. Because it is both machine readable and human readable, it supports the development, verification, synthesis, and testing of hardware designs, the communication of hardware design data, and the maintenance, modification, and procurement of hardware.

These 2 things are not the same…

- Actually…

- It's about the type system.
- Verilog don't have one (to speak of)
- In VHDL, everything* is a derived type.
- Even + is 'just' and overloaded operator.



Example code : Grey Vector Type

…

```
package gray_pack is

type gray_vector is array (natural range <>) of std_logic;

function "+" (L: gray_vector; R: integer) return gray_vector;

…

function "+" (L: gray_vector; R: integer) return gray_vector is
variable res : gray_vector(L'range);
begin
  res := gray_vector(gr_inc(std_logic_vector(L)));
  return res;
end "+";
```

Example code : CPU Top Level

```
…

 type cpu_instruction_i_t is record
     d    : std_logic_vector(15 downto 0);
     ack  : std_logic;
   end record;
…
 component cpu is port (
     clk              : in  std_logic;
     rst              : in  std_logic;
     db_o             : out cpu_data_o_t;
     db_i             : in  cpu_data_i_t;
     inst_o           : out cpu_instruction_o_t;
     inst_i           : in  cpu_instruction_i_t;
     debug_o          : out cpu_debug_o_t;
     debug_i          : in  cpu_debug_i_t;
     event_o          : out cpu_event_o_t;
     event_i          : in  cpu_event_i_t);
   end component cpu;
```

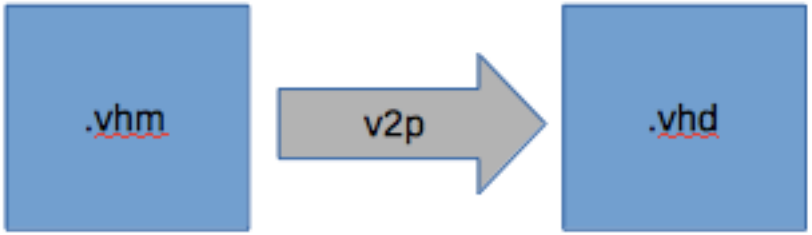
Going further : Structured VHDL Design Method

- In order to overcome the limitations of the classical 'dataflow' design style (large number of concurrent VHDL statements and processes, leading to bad readability and increased simulation time), a **'two-process'** coding method is proposed: one process contains all combinational logic, whereas the other process infers all (and only) the registers.
- The use of record types to increase readability and the safe use of variables to reduce simulation time. The method has been applied on several designs made by or for ESA.

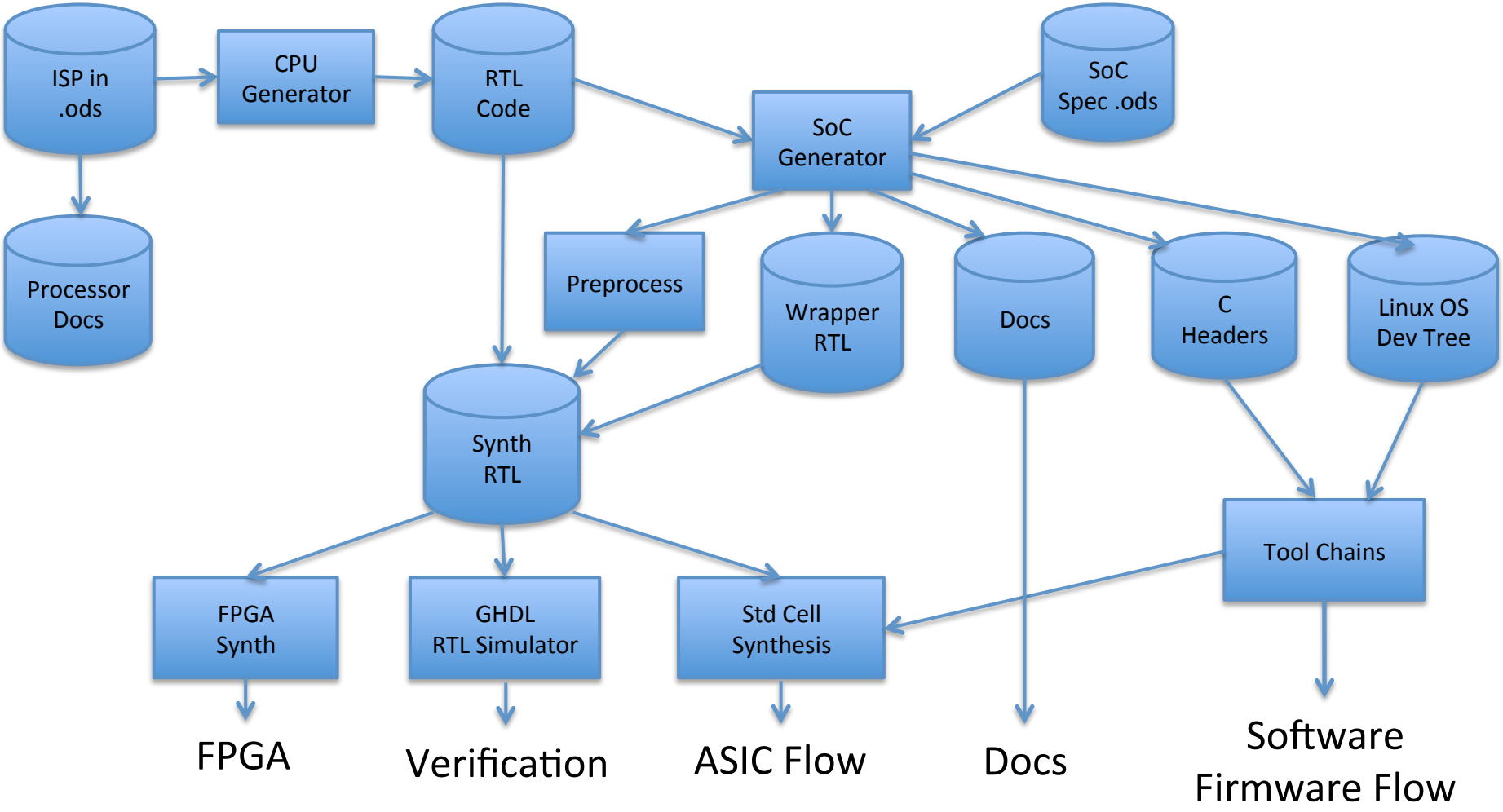
-Jiri Gaisler, **http://www.gaisler.com/doc/vhdl2proc.pdf**

Avoiding Common Errors

- We developed a pre-processor perl tool (v2p) to avoid latches, and other similar coding errors .
  - The `.vhm` file is a dialect of vhdl; sensitivity lists are generated by the perl script
- This has resulted in highly reliable RTL, and greatly increased the efficiency of our internal development process

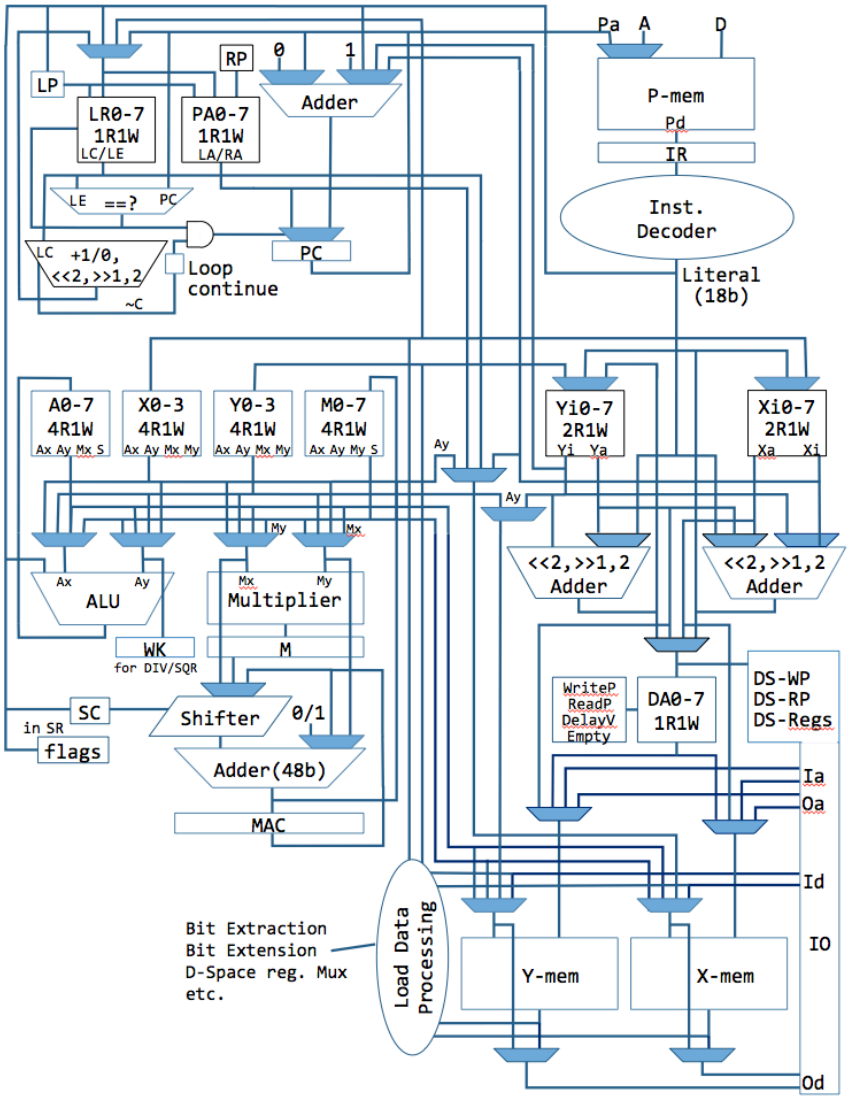


Even Further : Automated ISA -> RTL Sim->FPGA->ASIC->SW Tools Flow



18/36b S-Core DSP

- Development in Progress (Target Completion: August 2015)



DSP ISA and Code Map

ALU & MAC	35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	primary set
	0 0 ALU Xa Xd X Y MAC Ya Yd	Xa0(Xi4),Xc0(Xi6),Xi0,Xi2
		Ya0(Yi4),Yc0(Yi6),Yi0,Yi2
CAX Xin	Xin: Xi0-2, Xa:Xi{2b'10,n&1b'1} Xi 1 1	Xa+=Xi(w/Mask & Bit Rev.)
NOP		Xa =Xi4 (for STX/LDX if DA is empty)
STX Xd	Xd: A0,2,4,6,M0,2,4,6 Xd 0	@Xa = Xd(Xa -> DA[rp,wp]*) *)use DA
LDX Xd	Xd: X0-4 0 Xd 1	Xd = @Xa(Xa -> DA[wp] *) in some casess
LSWX	1 1 0 1	LP = ~LP (for two set case)
NOP	1 1 1 1	secondary set
reserved	others	Xa1(Xi5),Xc1(Xi7),Xi1
		Ya1(Yi5),Xc1(Xi7),Yi1
CAY Yin	Yin: Yi0-3, Ya:Yi{2b'10,n&1b'1}	Ya+=Yi(w/Mask & Bit Rev.) Yi
NOP		Ya =Yi4 1 1
STY Xy	Yd: A1,3,5,7,M1,3,5,7 0	@Ya = Yd(Ya -> DA[rp,wp]*) Yd
LDY Xy	Yd: Y0-4 1	Yd = @Ya(Ya -> DA[wp] *) 0 Xd
LSWY	1	LP = ~LP (for two set case) 1 1 0
NOP	1	1 1 1
reserved	1	others

Xi3,Yi3: always "0"

Xc/Yc
- [17] use cntrol field
- [16] "M" mask
- [15] "R" bit reverse
- [14:13] DA mode
  - 00: "_" nop
  - 01: "D" 1-cycle Delay
  - 10: "P" Pair
  - 11: "F" Pair Load/Store (Full four accesses)
- [8:4] encoded mask position
  - 1-17, the other is reerved
- [1:0] size
  - 00: 18b
  - 01: 1b(±1)

ADD Ax,Ay,Az	0 0 Ax Ay Az	Ax:X0,2/Y0,2/A0-3, Ay:X1,3/Y1,3/M0-3, Az:all	(C,Az)= Ax + Ay
SUB Ax,Ay,Az	0 1 Ax Ay Az	Ax:X0,2/Y0,2/A0-3, Ay:X1,3/Y1,3/M0-3, Az:all	(C,Az)= Ax - Ay
ADD Ax,Ay,Az	1 0 0 0 Ay Ax Az	Ax:X0,2/Y0,2/A0-3, Ay:A2-3, Az:all	(C,Az)= Ax + Ay
SUB Ax,Ay,Az	1 1 0 0 Ay Ax Az	Ax:X0,2/Y0,2/A0-3, Ay:A2-3, Az:all	(C,Az)= Ax - Ay
ADD Ax,Ay,Az	1 0 0 1 Ax Ay Az	Ax:M0-1, Ay:X1,3/Y1,3/M0-3, Az:all	(C,Az)= Ax + Ay
SUB Ax,Ay,Az	1 1 0 1 Ax Ay Az	Ax:M0-1, Ay:X1,3/Y1,3/M0-3, Az:all	(C,Az)= Ax - Ay
ADC Ax,Ay,Az	1 1 1 0 0 Ax Ay Az	Ax:X2/Y2/A0,2, Ay:X3/Y3/M1,3, Az:A1,3,5,7	(C,Az)= Ax + Ay + C
SBC Ax,Ay,Az	1 1 1 0 1 Ax Ay Az	Ax:X2/Y2/A0,2, Ay:X3/Y3/M1,3, Az:A1,3,5,7	(C,Az)= Ax +~Ay + C
ADC Ax,Ay,Az	1 1 1 1 0 0 0 Ax Az	Ax:X2/Y2/A0,2, Ay:A3, Az:A1,3,5,7	(C,Az)= Ax + Ay + C
ADC Ax,Ay,Az	1 1 1 1 0 0 1 Ay Az	Ax:M1, Ay:X3/Y3/M1,3, Az:A1,3,5,7	(C,Az)= Ax + Ay + C
SBC Ax,Ay,Az	1 1 1 1 0 1 0 Ax Az	Ax:X2/Y2/A0,2, Ay:A3, Az:A1,3,5,7	(C,Az)= Ax +~Ay + C
SBC Ax,Ay,Az	1 1 1 1 0 1 1 Ay Az	Ax:M1, Ay:X3/Y3/M1,3, Az:A1,3,5,7	(C,Az)= Ax +~Ay + C
ATN Ay,Az	1 1 1 1 1 0 0 Ax Az	Ay:X3/Y3/M1,3, Az:A1,3,5,7	explained later
SCN0 Ax,Ay,A4	1 1 1 1 1 0 1 Ay Ax	Ax:X0,2/Y0,2, Ay:X3/Y3/M1,3	explained later
SCN1 Ay,A4	1 1 1 1 1 1 0 0 0 Ax	Ay:X3/Y3/M1,3	explained later
LSWX #n	1 1 1 1 1 1 1 1 0 0 n		LP = n (Swich to n-th LOOP control set)
NOP	1 1 1 1 1 1 1 1 1 1 1		
reserved	other patterns		

MAD Mx,My,Mz	Mx:X0-1/Y0-1/A4-7,My:X2-3/Y2-3/M4-7,Mz:all	0 0 Mx My Mz	(Mz,MAC)= S(MAC + Mx * My)
MSB Mx,My,Mz	Mx:X0-1/Y0-1/A4-7,My:X2-3/Y2-3/M4-7,Mz:all	0 1 Mx My Mz	(Mz,MAC)= S(MAC - Mx * My)
MUL Mx,My,Mz	Mx:X0-1/Y0-1/A4-7,My:X2-3/Y2-3/M4-7,Mz:all	1 0 Mx My Mz	(Mz,MAC)= SR(Mx * My)
MAD Mx,Mz	Mx:X0-1/Y0-1/A4-7, Mz:all	1 1 0 0 0 Mx Mz	(Mz,MAC)= S(MAC + Mx^2)
MAD My,Mz	My:X2-3/Y2-3/M4-7,Mz:all	1 1 0 0 1 My Mz	(Mz,MAC)= S(MAC + My^2)
MSB Mx,Mz	Mx:X0-1/Y0-1/A4-7, Mz:all	1 1 0 1 0 Mx Mz	(Mz,MAC)= S(MAC - Mx^2)
MSB My,Mz	My:X2-3/Y2-3/M4-7,Mz:all	1 1 0 1 1 My Mz	(Mz,MAC)= S(MAC - My^2)
MUL Mx,Mz	Mx:X0-1/Y0-1/A4-7, Mz:all	1 1 1 0 0 Mx Mz	(Mz,MAC)= SR(Mx^2)
MUL My,Mz	My:X2-3/Y2-3/M4-7,Mz:all	1 1 1 0 1 My Mz	(Mz,MAC)= SR(My^2)
MULD Mx,My,Mz	Mx:X01/Y01/A45,67,My:X23/Y23/M45,67,Mz:M0123,4567	1 1 1 1 0 0 Mx My Mz	MAC=UU(MxL*MyL) ,MzLL=MACL
MULD Mx,Mx,Mz	Mx:X01/Y01/A45,67, Mz:M0123,4567	1 1 1 1 0 1 0 0 Mx Mz	MAC=UU(MxL^2) ,MzLL=MACL
MULD My,My,Mz	My:X23/Y23/M45,67,Mz:M0123,4567	1 1 1 1 0 1 0 1 My Mz	MAC=UU(MyL^2) ,MzLL=MACL
(2nd step)			MAC=US(MxH*MyL)+MAC>>18
(3rd step)			MAC=US(MyH*MxL)+MAC ,MzLH=MACL
(4th step)			MAC=SS(MyH*MxH)+MAC>>18,MzHL=MACL
(5th step)			MzHH=MACH
LSWY #n		1 1 1 1 1 1 1 0 0 n	LP = n (Swich to n-th LOOP control set)
NOP		1 1 1 1 1 1 1 1 1 1 1	

What can you do with this now?

- Download bitstream + vmlinux/initramfs, install on fpga board, boot kernel to shell prompt
- HOWTOs with background info
  - Where to order FPGA board(s)
  - Download and install xilinx/digilent tools
    - Free download for linux/mac, but alas no open source bitstream compiler yet. OpenOCD installer is todo item.
  - Build new bitstream from source
  - Program nommu Linux (gcc/binutils/musl toolchain)

Nommu.org

- Watch this space for a Git Repository and a mailing list.