Empowered by Innovation NEC

MPSoC with multi Configurable Processors and Design Environment

Jack, Kazutoshi Wakabayashi
EDA R&D center, Central Res. Labs. NEC corp.
and
chief architect, CWB business promotion, NEC System Technology, Ltd.
System IP Core Res. Labs, NEC Corp,
Visiting Professor, JAIST

URL:www.cyberworkbench.com
Email: wakaba@bl.jp.nec.com

June 28 (Th) 2008

U can change.

CyberWorkBench®
Pioneering C-based LSI Design

Agenda

One feature of NEC's ASSP (DAV,Mobile) is wide usage of C-based synthesis and verification.

This impacts ASSP architecture and design flow these five years.

(We applied Behavior synthesis even for CPUs)

Keywords:

1. Multiple Configurable Processors

(custom CPU + custom DSP)

2. Dynamic Reconfigurable Processors

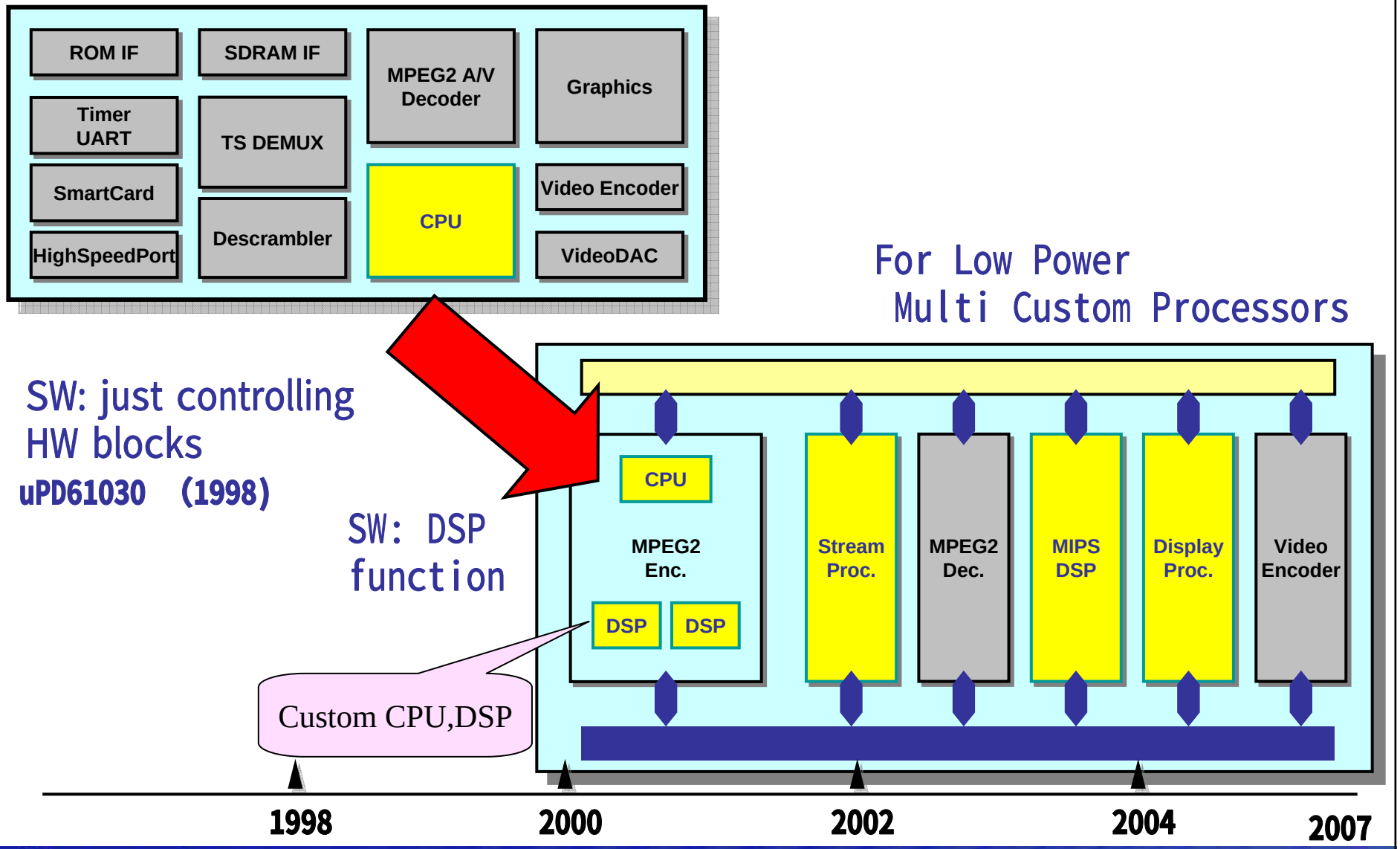
(flexible Hardware and controller)

3. **C-based Synthesis and Verification**

4. Reverse Trend : SW to HW

HW solution for RT operation, power efficiency

Global trend: Roll of SW(CPU) is increasing

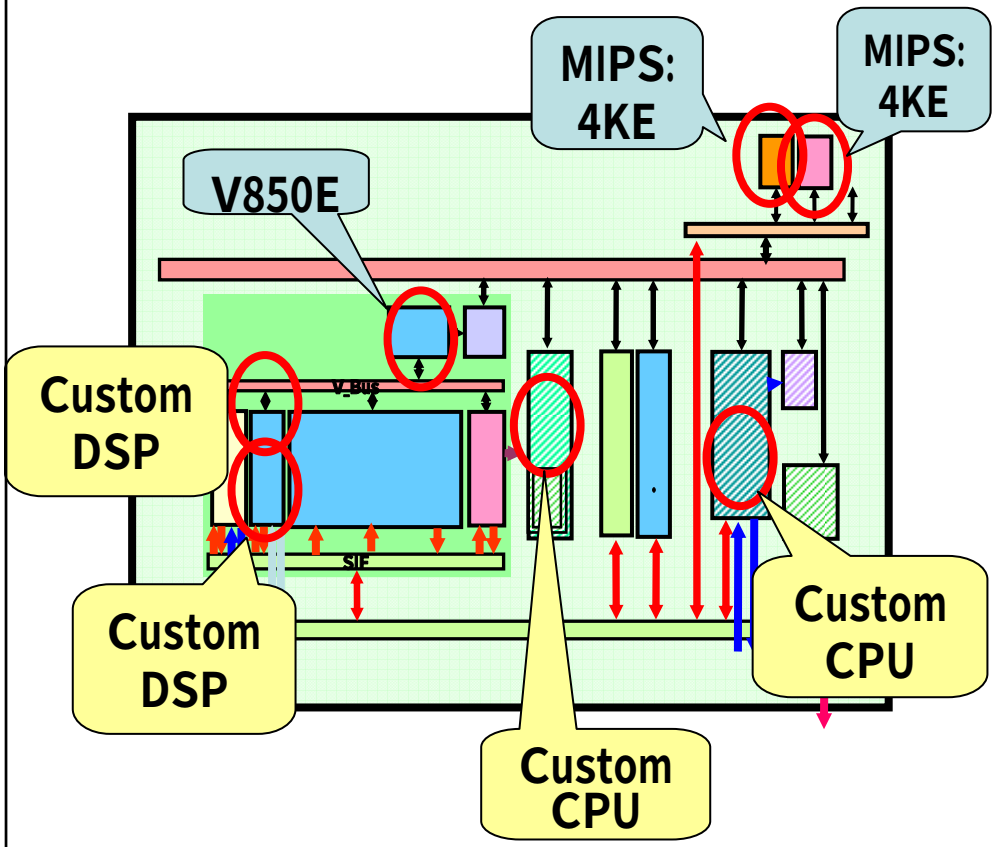


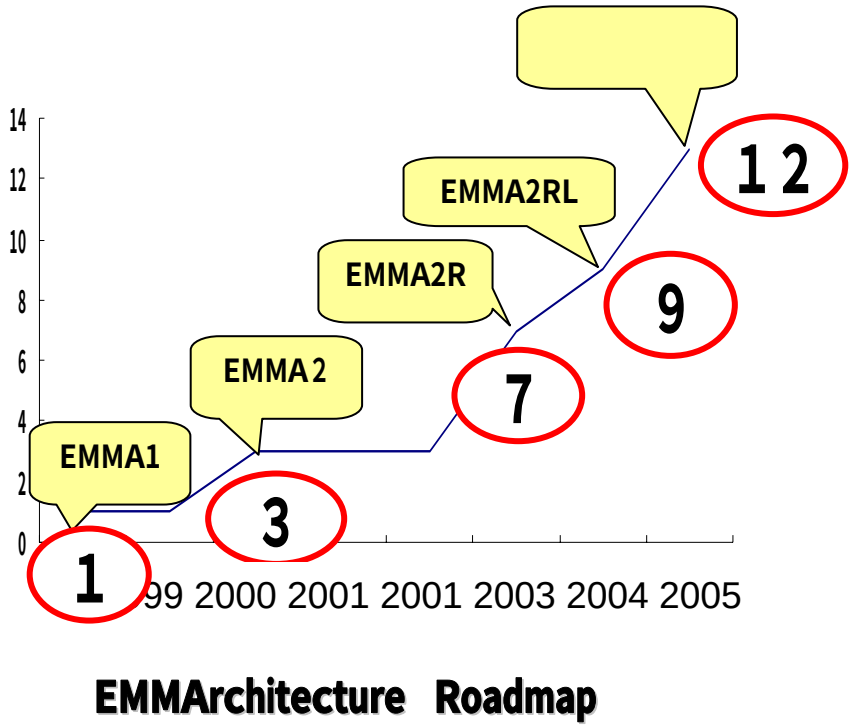
Global trend: MPSoC: Software-oriented system

V850E General CPU: for embedded SW : a few

Custom CPU,DSP configurable Proc. : ~20

MPSoC

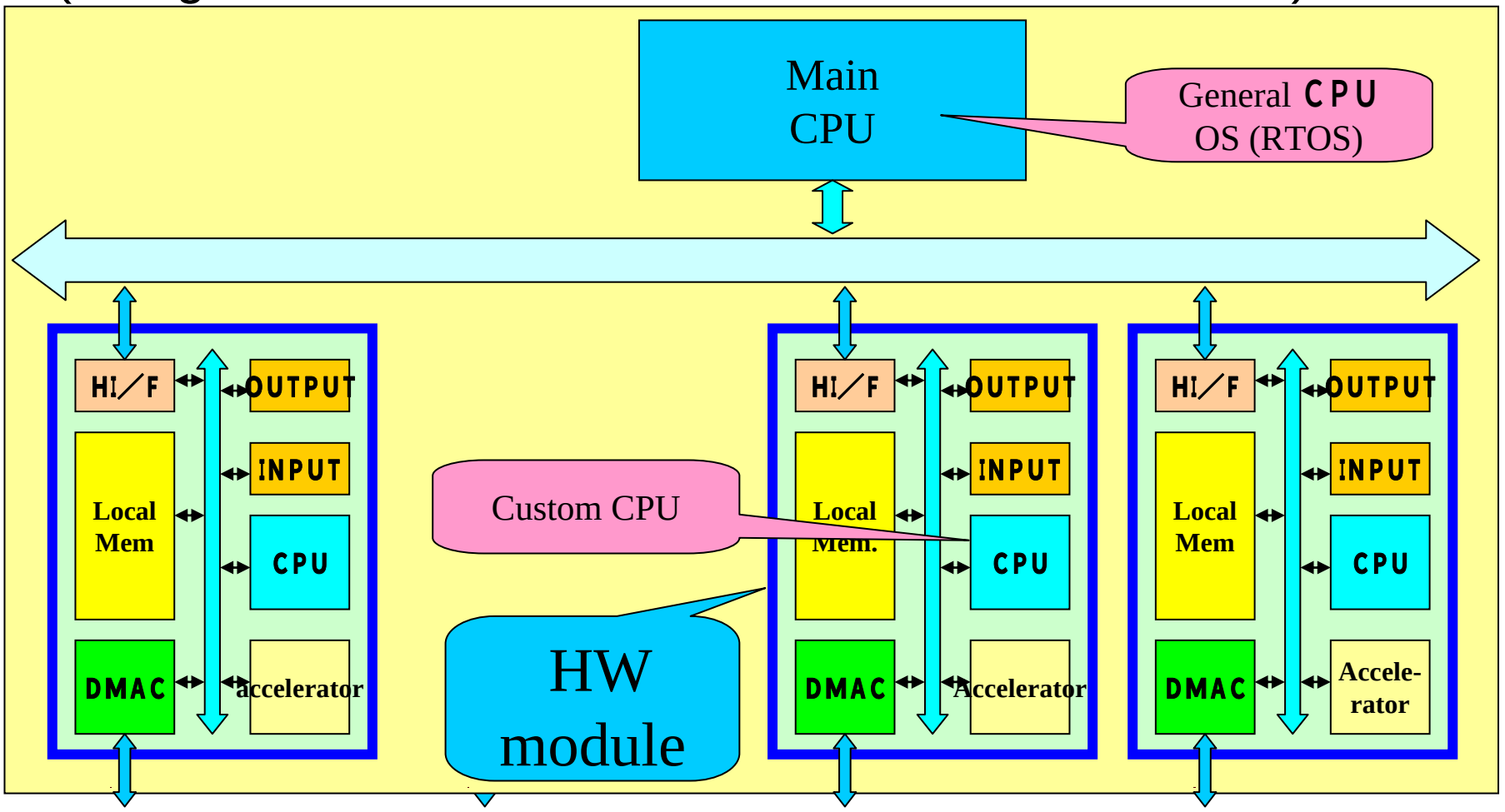




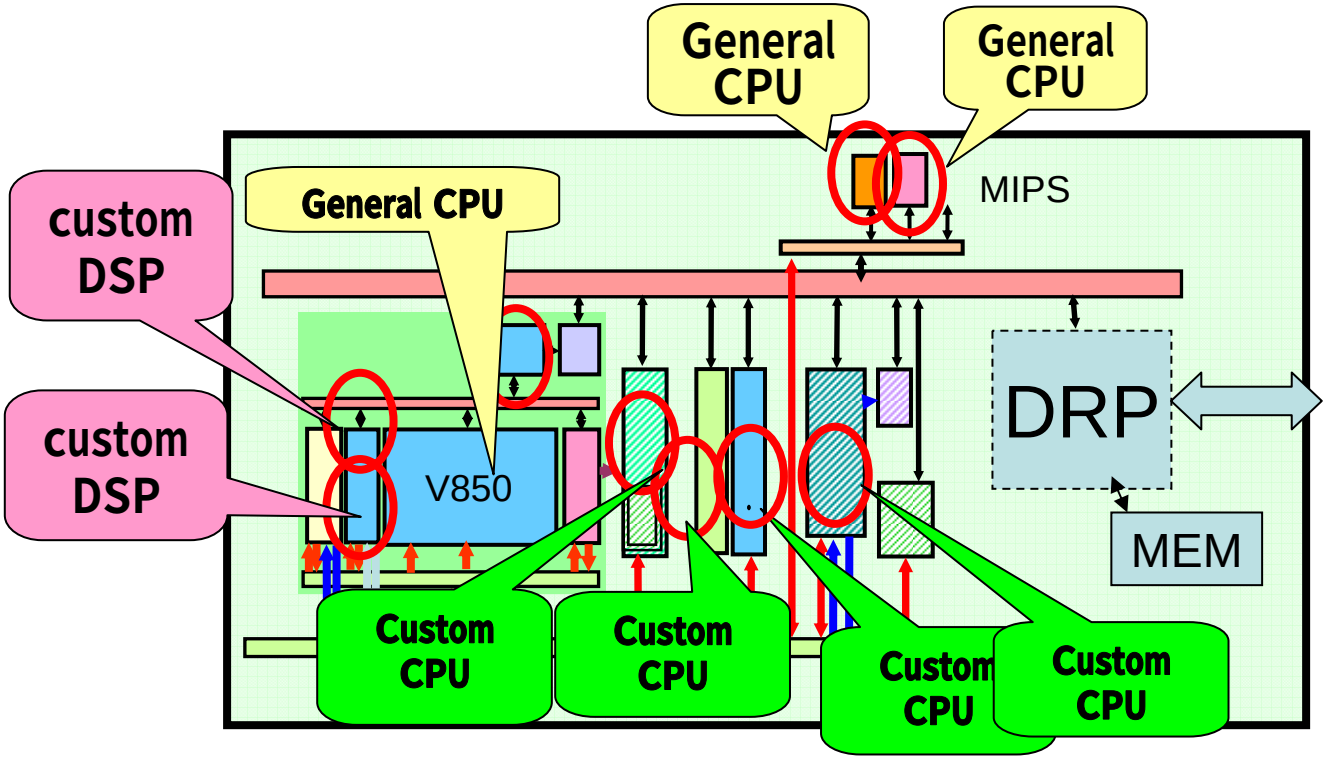
EMMArchitecture Roadmap

Custom CPUs exist usually inside HW module

of “custom CPU” is 20 or more, but they are not Main CPUs.
Main CPU is just a few… (not *so-called* MPSoC)
(though, custom CPU communicates with Main CPU)



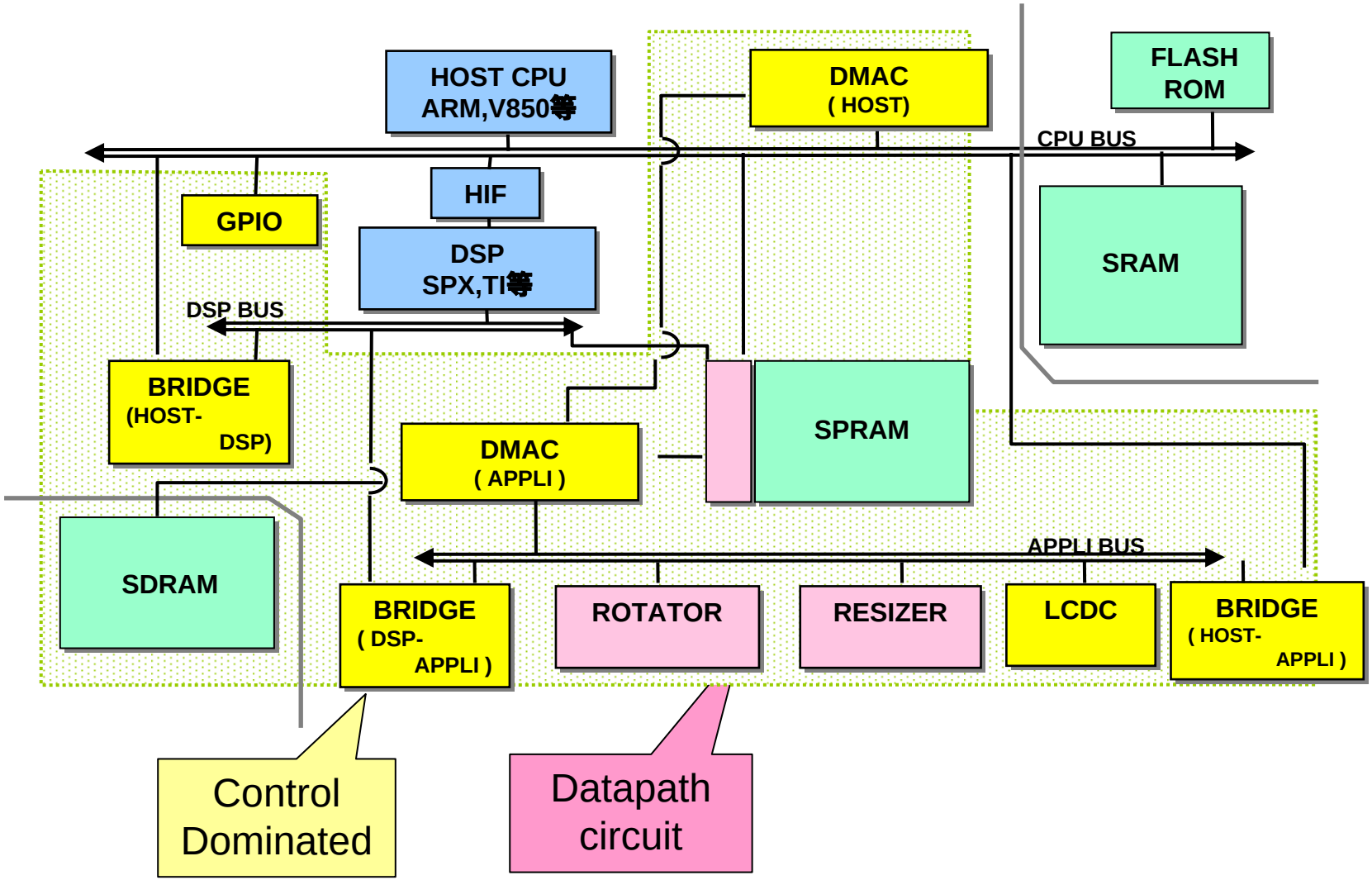
Our ASSP architecture
A few General CPUs + Multi Custom Processors + HW modules (+ DRP)



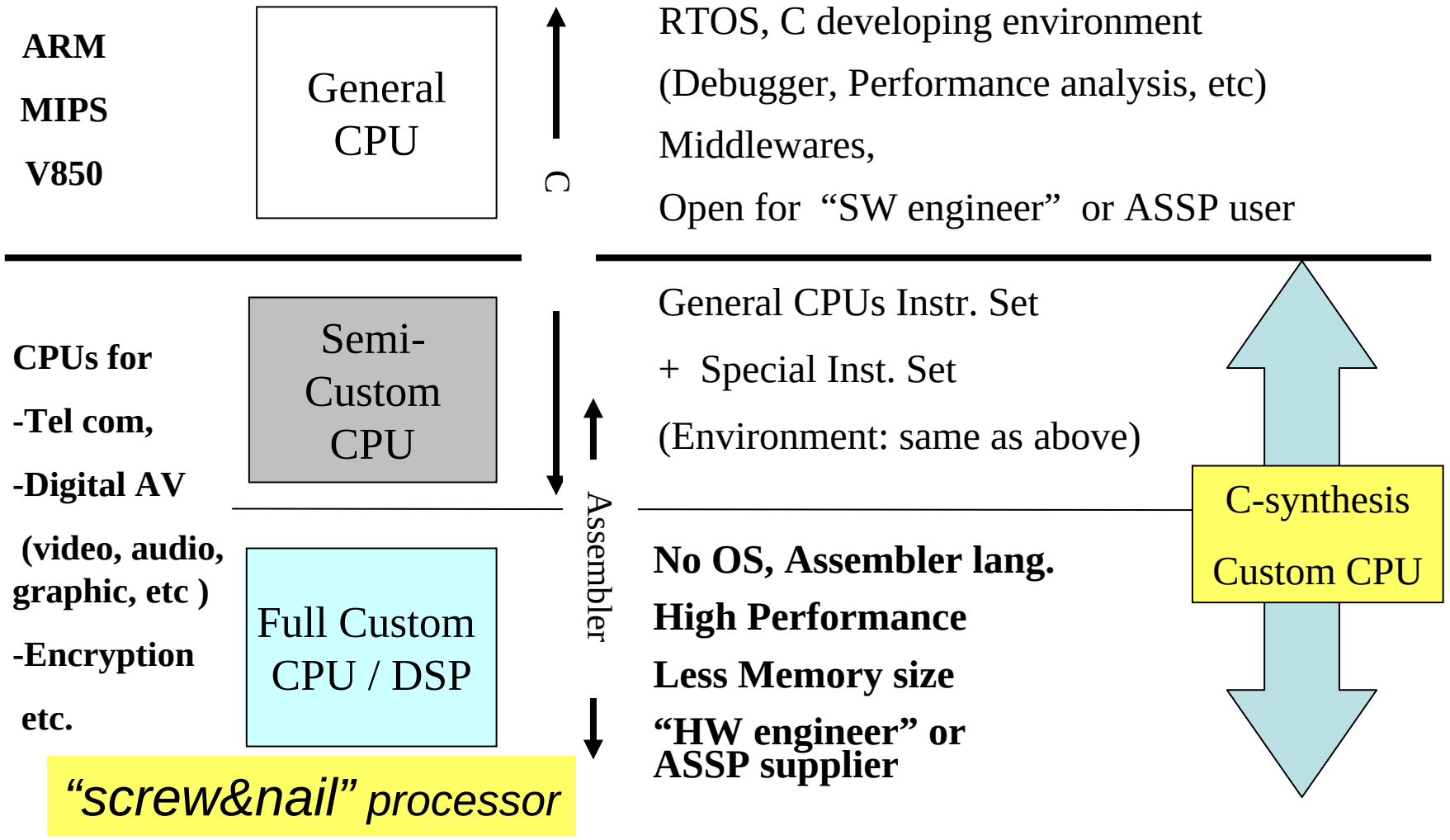
DRP:
Dynamic (every 1ns) reconfigurable (multi context) FPGA or Programable array

Custom CPU, DRP, HW modules are **based on C-synthesis**.
So, they can be replace into another.
e.g. C source for DRP can be synthesized into pure HW!!
CPU tasks are implemented in DRP (really happens often)!!

"All-in-C" : All Modules with C-based Synthesis



1. Multi Custom Processors
Types of CPUs on our ASSP



Merits of Custom CPU/DSP

1) HW control (sequencer)->SW control

more flexible, debuggable after chip fab.

2) Special Instruction Set : higher Performance

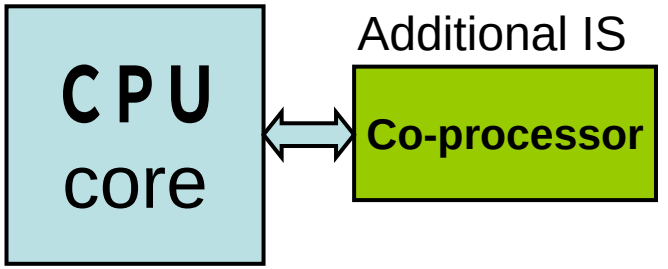
Low Power and low clock frequencies, memory save

Requirements for our favorite custom CPU/DSP

- Real time function : No Cache, avoid bus collision
- HW-like Performance for special procedures
- Smaller size of Memory (Instruction & Data)
- Bit handing, Branch with bit
- read/write of I/O reg.
- Easy to design/ Modify (few weeks for design)

Types of Additional Instruction Set

- **ALU extension：Easy, but very limited**
- **Co-Processor**



Pros: Easy even with RTL based Design

Cons: no resource sharing among co-processors and core-CPU

slow data transfer CPU-core and Co-processor(e.g. memory mapped IO)

- **Complex Instruction into CPU core**

Pro： sharing registers, FUs with core CPU and Extended Instructions

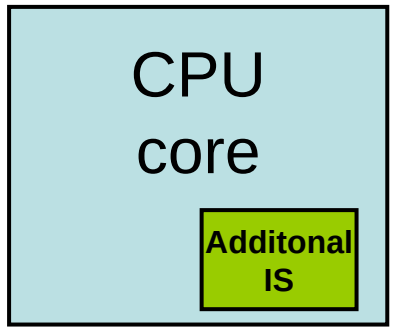
Direct read of multi public register

-> higher performance

Cons: CPU core has to be changed

(difficult for RTL design, timing closure)

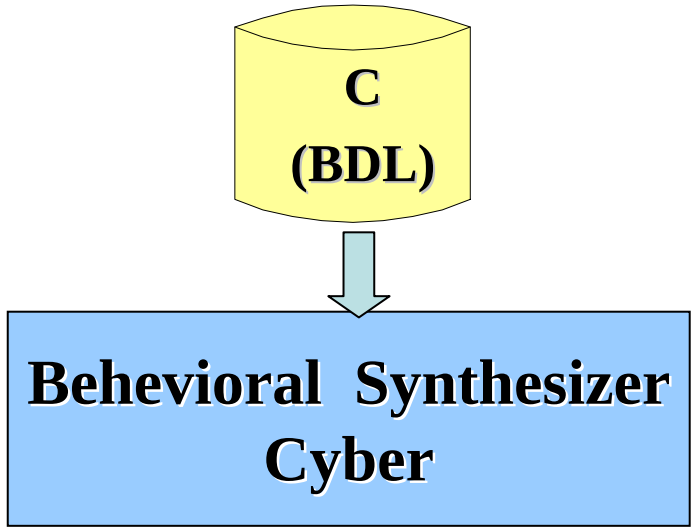
(***CPU core should be described in C***, C-synthesis!)



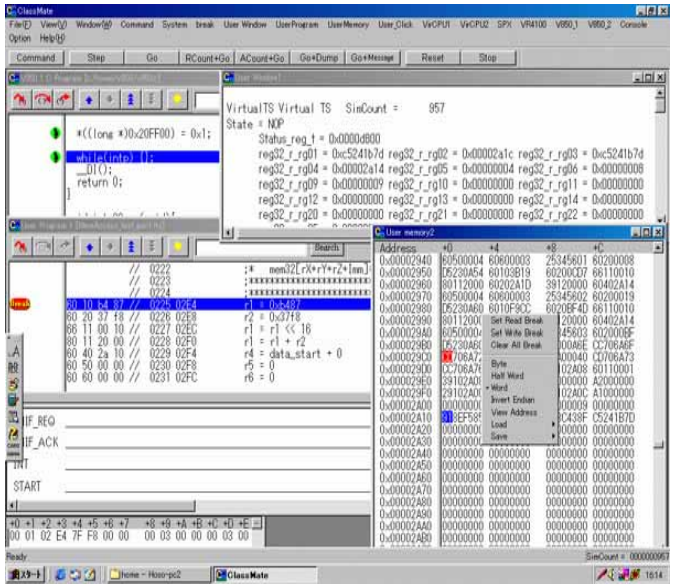
"C-base behavior synthesis" supports all types

Tool Flow for Custom CPUs

C behavior of Custom CPU



Source Code Debugging



Cycle Accurate Models
For other HW modules
+ General CPU models

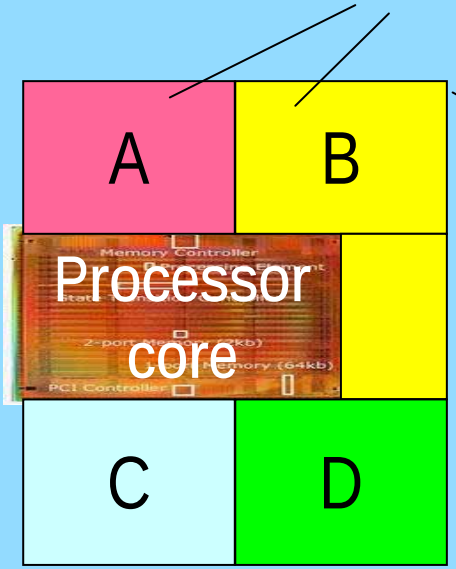
FPGA emu.

HW-SW co-sim

Configurable Processors Generation
RTL-based vs Behavioral-synthesis-based

Logic Synthesis based

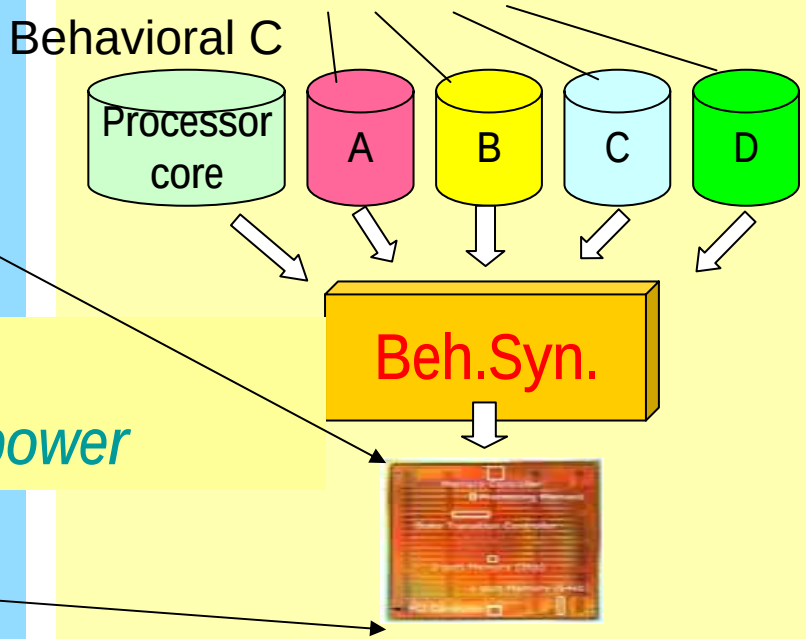
Adding Instructions : RTL Co-processors



Special IS: Limited
Base Processor is not flexible

Behavioral Synthesis based

Adding Instructions: Melted into Core



Smaller
Lower power

Special IS: less limited
Base Processor is flexible

Effects of Resource Sharing

Base I.S. : 81
Adding I.S. : 24

For Stream Processing
e.g.CRC, check-sum,
start-code check...

Base Processor : 33.9K Gate
After Adding IS : 42.6K Gate
only 8.7K(25%) increase

Effects of FU sharing among
Base IS's and Adding IS's

Kind	Bit width	Before adding	After adding	Diff.	Necessary FUs in new IS
>>	32	1	3	2	8
>>	33	1	1	0	0
+	3	1	1	0	0
+	12	2	2	0	0
+	14	0	1	1	4
+	16	4	5	1	3
+	32	3	3	0	6
<=	14	4	4	0	0
<=	32	1	1	0	0
−	3	1	1	0	0
−	5	0	1	1	2
−	32	2	2	0	0
<<	32	1	1	0	0
<<	33	1	1	0	0
*	8	1	1	0	0
>	16	0	2	2	2
>	32	1	1	0	0
>=	32	1	1	0	0
<	16	0	2	2	2
<	32	1	1	0	0

Adding IS's include
Six Add Operations(32bit),
But No FU increases

Adding extra Instractions :
Not big area impact like co-processors

Summary for our custom CPU/DSP design

We are "C-maniac"

all modules are C-synthesis: this changes ASSP platform

1) **CPU is described in behavioral C,**

and RTL is synthesized with our behavioral synthesizer (Cyber)

2) **Additional Instruction set is also described in behavioral C**

1)CPU basic Instractions and 2)additional Instructions share resources.

3) **ISS of the CPU is also generated with Cyber, used for entire SoC simulation including HW modules is constructed**

4) **C compiler or C-like structured assembler**

(semi-custom CPU case, RTOS and development env. can be used. Additional IS is handled as macro instruction.)

What size of additional Instruction is appropriate?

Fine grain Instruction : Flexible, but low performance

Coarse grain Instruction: HW like performance, but low flexibility

1) Find bottleneck contains **many general CPU instructions**.

2) Hierarchical Instruction (for flexibility)

ex. Level 3 : function level instruction (DES encryption)
Level 2: SIMD, Protocol, combination (2bit shift&mask)
Level 1: Reg access, single ALU, testing, status checks

⇨ Bug, Spec. Change : Insted of ECO,
HW bugs in level 3, can be realized with level 2 or 1.
(use current chip until next ES)

Example of Additional Instruction

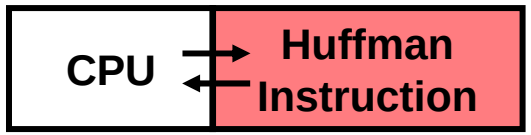
basic example:several codes -> one instruction
e.g. Huffman Decoding Instruction

Only with Basic Instractions which Original CPU core has

```
tmp32A = FAST_LEN_TABLE;
tmp32A = memr32(tmp16a + tmp32A);
tmp16b = tmp32A >> 16;
tmp16a = tmp32A;
if (tmp16b != 0) {
   tmp16b = VLD_GetBitReverse (tmp16b);
}
Len = tmp16a + tmp16b;
```

Additional Instruction

```
Len = VLD_CalcLenDist (tmp32A, TableData1, 1);
```



7 instructions → 1 instructions
7 cycles 2 cycles

☆Large procedure containing hundreds instructions, might be synthesized as a HW accelerator.
(e.g. highly parallel=many FUs, low power, ***routability***)

Additional IS containing control flow: special branch, loop instruction

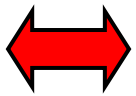
branch with bit calculation: common for DAV, mobile

Function: If 25th bit of R3 is 0, then goto L1

Base Instructions

```
mov   r2,0xfeffffff
and   r3,r2
jz    L1
```

3 instruction
(3 cycle)



New Instruction

For custom CPU

If(bit(r3,25)==0) goto L1

1 instruction
(1 cycle)

Result: “ Screw & snail” CPU

C(BDL)	2,267 Line
RTL(verilog)	12,985 Line
# of IS	105
clock	108 MHz
Gate size	42.6 K Gate
Man Power (design)	3.0 MM
(C++ simulation)	3.5 MM

Basic：81,
Special：24

Less than 1% area -> dozens is fine, but for 1000, 42KG is too large.

First Trial

Next version took
Much less
Just inserting Specials

Effects of C-based design

1. Flexible enough to follow spec changes

2.HW/SW co-design
SW and HW guys discussed on additional ISs

3 Smaller circuits with resource sharing

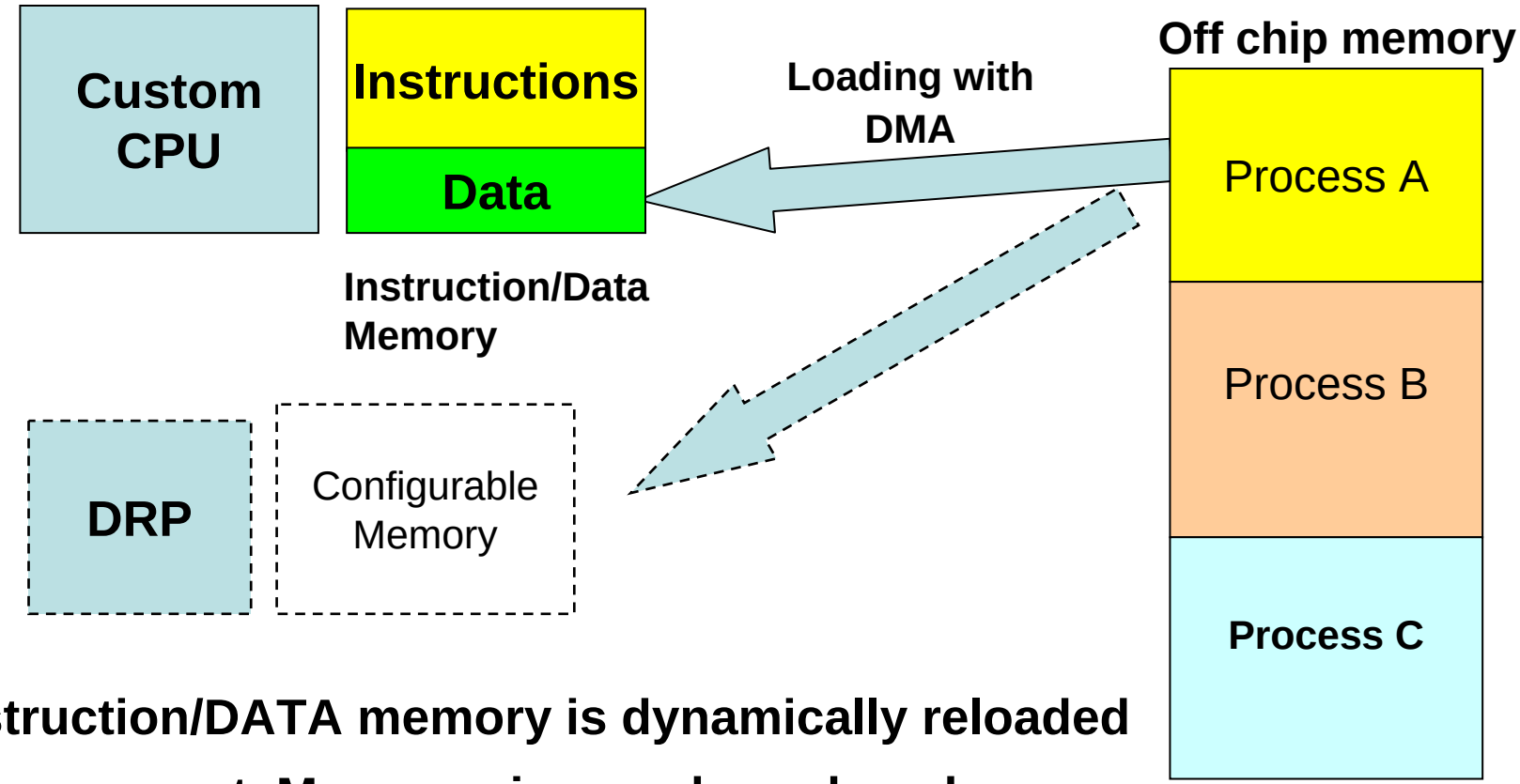
Comparison of Custom Processors with C-Synthesis vs. Manual RTL

Configurable Processor	Behavior C	RTL	Rate
Code Size	1.3KL	9.2KL	7.6X
Simulation (*1)	61Kc/s	0.3K	203X
Gate Size	19KG	18KG	~5%+

"Screw&nail" processor should be with C-synthesis, since custom CPU design effort should be small enough!

*1 Pentium3@1GHz, Testbench, debug included

Dynamic Program Loading for custom CPU and DRP

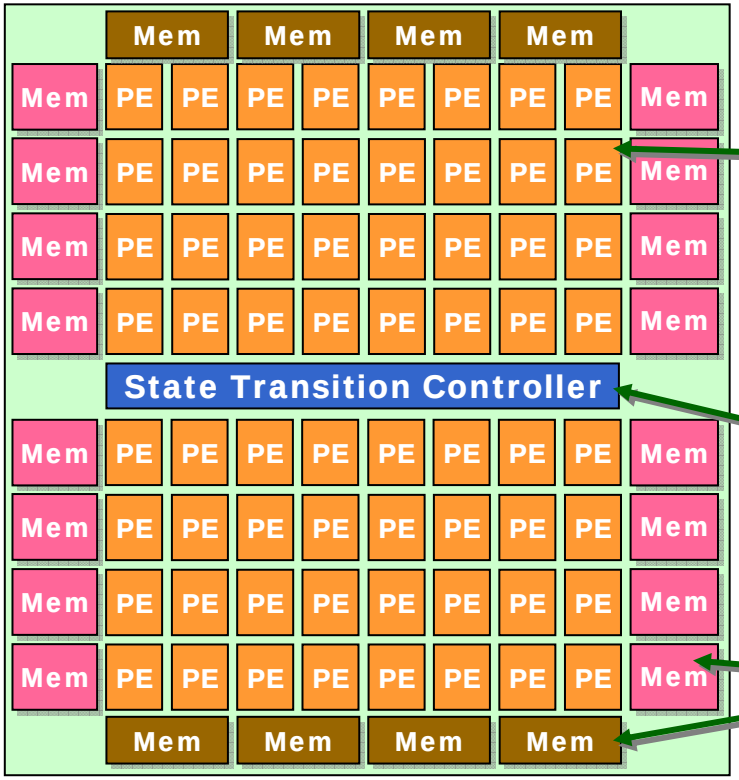


Instruction/DATA memory is dynamically reloaded upon request. Memory size can be reduced.

With C++ cycle accurate simulation, good size of memory should be Determined.

2. DRP Architecture (NECEL)

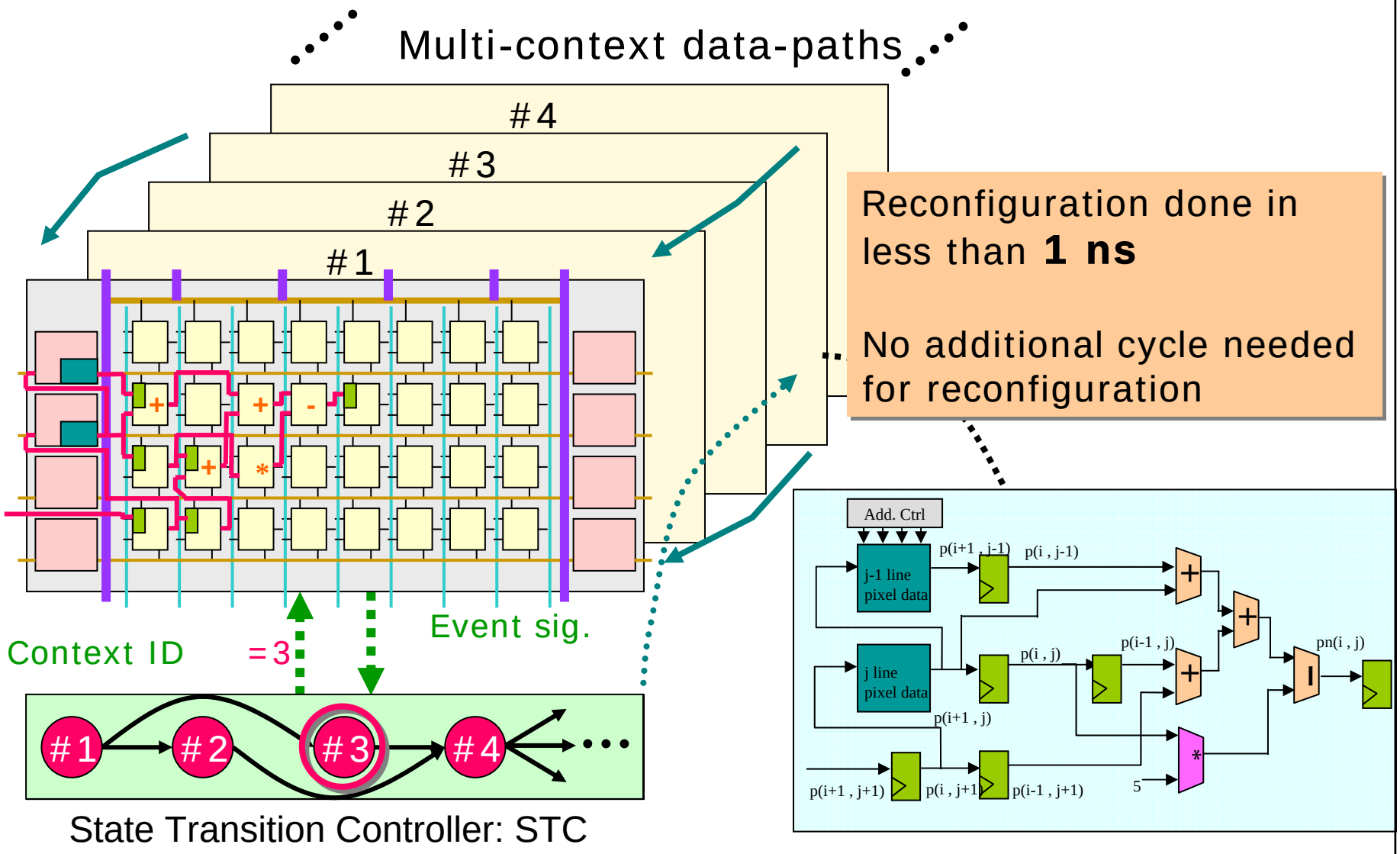
DRP Tile (variable array size)



- Array of byte-oriented processing elements
- Fully programmable inter-PE wiring resources
- STC: a simple sequencer
- Array of configurable data memories

- Fine-grained processor-based programmable array architecture
- Architected for stream data processing, such as NW packet, motion/still picture, and wireless data streams, etc.

Mechanism of Dynamic Reconfiguration



DRP's roll in ASSP

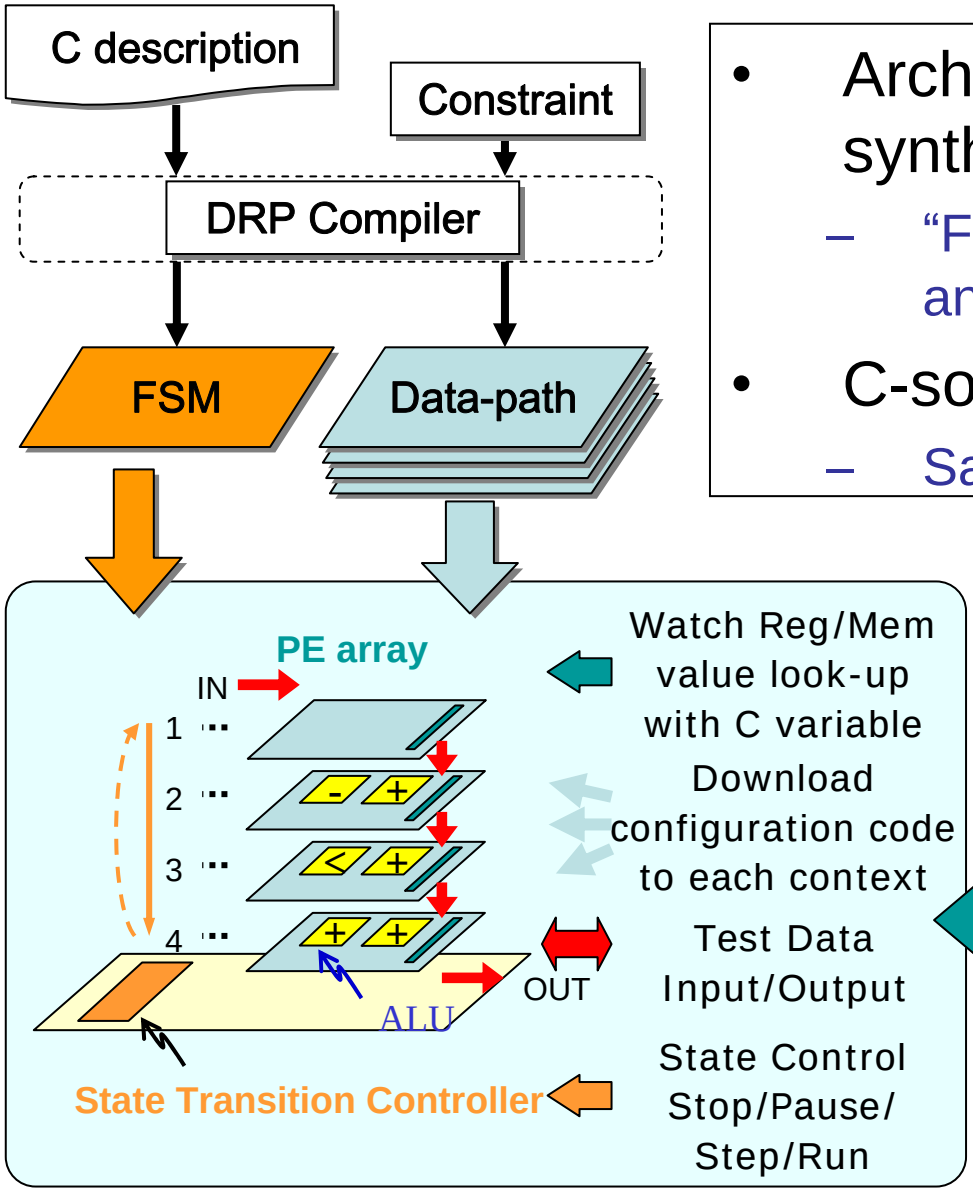
1) Original Roll : programmable HW in ASSP (faster than DSP)
for DSP accelerator, Encryption, etc.
several rolls of different sized procedure
(Dynamic context , Dynamic program re-loading)

2) Proxy of Main CPU
controlling chip, software procedure instead of main CPUs

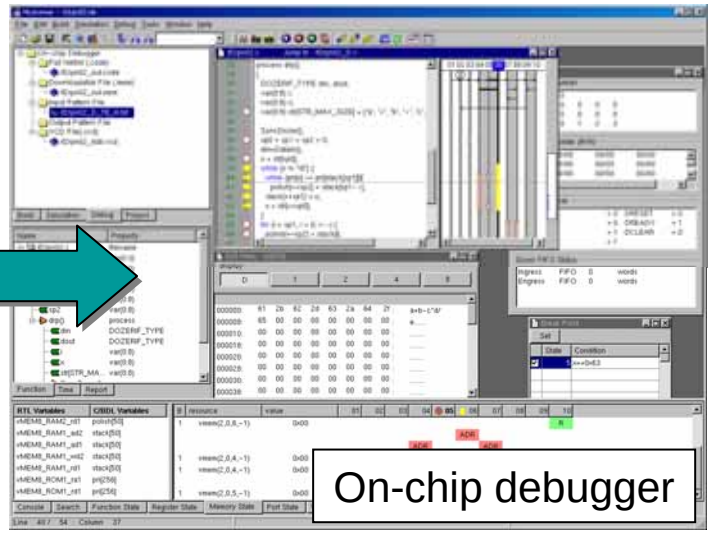
Main CPU is not powerful enough to perform several jobs
in terms of performance and size
-> several tasks for main CPU are transferred to DRP after chip fab.
(dynamic program re-loading: very fast for DRP)

DRP gives flexibility to ASSP
(CPU is not flexible enough in terms of performance)

DRP: Tool Driven Architecture!

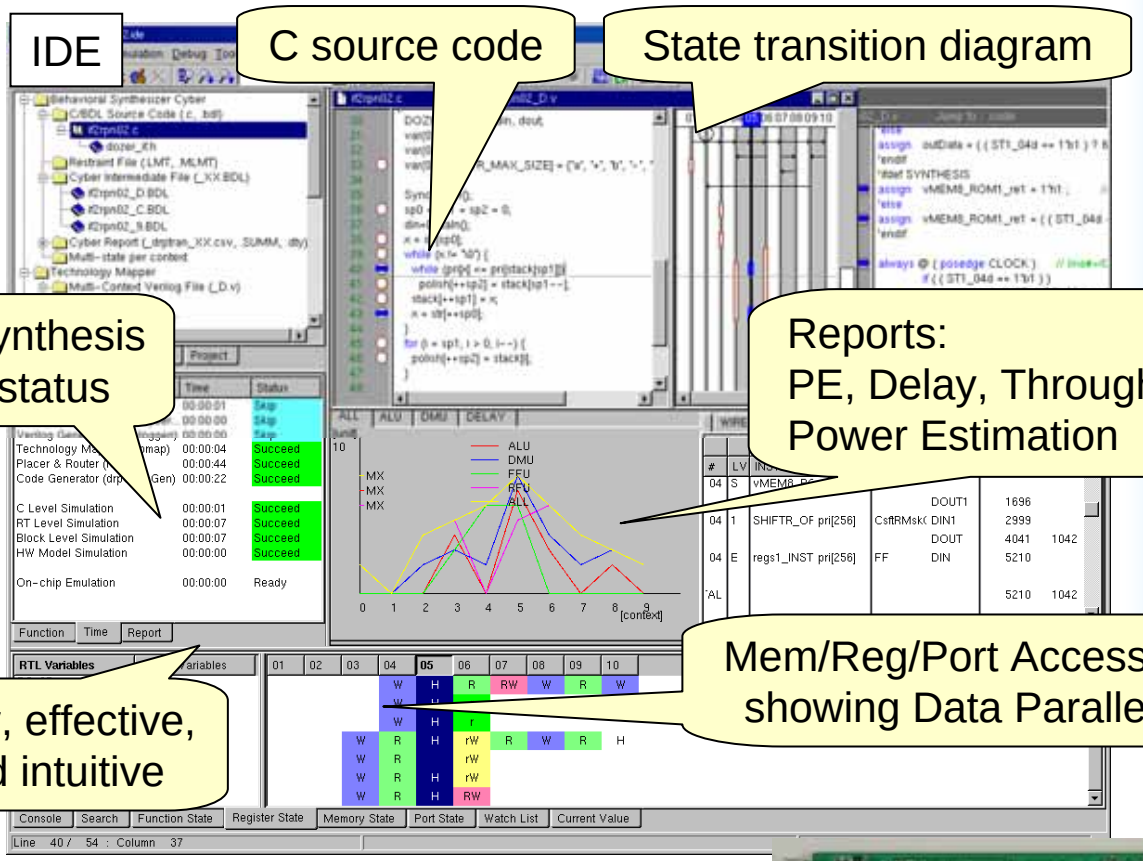


- Architecture is derived from C-synthesis “Cyber”
  - “FSMD” directly mapped onto “STC and PE array”
- C-source code Debugging
  - Same as Pentium, Arm, MIPS!

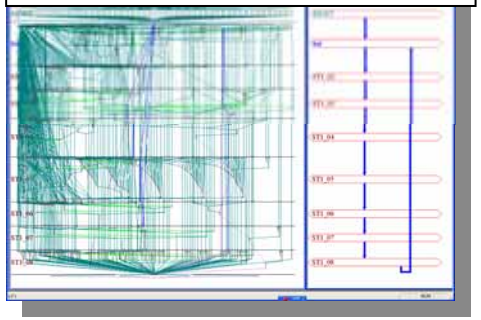


DRP : Fully Integrated Design Environment

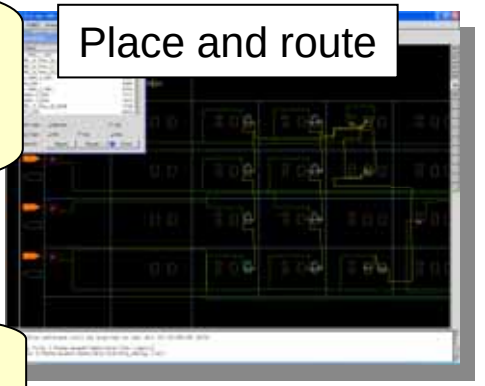
Variety of graphical feedbacks provided for designers
in order to support C source code optimization



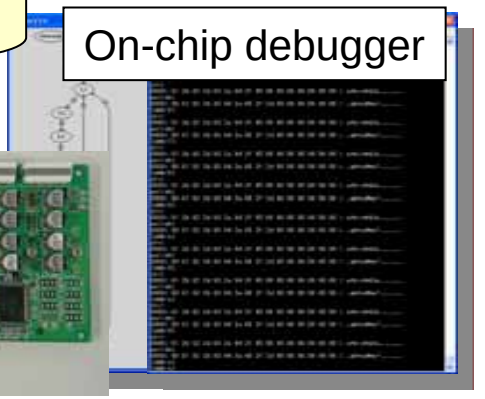
Behavioral synthesizer

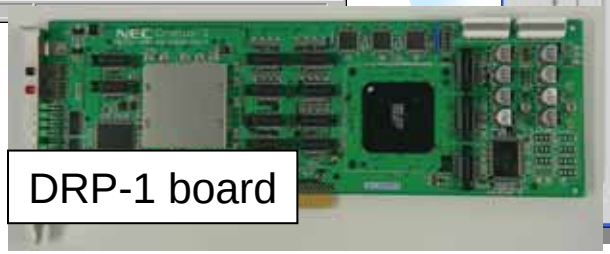


Place and route



On-chip debugger

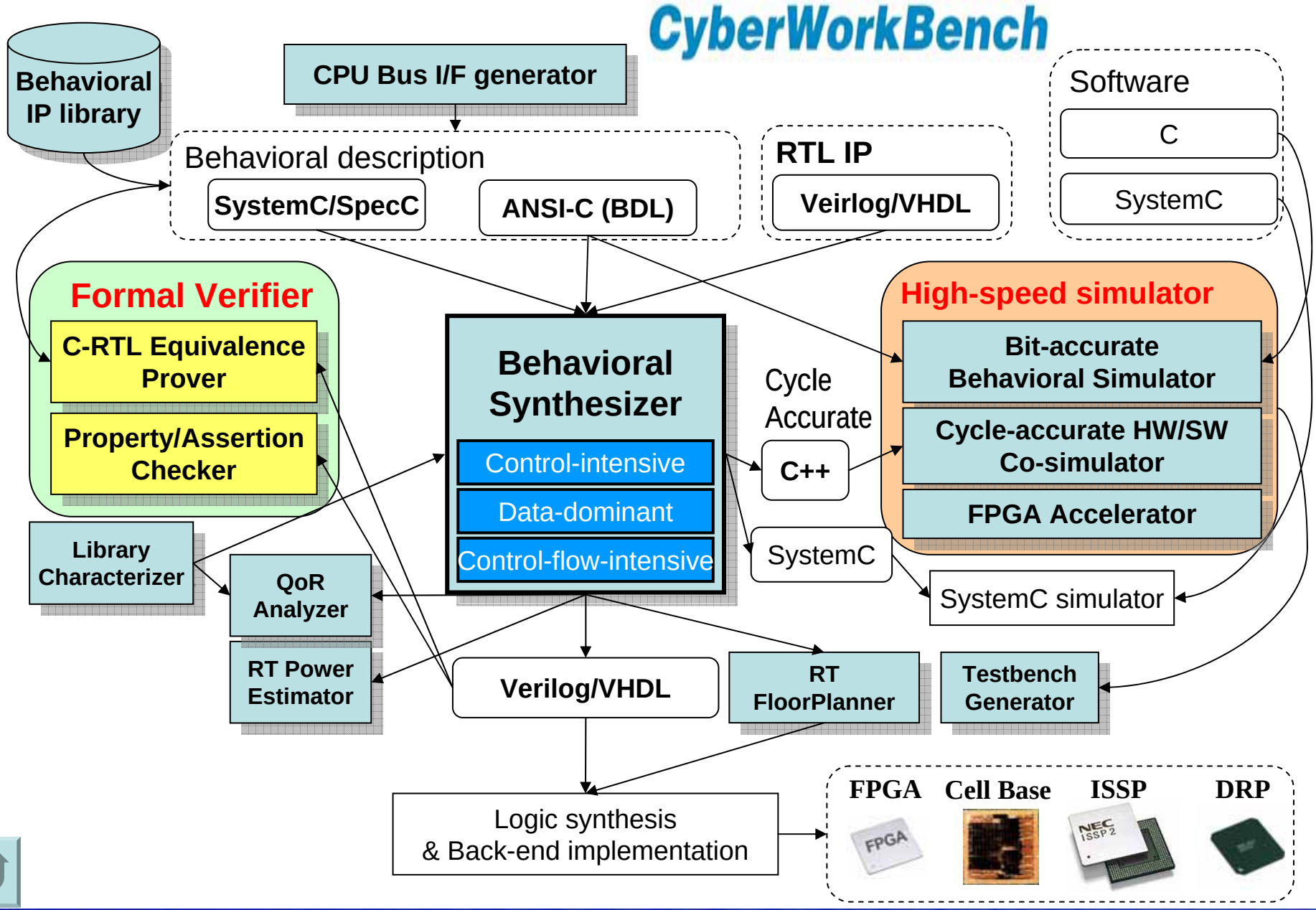




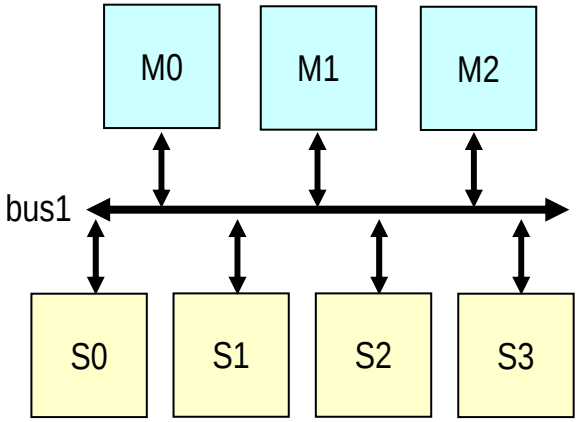
DRP-1 board

C-source level debugging for HW is finally supported just like IDE for CPU.

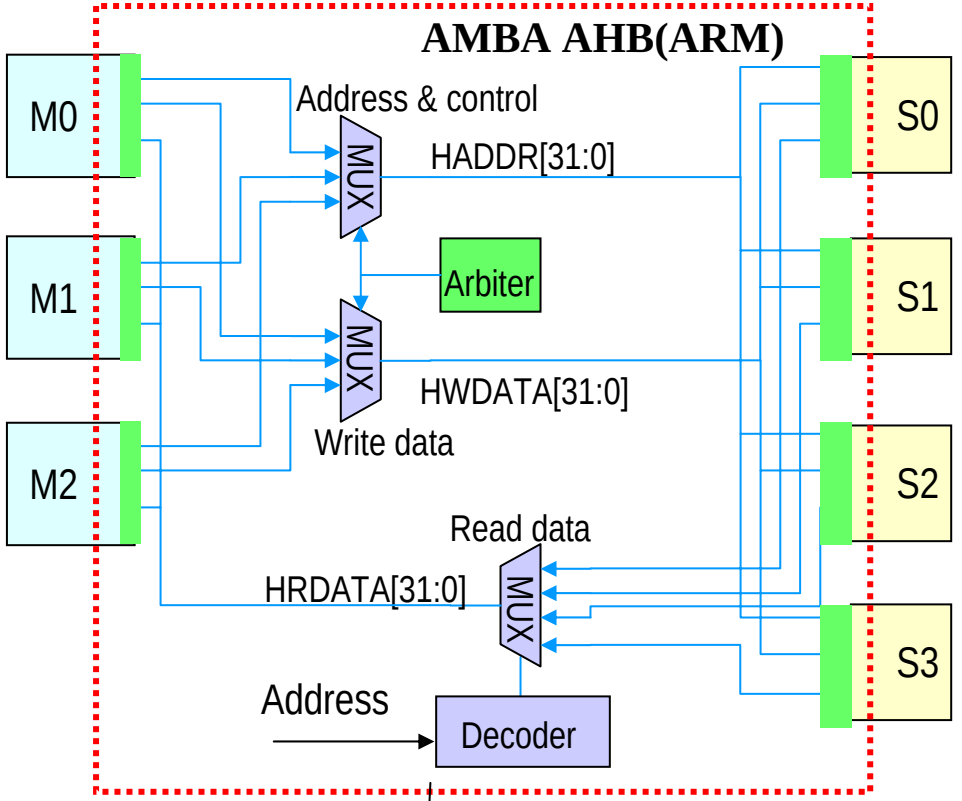
3. "All-in-C" : all tools work for original C source code



How to connect modules and CPUs?
CPU Bus and BUS I/F generator: ex. AMBA AHB,AXI

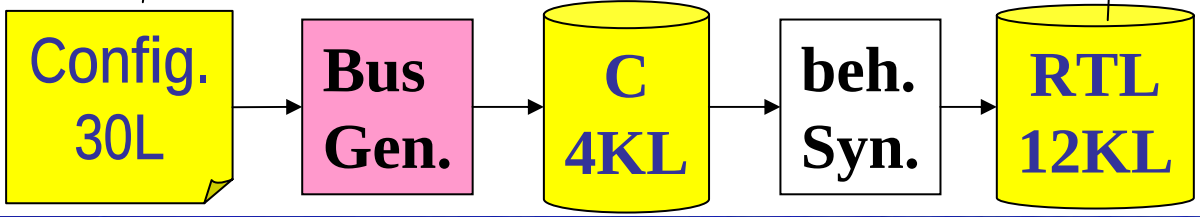


Generate



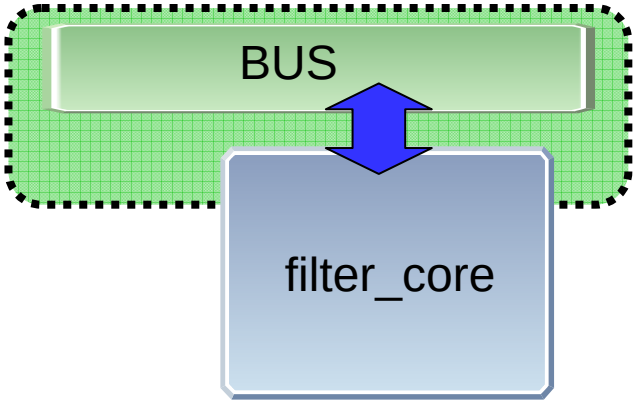
- Kind=AMBA_AHB
- Bus Master {M0, M1, M2}
- Bus Slave { S1,S2,S3,S4}
- Arbiter : {RoundRobin | FixedPriority}

behavior



Bus and I/F

Communication via BUS



CWB generates bus I/Fs, a bus, and an arbiter from our bus access APIs and a bus definition.

bus definition (for BUS synthesis)

```
defbus AMBA_AHB {                          /* on-chip bus protocol(AMBA AHB) */
  width address =  32;                     /* address bus     bit width */
  width data =     32;                     /* data bus bit width */
  module master ={pci2master, filter_core}; /* master UDL list for this bus */
  module slave =  {slave2sdram,filter_ctrl}; /* slave UDL list for this bus */
  mode arbiter_rule =RoundRobin;           /* arbitration rule */
} bus1;                                    /* bus instance name */

module AMBA_AHB_MASTER {                   /* master spec. declaration */
  mode burst =            Enable;          /* burst transfer capability */
  mode data_transfer =Direct;              /* UDL-I/F  communication type */
  mode clock =            Enable;          /* clock signal source declaration */
  mode reset =            Enable;          /* reset signal source declaration */
} pci2master,filter_core;                  /* master instance name */

module AMBA_AHB_SLAVE {
  mode burst =     Enable;                          /* burst transfer capability */
  map address =   0x00000000-0x00ffffff & 0xffffff00;   /*  address map & decoder mask */
} filter_ctrl;                             /* slave instance name */
```

API for BUS I/F (for BUS I/F synthesis)

```
void read_stream(uint24 *p, ureg32 *idx)
{
  *p = CBM_single_read(*idx) ;
  *idx += 4 ;
  return ;
}

void write_stream(uint24 v, ureg32 *idx)
{
  CBM_single_write(*idx, v) ;
  return ;
}

filter_core()
{
  ....
  /* Line 0 (odd=0) */
  for (j = 0; j < w_size; j++) {
     read_stream(&val,&read_index);
     line[0][j] = val ;
  }
  ...
}
```

4. Reverse(?) trend with C-synthesis
Change SW control into Hardware control

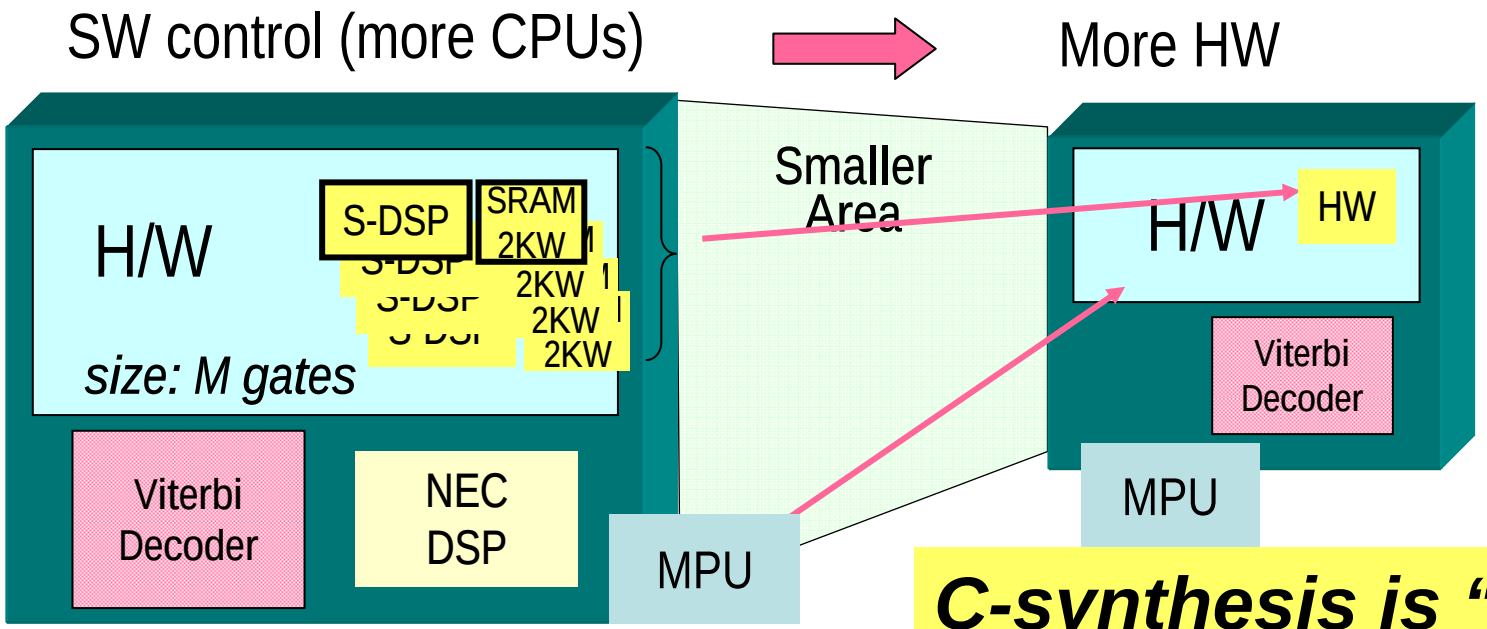
Global trend: more SW,
but still we have minor trend of SW to HW solution

1) After several series of versions
- For Low Power
- For low cost, area, performance

Very complex control can be implemented as a HW

2) Hard Real Time Operation in HW rather than SW
For easiness of design and reliability of RT operation

more Hardware for Power efficiency, Easy to design



C-synthesis is "must"

1)Power(Area) Reduction

-Less custom DSP, SRAM

-> Pure Hardware

(less active power & leakages)

2) SW control into HW control

- Waiting Procedure: S/W into H/W

  => Power Reduction

- Real time constrained S/W into H/W

  => Easy to design , Reliable

Summary

1) Introduced our ASSP approach,
   a few main CPUs + multi custom CPUs + DRPs
2) C-based synthesis affects our architecture
   (without C-based synthesis, 1) does not work well)
   -tasks in main CPU, custom CPU, HW modules, DRP
   could be exchangeable (that actually happens)
3) Our Global trend is “HW to SW” ,
   but “SW to HW” trend is also realized with C-synthesis

☆Followings are all C-synthesis HW modules.
Designer can select realization with similar MonPower
- custom CPU : very flexible & programmable HW
- DRP : flexible and medium performance
- HW modules : hardwired high performance

CyberWorkBench®

Pioneering C-based LSI Design

Email: cwbinfo@mls.necst.nec.co.jp
URL:http://www.cyberworkbench.com/