Silicon Hive

Flexibility Tradeoffs in SoC Design for Low-Cost SDR

Geoff Burns, Director of Applications
Paul Gruijters, Principal Application Engineer
Silicon Hive

Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

Silicon Hive

Objectives of this tutorial

- Highlight issues in designing programmable systems-on-chips that are practical for consumer products
- Show how standard host/accelerator platforms may be extended with c-programmable domain-specific accelerators
- Provide details and examples of SiHive processor generation and programming technology that enable this approach

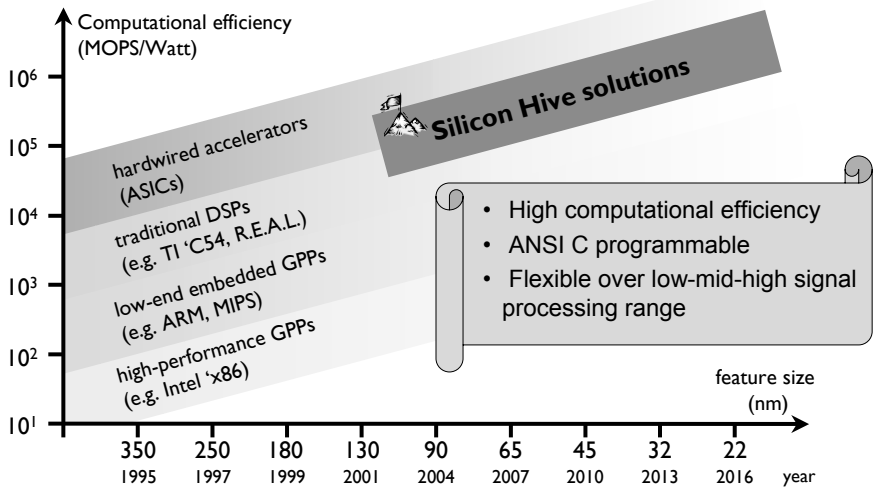
Silicon Hive

Who is Silicon Hive

- Develops and licenses high-performance and programmable DSP accelerators, programming tools, and development systems, that bring full programmability to SoC's while reducing cost and power consumption.
- Specializes accelerators to customer specifications.
- Is a fully-owned business of Philips Electronics, N.V.

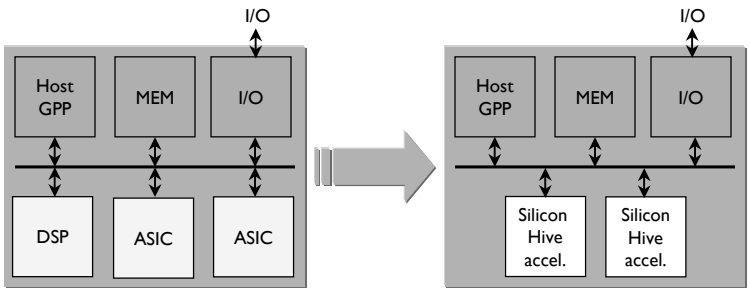
Silicon Hive

Technology positioning



Silicon Hive

Platform SoCs



Standard platform SoC's (e.g. ARM/MIPS & accelerator) provide a familiar starting point, although the accelerators are platform-independent.

Silicon Hive

Application segments

	Connectivity	Media processing
Automotive	Digital Car Radio, Telematics	Driver assistance, Collision avoidance
Communications	2.5G, 3G, CDMA	Image and Video pre/post processing Image and Video encoding/decoding
Data processing	Multi-channel and Multi-antenna WLAN	
Consumer	Ultra-Wide Band, Home gateways	

Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

Silicon Hive

System-on-Chip market

Semiconductor market in 2002 (all figures in USD)

General Purpose

102 billion

Application Specific (ASIC+ASSP)

51 billion

ASIC + ASSP w/o programmable IP blocks	ASIC + ASSP with programmable IP blocks (SoCs)
28 billion	23 billion

In 2007, SoC market will grow to USD 56 billion

Silicon Hive

Drivers for programmability in High-Volume SoC's

- Product development
  - Time to Market
  - NRE
  - Si Mask Costs
  - Discontinuities in flow of design representation (e.g. specification, prototype, ASIC)
- Product maintenance
  - Feature upgradeability across product lifetime
  - Cost-down across product lifetime
  - Compressing market cycles, particularly in high-volume consumer

Silicon Hive

Issues for programmability in High-Volume SoC's

- Applications with extreme real-time requirements relative to capabilities of programmable DSP technology, typically compel the design of ASIC implementations, or ASIC accelerators in partitioned designs.

- Silicon Cost
  - A general purpose processor, with sophisticated control, and which is designed for all cases, introduces area overheads that cannot be easily amortized for these applications.
- Power
  - Increased silicon area, combined with an increased clock frequency, drive increased power consumption.
  - $CV^2f + I_{leakage}$

Silicon Hive

Efficient Programmable SoC—how?

- Already in practice:
  - Partitioning—control versus DSP inner loop, as per host/accelerator platform.
  - Architecture & VLSI techniques for power control (clock gating, sleep modes, …)
- But also:
  - Use multiple c-programmable accelerators, judiciously specialized for requirements of the domain, dataflow, function, precision, etc.
- Using Silicon Hive Key enablers:

  Automatic processor generation

  Automatically-generated C-based spatial compilers

Silicon Hive

Other Implications

- Development effort shifts to DSP SW development on pre-optimized cores
- A more continuous flow for product development and maintenance is possible
  - Prototyping
  - Feature addition
  - Redesign for cost-down

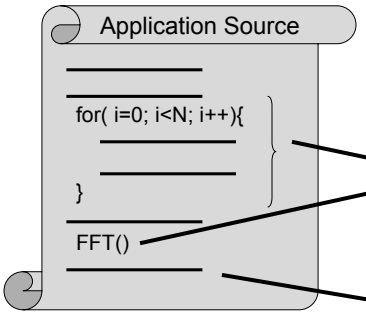
Silicon Hive

With these degrees of freedom, design issues include

- System partitioning, and accelerator specification
- Tradeoff between flexibility and efficiency for each accelerator

Silicon Hive

System Partitioning



In many important SoC applications source can be partitioned between loops or functions which occupy

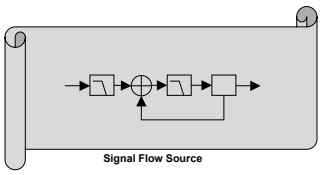
majority of execution time
minority of instructions

minority of execution time
majority of instructions

Partitioning can be achieved using profiling tools or inspection.

Silicon Hive

System Partitioning



In the SDR physical layer, such source partitioning is clear from the signal flow graph

Silicon Hive

Accelerator Domain Specialization

The primary choice is between stream and block Acceleration

Further specialization within these categories can be pursued depending upon the specific domain

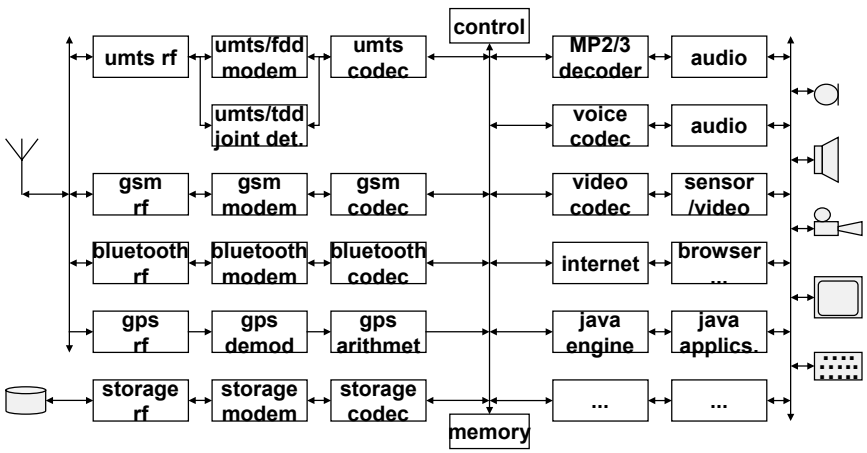
- •CDMA
- •WLAN
- •Video

…or goals for the SoC

- •Prototyping
- •Multi-standard combination of similar standards
- •Silicon area reduction

Silicon Hive

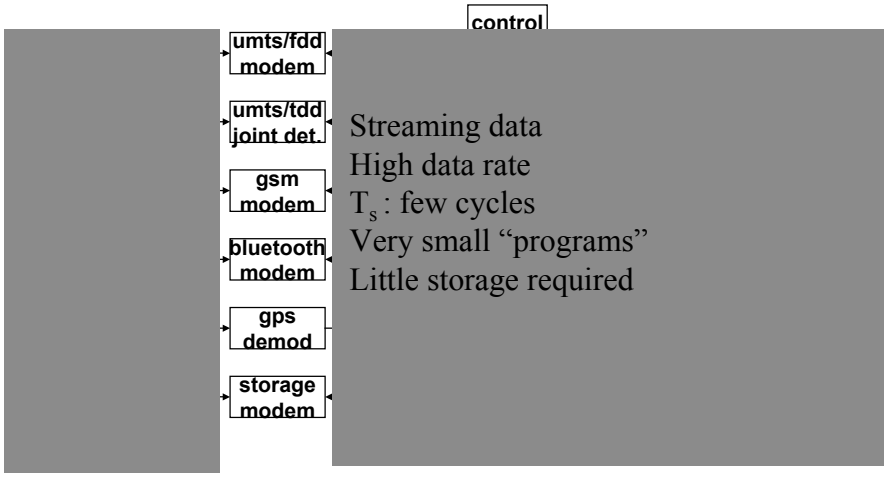
Accelerator Domain Specialization



Consider a Hypothetical Universal Terminal

Silicon Hive

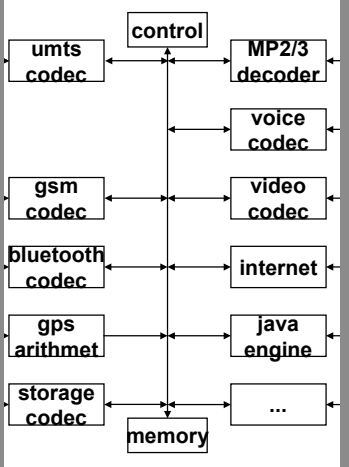
Stream processing: $t_{clk} \sim T_s$



Silicon Hive

Block processing: $t_{clk} << T_s$

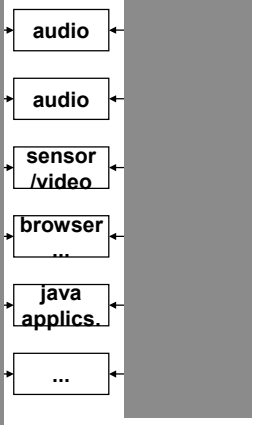
Block-based data
Modest data rates
T_s : many cycles
Significant program sizes
Storage required



Silicon Hive

Stream processing: $t_{clk} \sim T_s$

Streaming data
High data rate
T_s : few cycles
Very small "programs"
Little storage required



Silicon Hive

Accelerator Domain Specialization

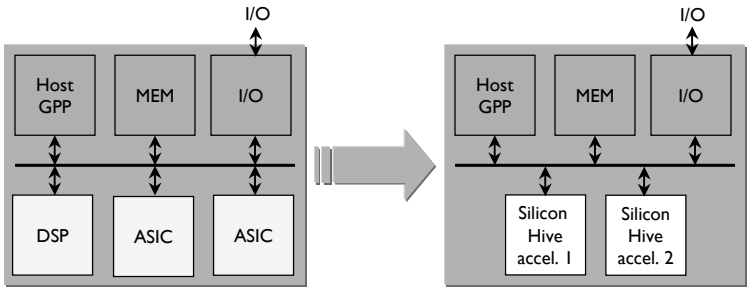
The primary choice is between stream and block Acceleration

Further specialization within these categories can be pursued depending upon the specific domain
- •CDMA
- •WLAN
- •Video

…or goals for the SoC
- •Prototyping
- •Multi-standard combination of similar standards
- •Silicon area reduction

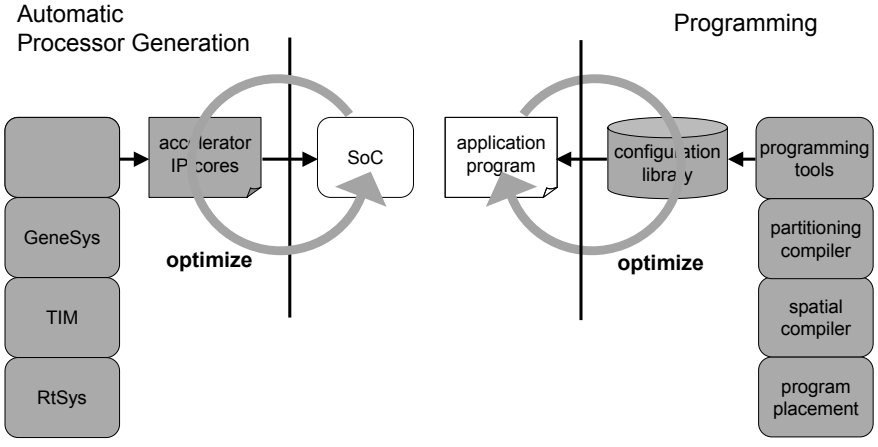
Silicon Hive

Platform SoCs



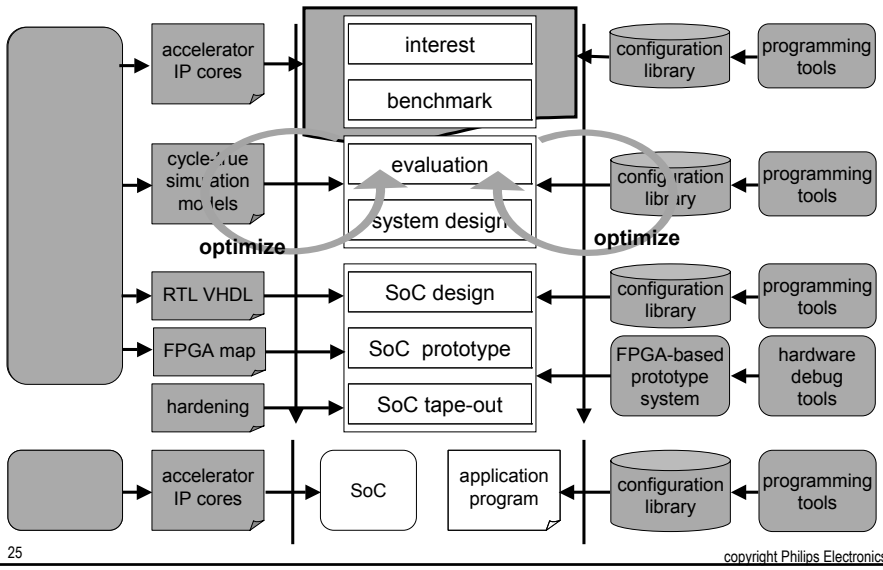
Silicon Hive

What technology enables such SoCs?



Silicon Hive

the design cycle



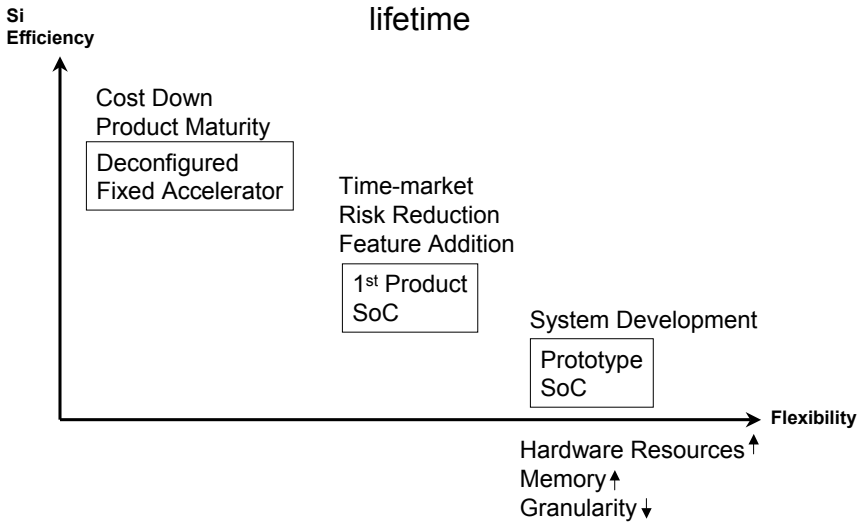
Silicon Hive

What about product maintenance over its life cycle?

- An SoC may be redesigned through several phases during the product lifetime, each with associated NRE.
  - Reference prototype (FPGA,DSP, or ASICs)
  - Product
  - Cost-down
- A c-programmable accelerator platform, along with the technology to deconfigure accelerators, enables products to pass through product lifetime stages quickly at reduced NRE.

Silicon Hive

Role of [re][de]configurable cores throughout product lifetime



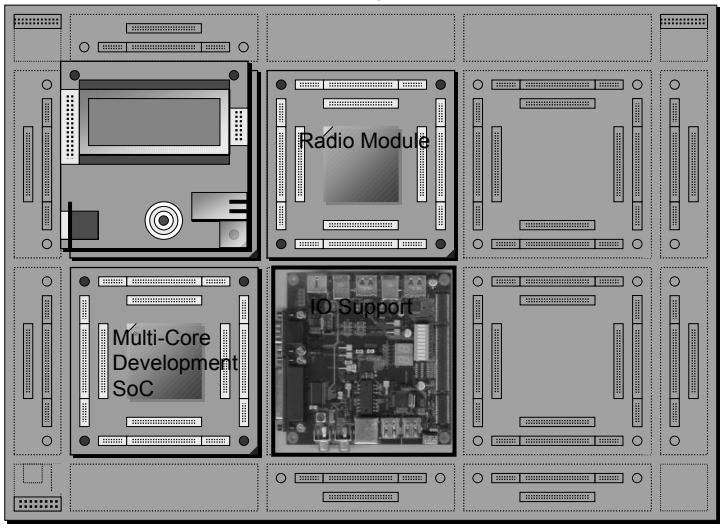
Silicon Hive

Development System

- Based Xplorer Advanced Prototyping System developed and sold by Philips Electronics
- Supported by SiHive for Real-Time System prototyping
- Extensive connectivity allows multi-layer bus configurations (AHB)
- Allows integration various IP-cores, including analog

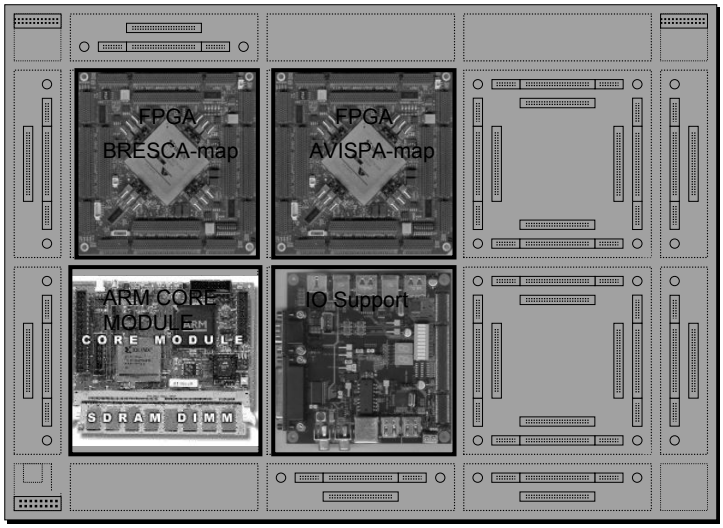
Silicon Hive

Development System Concept



Silicon Hive

Development System -- Current



Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

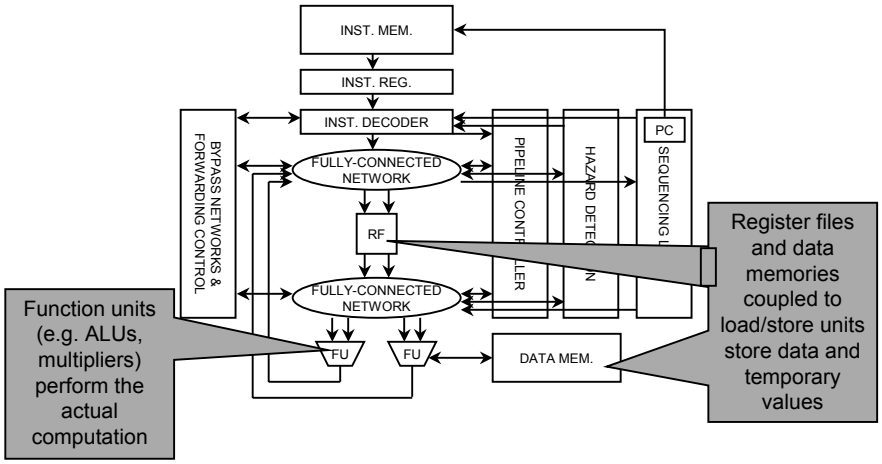
Silicon Hive

Architecture Overview

- The architecture is actually a template, from which to express a family of accelerators
- Requirements
  - Minimize overheads
  - c-based compiler target
  - Shift scheduling burdens to c-compiler
  - Both processor and tool-set conducive to automatic generation
  - Sufficiently general to express multiple tradeoffs for [flexibility,parallelism, dataflow type, …]

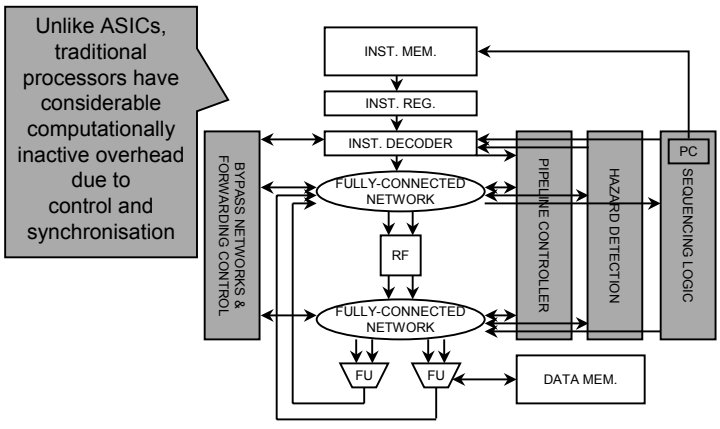
Silicon Hive

Traditional processor architecture



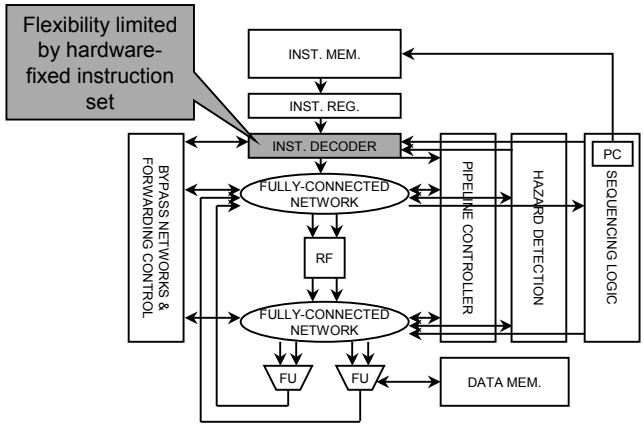
Silicon Hive

Traditional processor architecture



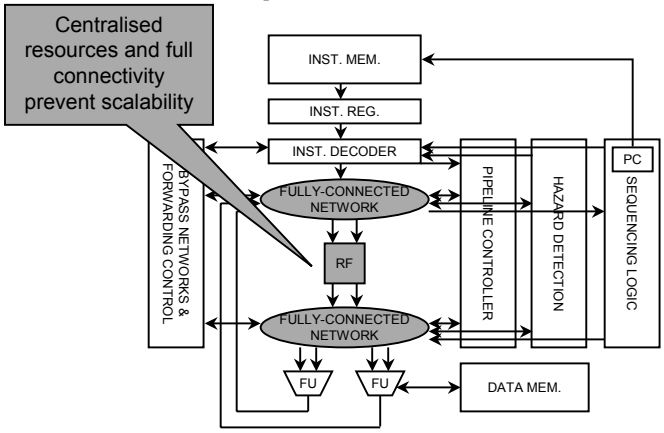
Silicon Hive

Traditional processor architecture



Silicon Hive

Traditional processor architecture



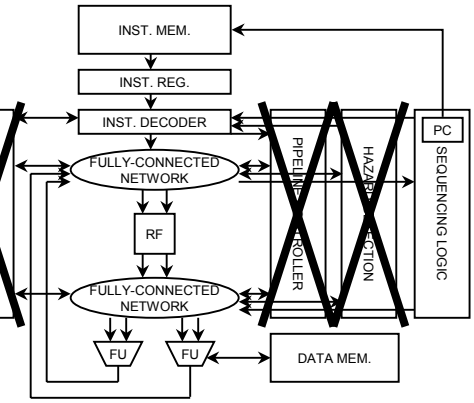
Silicon Hive

ULIW architecture template

Dramatically reduce control overhead, expose all pipeline management to the instruction set, move complexity to the compiler.

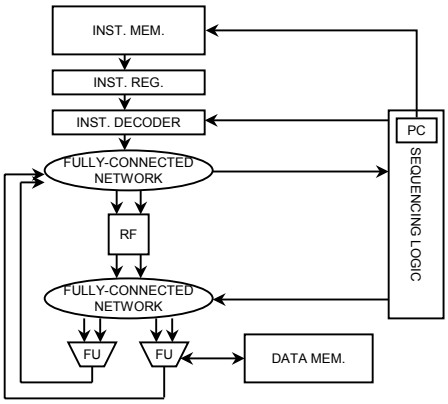
→

Compiler must explicitly schedule all pipeline stages!



Silicon Hive

ULIW architecture template



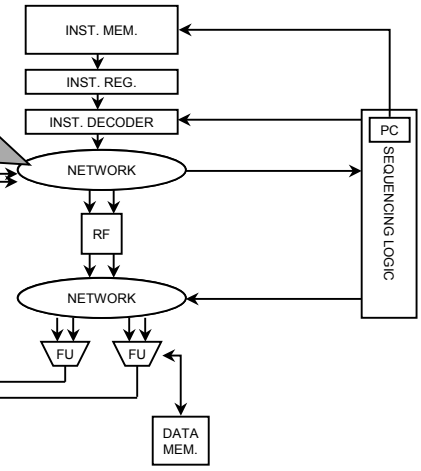
Silicon Hive

ULIW architecture template

Remove full connectivity. Use point-to-point, partially connected networks.

→

Things become more scalable…



Silicon Hive

ULIW architecture template

Partition and distribute all resources

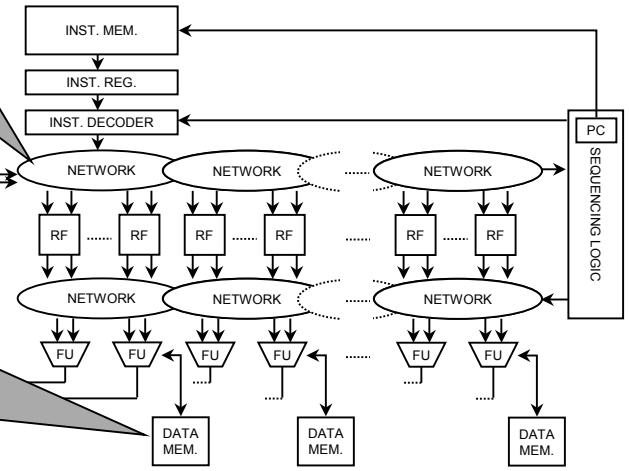
→

Massively parallel architecture

Use many local data memories to improve locality of reference, increase bandwidth, and reduce bus traffic

→

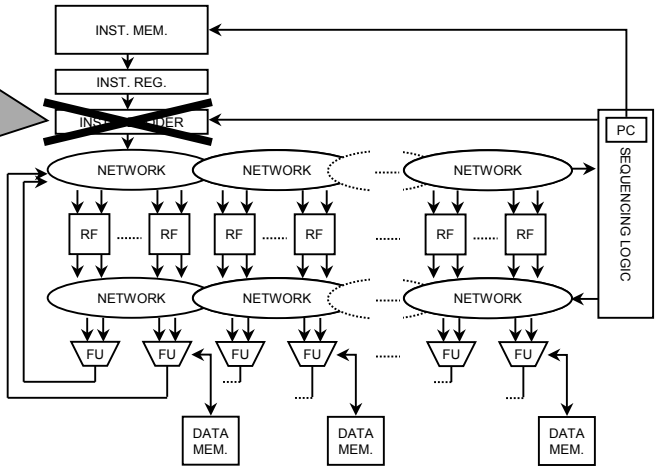
Higher performance and lower power



Silicon Hive

ULIW architecture template

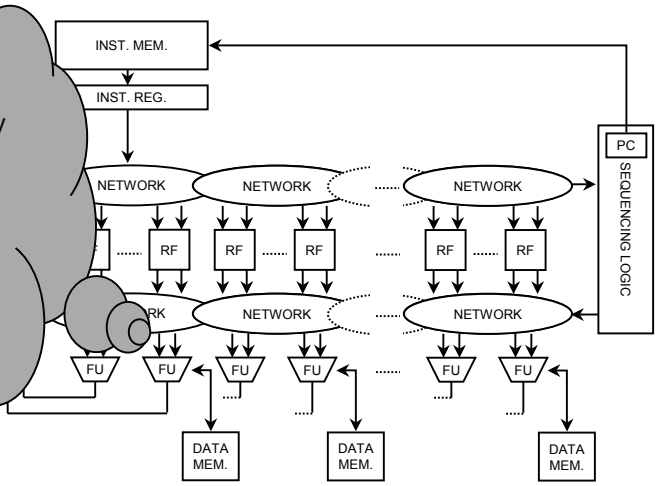
Expose all control points in the datapath to the instructions. Add flexibility to do all combinations the datapath can support (and there are many!)
→
ultra long instructions
→
ULIW

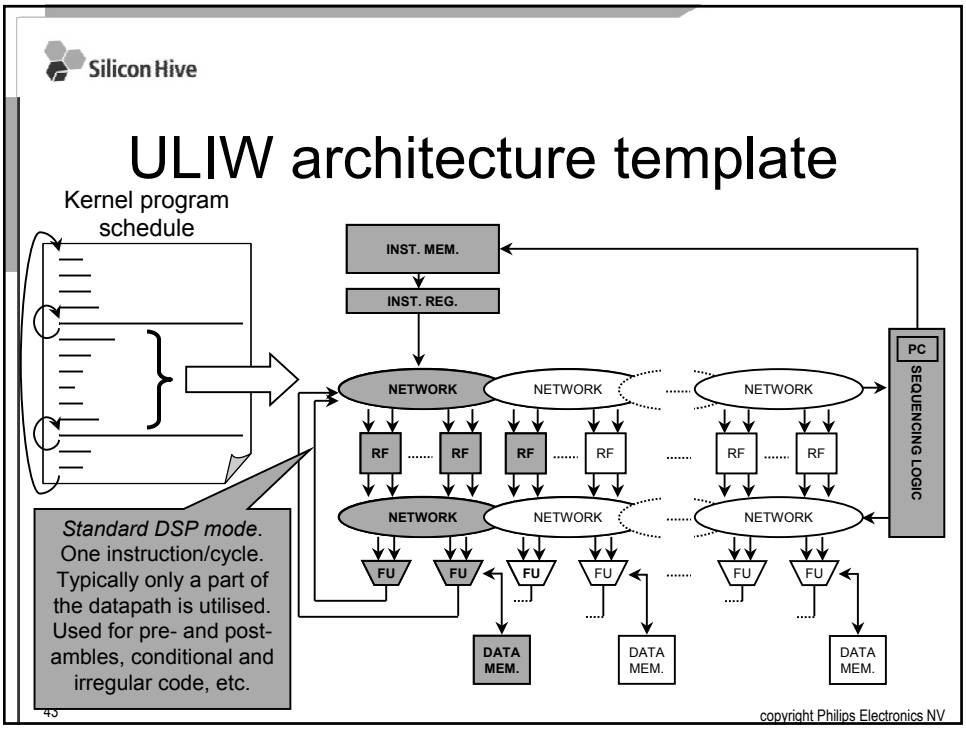


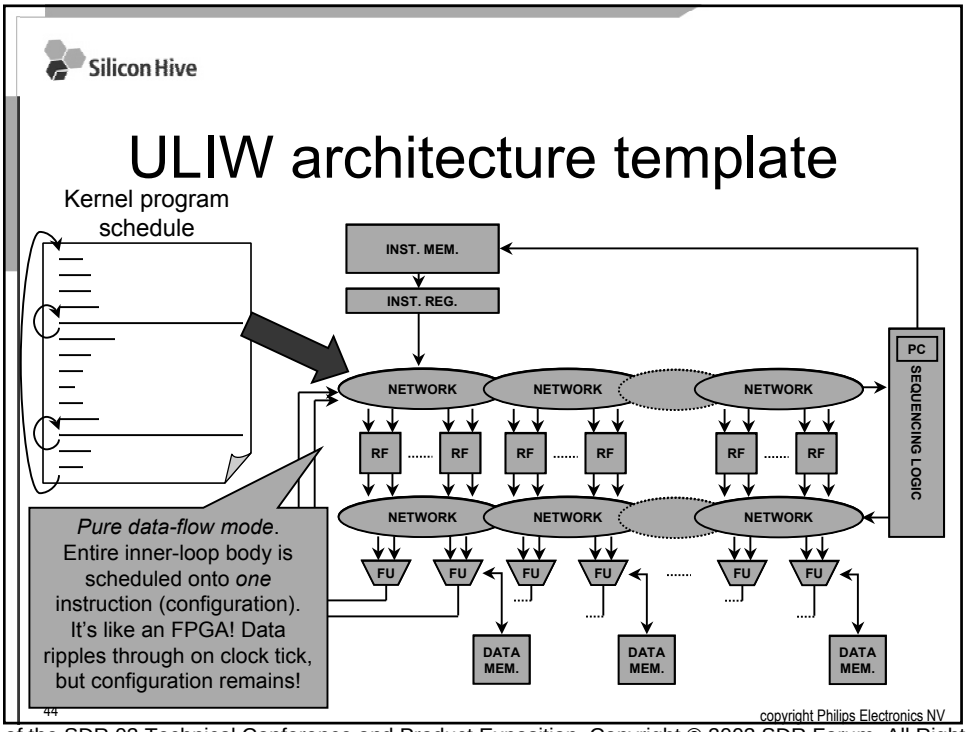
Silicon Hive

ULIW architecture template

High locality of reference & regularity
+
Massive parallelism
+
Very low control overhead
=
Very high computational efficiency (MOPS/W)!!!

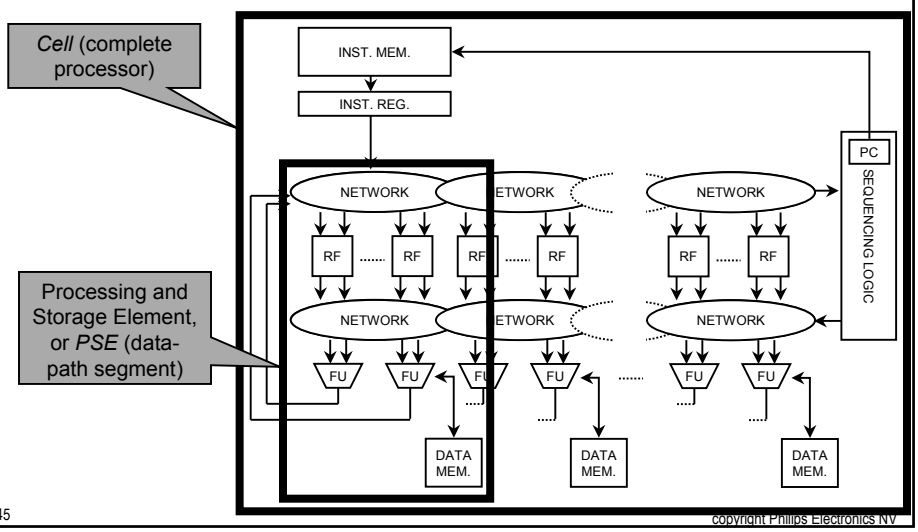






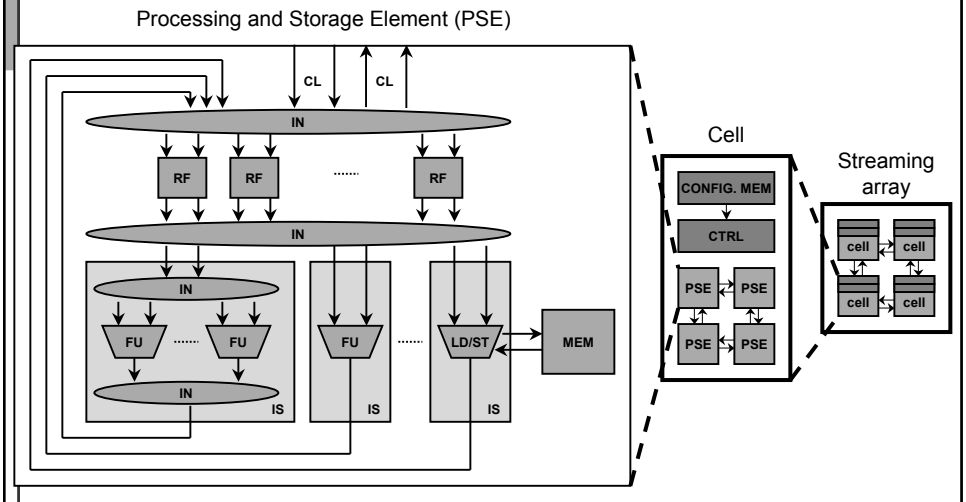
Silicon Hive

Architecture components



Silicon Hive

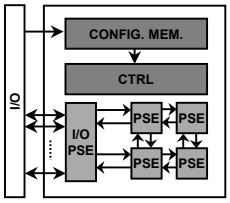
Architecture dimensions



Silicon Hive

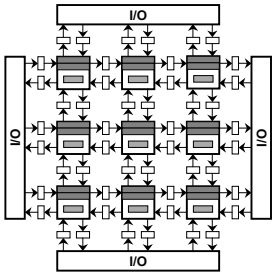
Different instances

Instances are *not* hand-written, but are generated from internal design methodology



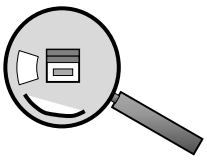
AVISPA block accelerator
(ONE CELL, MULTIPLE PSEs)

High-performance & efficiency for data framed data in memory



BRESCA stream accelerator
(MULTIPLE CELLS, ONE PSE EACH)

Streaming data, very high/dynamic rates (e.g. IF front-end filtering)



MOUSTIQUE block accelerator
(ONE CELL, ONE PSE)

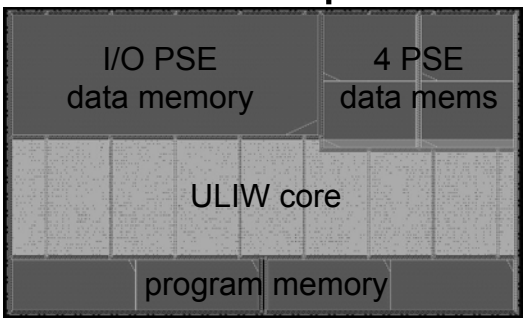
Framed data in memory for highly cost-sensitive apps. "Smaller than ASIC"

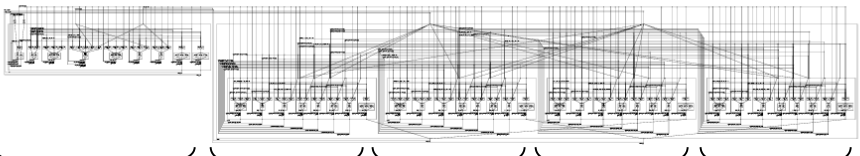
Silicon Hive

AVISPA instance example

accelerator for software defined radio

- 16 bit datapath
- 41 issue slots, 75 function units
- 95 register files
- 5 dual port data mems
- 115K gates excl. memories
- 150 MHz (standard cell, CMOS12, WCMIL)
- 0.85 mW/MHz

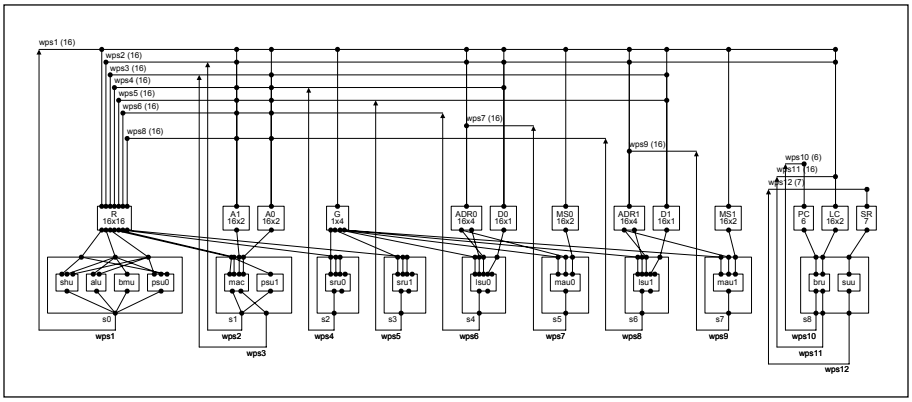




control & I/O PSE **four identical arithmetic Processing & Storage Elements (PSE)**
(contain a.o. ALU, MAC, barrel shifter, AGU, 2 LSUs, dual port RAM)

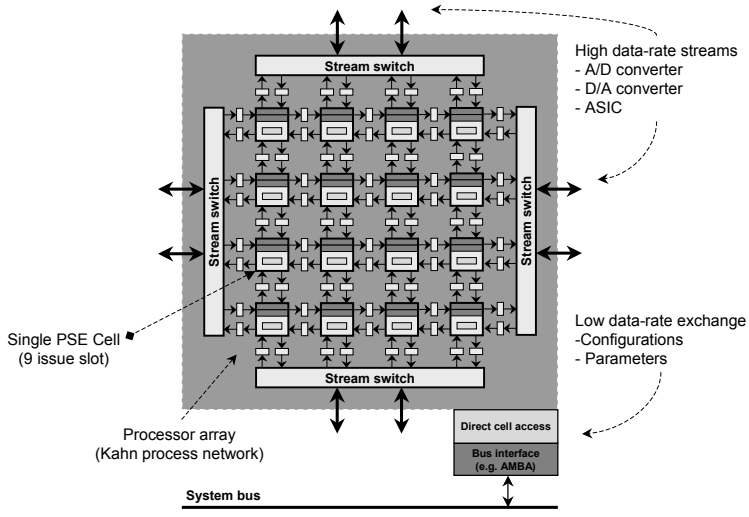
Silicon Hive

Hive core: Bresca



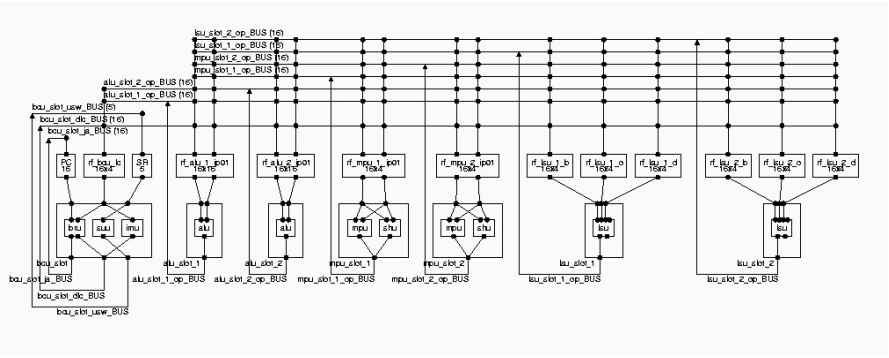
Silicon Hive

Streaming Array



Silicon Hive

Moustique Instance Example



- 16 bit datapath
- 7 issue slots, 1 function units
- 78 register files

Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

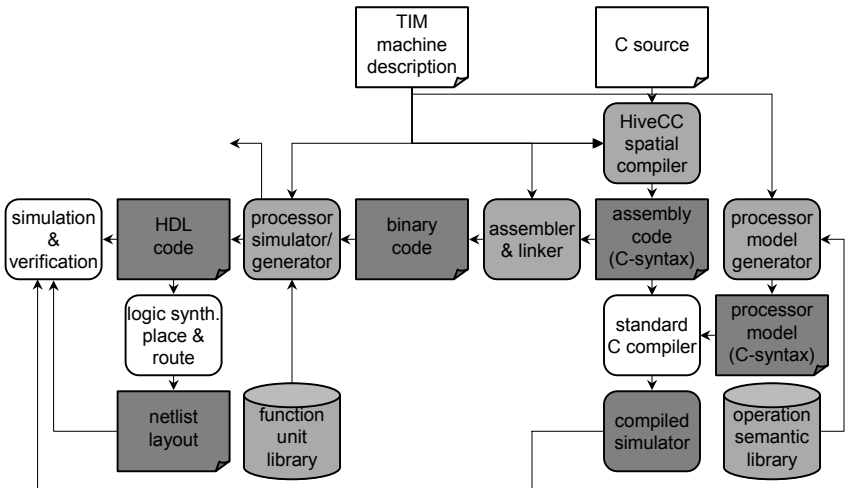
Silicon Hive

Processor Generation Methodology

- Concise description (TIM) describes instance of architecture
- Processor RTL, simulator, compiler generated consistently from same description
- Allows SiHive to quickly customize accelerators at significantly reduced risk of bug introduction

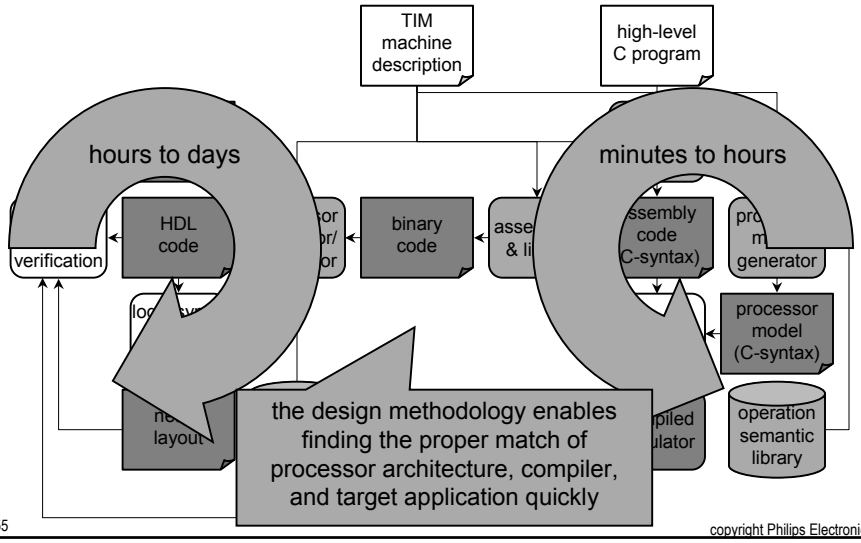
Silicon Hive

Processor generation methodology



Silicon Hive

Processor generation methodology



Silicon Hive

TIM description fragment

```
/* port types */
Port Port32 { Width = 32; };
...

/* register file types */
RF RF32r8 (Port32 ip0) -> (Port32 op0) {
 Width = 32; Latency = 1; Capacity = 8;
};
...

/* operation semantics */
OP passh (wsigned a) -> (wsigned b) {
 SEM b = a << (width_b-width_a);
};
...

/* function unit types */
FU psu_passonly_imm (Port32 ip) -> (Port32 op) {
 imm: op = pass(Immediate(ip,[-32768, .., 32767]));
 immh:op = passh(Immediate(ip,[0, .., 65535]));
 pass:op = pass(ip);
};
...

/* issue slot types */
IS slot2 (Port32 ip0, ip1, ip2) -> (Port32 op) {
 lsu lsu(ip0, ip1, ip2); op = lsu.op;
 psu_passonly_imm_1 psu(ip0); op = psu.op;
};
...
```

```
/* cell description */
#define ALL s1.op, s2.op, s3.op, s4.op, s1.dlc

Machine moustique {
 program_counter PC (s1.ja);
 status_register SR (s1.usw);

 RF32r8 rf1 ({ALL});
 RF32r8 rf2 ({ALL});
 RF32r8 rf3 ({ALL});
 RF32r8 rf4 ({ALL});
 RF32r8 rf5 ({ALL});
 RF32r8 rf6 ({ALL});
 RF32r8 rf7 ({ALL});
 RF64r2 rf64  ({s3.op64, s4.op64});
 RF64r4 rf64b ({s3.op64, s4.op64});

 slot1 s1 (rf1.op0, rf2.op0, rf5.op0, PC.rp, SR.rp);
 slot2 s2 (rf3.op0, rf4.op0, rf5.op0);
 slot3 s3 (rf3.op0, rf4.op0, rf5.op0, rf64.op0);
 slot4 s4 (rf6.op0, rf7.op0, rf64b.op0);
}
```

Silicon Hive

TIM description fragment

```
/* port types */
Port Port32 { Width = 32; };
...

/* register file types */
RF RF32r8 (Port32 ip0) -> (Port32 op0) {
 Width = 32; Latency = 1; Capacity = 8;
};
...

/* operation semantics */
OP passh (wsigned a) -> (wsigned b) {
 SEM b = a << (width_b-width_a);
};
...

/* function unit types */
FU psu_passonly_imm (Port32 ip) -> (Port32 op) {
 imm: op = pass(Immediate(ip,[-32768, .., 32767]));
 immh:op = passh(Immediate(ip,[0, .., 65535]));
 pass:op = pass(ip);
};
...

/* issue slot types */
IS slot2 (Port32 ip0, ip1, ip2) -> (Port32 op) {
 lsu lsu(ip0, ip1, ip2); op = lsu.op;
 psu_passonly_imm_1 psu(ip0); op = psu.op;
};
...
```

```
/* cell description */
#define ALL s1.op, s2.op, s3.op, s4.op, s1.dlc

Machine moustique {
 program_counter PC (s1.ja);
 status_register SR (s1.usw);

 RF32r8 rf1 ({ALL});
 RF32r8 rf2 ({ALL});
 RF32r8 rf3 ({ALL});
 RF32r8 rf4 ({ALL});
 RF32r8 rf5 ({ALL});
 RF32r8 rf6 ({ALL});
 RF32r8 rf7 ({ALL});
 RF64r2 rf64  ({s3.op64, s4.op64});
 RF64r4 rf64b ({s3.op64, s4.op64});

 slot1 s1 (rf1.op0, rf2.op0, rf5.op0, PC.rp, SR.rp);
 slot2 s2 (rf3.op0, rf4.op0, rf5.op0);
 slot3 s3 (rf3.op0, rf4.op0, rf5.op0, rf64.op0);
 slot4 s4 (rf6.op0, rf7.op0, rf64b.op0);
}
```

Silicon Hive

TIM description fragment

/* port types */
Port Port32 { Width = 32; };
...

/* register file types */
RF RF32r8 (Port32 ip0) -> (Port32 op0) {
 Width = 32; Latency = 1; Capacity = 8;
};
...

/* operation semantics */
OP passh (wsigned a) -> (wsigned b) {
 SEM b = a << (width_b-width_a);
};
...

/* function unit types */
FU psu_passonly_imm (Port32 ip) -> (Port32 op) {
 imm: op = pass(Immediate(ip,[-32768, .., 32767]));
 immh:op = passh(Immediate(ip,[0, .., 65535]));
 pass:op = pass(ip);
};
...

/* issue slot types */
IS slot2 (Port32 ip0, ip1, ip2) -> (Port32 op) {
 lsu lsu(ip0, ip1, ip2); op = lsu.op;
 psu_passonly_imm_1 psu(ip0); op = psu.op;
};
...

/* cell description */
#define ALL s1.op, s2.op, s3.op, s4.op, s1.dlc

Machine moustique {
 program_counter PC (s1.ja);
 status_register SR (s1.usw);

 RF32r8 rf1 ({ALL});
 RF32r8 rf2 ({ALL});
 RF32r8 rf3 ({ALL});
 RF32r8 rf4 ({ALL});
 RF32r8 rf5 ({ALL});
 RF32r8 rf6 ({ALL});
 RF32r8 rf7 ({ALL});
 RF64r2 rf64 ({s3.op64, s4.op64});
 RF64r4 rf64b ({s3.op64, s4.op64});

 slot1 s1 (rf1.op0, rf2.op0, rf5.op0, PC.rp, SR.rp);
 slot2 s2 (rf3.op0, rf4.op0, rf5.op0);
 slot3 s3 (rf3.op0, rf4.op0, rf5.op0, rf64.op0);
 slot4 s4 (rf6.op0, rf7.op0, rf64b.op0);
}

Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

Silicon Hive

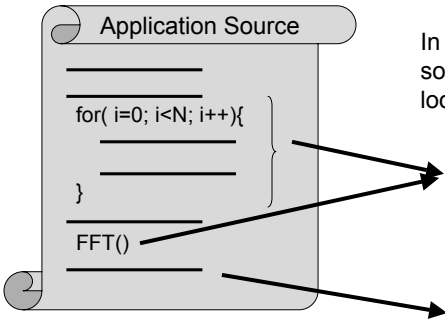
Programming tools

- ***Partitioning compiler*** analyses any ANSI C program for:
  - execution count
  - data traffic
  - instruction level parallelism (ILP) potential

  aids in obtaining code to be accelerated
- ***Spatial compiler HiveCC*** generates code for ***one cell***
  - Aggressive operation scheduling and resource allocation extract ILP
    - Schedules inner-loops onto *one* instruction (configuration)
  - Turns the constraints (i.e. lack of full connectivity) of a scalable architecture to an advantage, as opposed to a problem!
    - Constraint analysis reduces scheduling space
    - Makes it easier to find optimal solutions
- The ***array placement tool SHAPE*** assigns pre-compiled programs to cells in a ***streaming array***

Silicon Hive

System Partitioning



In many important SoC applications source can be partitioned between loops or functions which occupy

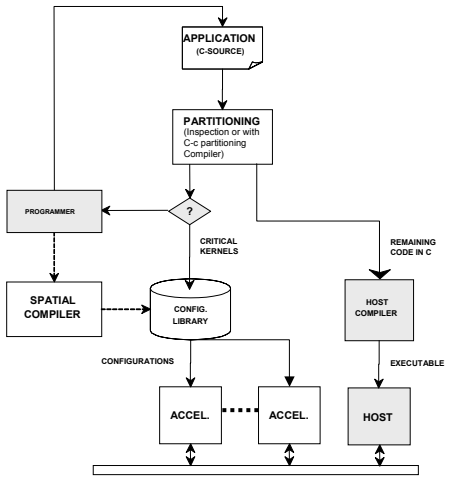
majority of execution time
minority of instructions

minority of execution time
majority of instructions

Partitioning can be achieved using profiling tools or inspection.

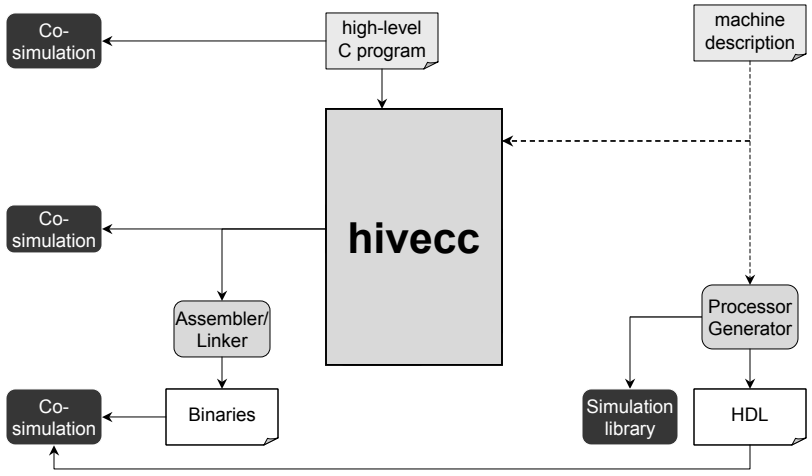
Silicon Hive

Partitioning flow



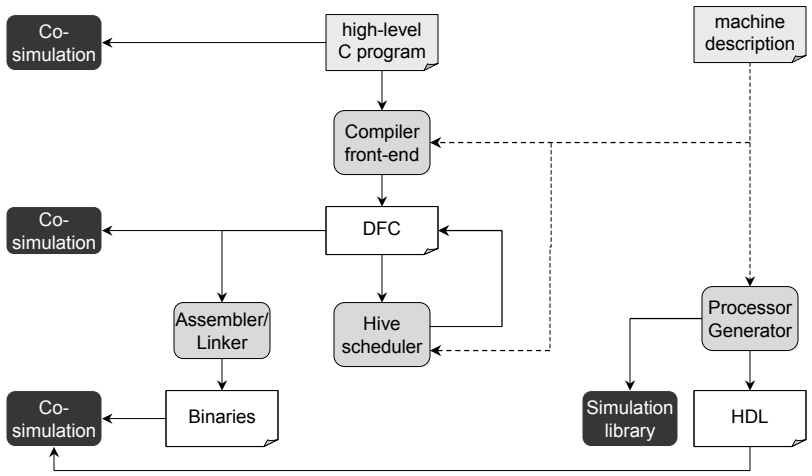
Silicon Hive

The “hivecc” spatial compiler



Silicon Hive

The “hivecc” spatial compiler



Silicon Hive

DFC

- Data Flow C
- Subset of C
- Represents
  - Input program
  - Partially scheduled code
  - Fully scheduled code
- Simulation by compilation (cycle accurate)

Silicon Hive

C to DFC compilation

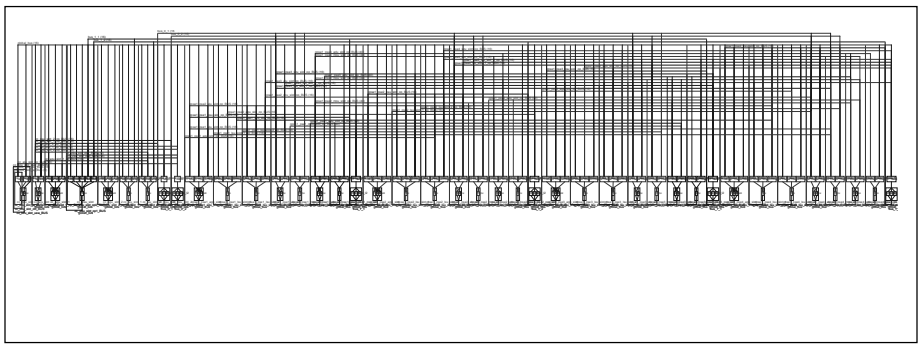
```
k1 = k2 = k5 = 0;
for (loopcnt1 = 1023; loopcnt1; loopcnt1--) {
    twidre = base9[k5];
    twidim = base10[k5];

    v1re = base1[k1];
    v1im = base2[k1];
    v2re = base3[k1];
    v2im = base4[k1];

    sum1re = ROUND(v1re+v2re);
    sum1im = ROUND(v1im+v2im);
    sum2re = ROUND(v1re-v2re);
    sum2im = ROUND(v1im-v2im);

    base5[k2] = sum1re;
    base6[k2] = sum1im;

    CMUL_SCALE (1, scale, sum2re, sum2im, twidre,
        twidim, tmp5re, tmp5im);

    base7[k2] = tmp5re;
    base8[k2] = tmp5im;

    k1++;
    k5 = k1 & mask;
    k2 += 2;
}
```

```
do BLOCK (loop1_1) {
  ld16o (cc_lsu1_0, base9_1, k5_1, twidre_1);
  ld16o (cc_lsu1_1, base10_1, k5_1, twidim_1);
  pass0 (Any, twidre_1, twidre_p1);
  pass0 (Any, twidim_1, twidim_p1);
  ld_1_0: ld16o (qpse1_lsu1_0, base1_1, k1_1, v1re_1);
  ld_2_0: ld16o (qpse1_lsu2_0, base2_1, k1_1, v1im_1);
  ld_1_1: ld16o (qpse1_lsu1_1, base3_1, k1_1, v2re_1);
  ld_2_1: ld16o (qpse1_lsu2_1, base4_1, k1_1, v2im_1);
  addshift (Any, v1re_1, v2re_1, sum1re_1);
  addshift (Any, v1im_1, v2im_1, sum1im_1);
  subshift (Any, v1re_1, v2re_1, sum2re_1);
  subshift (Any, v1im_1, v2im_1, sum2im_1);
  st_3_0: st16o (qpse1_lsu3_0, base5_1, k2_1, sum1re_1);
  st_4_0: st16o (qpse1_lsu4_0, base6_1, k2_1, sum1im_1);

  CMUL_SCALE (1, scale_1, sum2re_1, sum2im_1, twidre_p1,
                twidim_p1, tmp5re_1, tmp5im_1);

  st_3_1: st16o (qpse1_lsu3_1, base7_1, k3_1, tmp5re_1);
  st_4_1: st16o (qpse1_lsu4_1, base8_1, k3_1, tmp5im_1);
  inc (Any, k1_1, k1_1);
  inc (Any, k4_1, k4_1);
  and (Any, k4_1, mask, k5_1);
  add (Any, k2_1, const2_1, k2_1);
  add (Any, k3_1, const2_1, k3_1);
  cjmptdec (Any, loop1_1, loopcnt1_1,
            REGALLOC(PC,0,Any,NOBUS), loopcnt1_1);
} while (loopcnt1_1!=-1);
```

Schedule, dfc

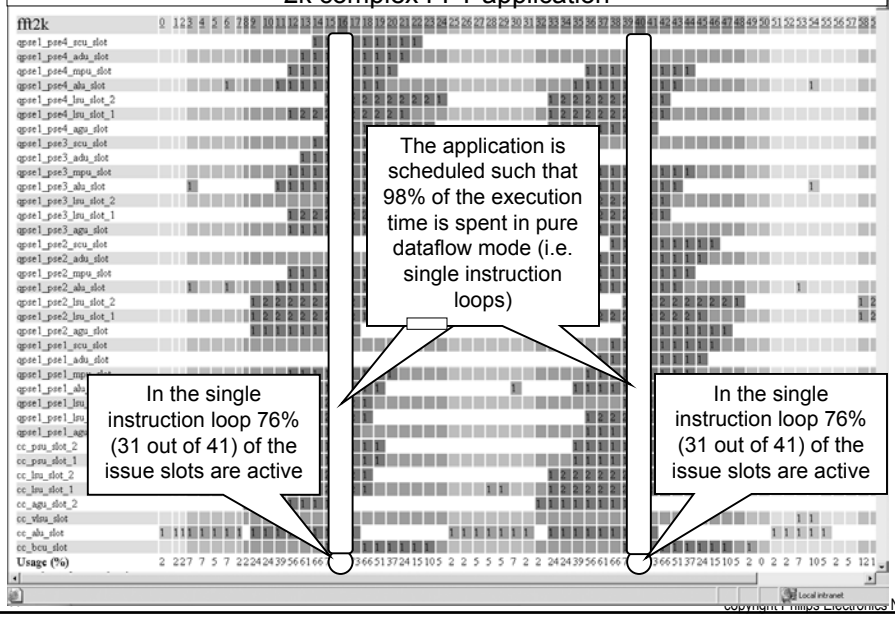
```
CYCLE(16,
    st16o_PIPE_1_1 (qpse1_pse3_lsu_slot_2_lsu);
    st16o_PIPE_1_1 (qpse1_pse4_lsu_slot_2_lsu);
    st16o_PIPE_0_1 (qpse1_pse3_lsu_slot_2_lsu, IN(b4re_1_e61_e289_e289,0), JOIN(IN(o4_1_e62_e291_e290,1),BACK(o4 1 e62 e290 e291,1)), JOIN(IN(
    st16o_PIPE_0_1 (qpse1_pse4_lsu_slot_2_lsu, IN(b4im_1_e65_e294_e294,0), JOIN(IN(o4_1_e66_e296_e295,1),BACK(o4 1 e66 e295 e296,1)), JOIN(IN(
    add            (qpse1_pse3_agu_slot_mau,   JOIN(IN(o4_1_e82_e300_e299,0),BACK(o4_1_e82_e299_e300,0)), IN(const2 e84 e301 e301,1), SPLIT(SP
    cjmptdec       (cc_bcu_slot_bru, 21,       JOIN(IN(loopcnt1_1_e85_e303_e302,2),BACK(loopcnt1_1_e85 e302 e303,2)), OUT(pc,0), SPLIT(SPLIT(E
    asrrnd40       (qpse1_pse4_scu_slot_scu,   JOIN(IN(ca1_1_e57_e305_e304,0),BACK(ca1_1_e57_e304_e305,0)), IN(scale 1 e58 e306 e306,1), SPLIT
    asrrnd40       (qpse1_pse2_scu_slot_scu,   JOIN(IN(ca2_1_e59_e308_e307,0),BACK(ca2_1_e59_e307_e308,0)), IN(scale 1 e60 e309 e309,1), SPLIT
    st16o_PIPE_1_1 (qpse1_pse3_lsu_slot_1_lsu);
    st16o_PIPE_1_1 (qpse1_pse4_lsu_slot_1_lsu);
    rsub40         (qpse1_pse4_adu_slot_adu,   JOIN(IN(cm2_1_e53_e311_e310,0),BACK(cm2_1_e53_e310_e311,0)), JOIN(IN(cm1 1 e54 e313 e312,1),BAC
    add40          (qpse1_pse2_adu_slot_adu,   JOIN(IN(cm3_1_e55_e315_e314,0),BACK(cm3_1_e55_e314_e315,0)), JOIN(IN(cm4 1 e56 e317 e316,1),BAC
    st16o_PIPE_0_1 (qpse1_pse3_lsu_slot_1_lsu, IN(b3re_1_e37_e318_e318,0), JOIN(IN(o3_1_e38_e320_e319,1),BACK(o3 1 e38 e319 e320,1)), JOIN(IN(
    st16o_PIPE_0_1 (qpse1_pse4_lsu_slot_1_lsu, IN(b3im_1_e41_e323_e323,0), JOIN(IN(o3_1_e42_e325_e324,1),BACK(o3 1 e42 e324 e325,1)), JOIN(IN(
    smul40         (qpse1_pse4_mpu_slot_mpu,   JOIN(IN(d4re_1_e45_e329_e328,0),BACK(d4re_1_e45_e328_e329,0)), JOIN(IN(twidre 1 e46 e331 e330,1
    smul40         (qpse1_pse3_mpu_slot_mpu,   JOIN(IN(d4im_1_e47_e333_e332,0),BACK(d4im_1_e47_e332_e333,0)), JOIN(IN(twidim 1 e48 e335 e334,1
    smul40         (qpse1_pse1_mpu_slot_mpu,   JOIN(IN(d4re_1_e49_e337_e336,0),BACK(d4re_1_e49_e336_e337,0)), JOIN(IN(twidim 1 e50 e339 e338,1
    smul40         (qpse1_pse2_mpu_slot_mpu,   JOIN(IN(d4im_1_e51_e341_e340,0),BACK(d4im_1_e51_e340_e341,0)), JOIN(IN(twidre 1 e52 e343 e342,1
    add            (qpse1_pse4_agu_slot_mau,   JOIN(IN(o3_1_e79_e345_e344,0),BACK(o3_1_e79_e344_e345,0)), IN(const2 e81 e346 e346,1), SPLIT(SP
    pass           (cc_psu_slot_2_psu_l0,      JOIN(IN(twidre_1_e15_e348_e347,0),BACK(twidre_1_e15_e347_e348,0)), SPLIT(SPLIT(SPLIT(EXIT(twidr
    pass           (cc_psu_slot_1_psu_l0,      JOIN(IN(twidim_1_e16_e350_e349,0),BACK(twidim_1_e16_e349_e350,0)), SPLIT(SPLIT(SPLIT(EXIT(twidi
    add            (qpse1_pse1_alu_slot_alu,   JOIN(IN(d1re_1_e29_e352_e351,0),BACK(d1re_1_e29_e351_e352,0)), JOIN(IN(d2re 1 e30 e354 e353,1),
    add            (qpse1_pse4_alu_slot_alu,   JOIN(IN(d1im_1_e31_e356_e355,0),BACK(d1im_1_e31_e355_e356,0)), JOIN(IN(d2im 1 e32 e358 e357,1),
    sub            (qpse1_pse3_alu_slot_alu,   JOIN(IN(d1re_1_e33_e360_e359,0),BACK(d1re_1_e33_e359_e360,0)), JOIN(IN(d2re 1 e34 e362 e361,1),
    sub            (qpse1_pse2_alu_slot_alu,   JOIN(IN(d1im_1_e35_e364_e363,0),BACK(d1im_1_e35_e363_e364,0)), JOIN(IN(d2im 1 e36 e366 e365,1),
    ld16o_PIPE_1_1 (cc_lsu_slot_1_lsu,         SPLIT(EXIT(twidre_1_e15_e347_e348,0),EXIT(twidre_1_e15_e469_e469,0)));
    ld16o_PIPE_1_1 (cc_lsu_slot_2_lsu,         SPLIT(EXIT(twidim_1_e16_e349_e350,0),EXIT(twidim_1_e16_e470_e470,0)));
    ld16o_PIPE_1_1 (qpse1_pse1_lsu_slot_1_lsu, SPLIT(SPLIT(SPLIT(EXIT(d1re_1_e29_e351_e352,0),EXIT(d1re_1_e29 e471 e471,0)),EXIT(d1re 1 e33 e3
    ld16o_PIPE_1_1 (qpse1_pse2_lsu_slot_1_lsu, SPLIT(SPLIT(SPLIT(EXIT(d1im_1_e31_e355_e356,0),EXIT(d1im_1_e31 e473 e473,0)),EXIT(d1im 1 e35 e3
    ld16o_PIPE_1_1 (qpse1_pse1_lsu_slot_2_lsu, SPLIT(SPLIT(SPLIT(EXIT(d2re_1_e30_e353_e354,0),EXIT(d2re_1_e30 e472 e472,0)),EXIT(d2re 1 e34 e3
    ld16o_PIPE_1_1 (qpse1_pse2_lsu_slot_2_lsu, SPLIT(SPLIT(SPLIT(EXIT(d2im_1_e32_e357_e358,0),EXIT(d2im_1_e32 e474 e474,0)),EXIT(d2im 1 e36 e3
    ld16o_PIPE_0_1 (cc_lsu_slot_1_lsu,         IN(b0re_1_e9_e367_e367,0), JOIN(IN(m0_1_e10_e369_e368,1),BACK(m0_1_e10_e368_e369,1)));
    ld16o_PIPE_0_1 (cc_lsu_slot_2_lsu,         IN(b0im_1_e12_e370_e370,0), JOIN(IN(m0_1_e13_e372_e371,1),BACK(m0_1_e13_e371_e372,1)));
    ld16o_PIPE_0_1 (qpse1_pse1_lsu_slot_1_lsu, IN(b1re_1_e17_e373_e373,0), JOIN(IN(o1_1_e18_e375_e374,1),BACK(o1_1_e18_e374_e375,1)));
    ld16o_PIPE_0_1 (qpse1_pse2_lsu_slot_1_lsu, IN(b1im_1_e20_e376_e376,0), JOIN(IN(o1_1_e21_e378_e377,1),BACK(o1_1_e21_e377_e378,1)));
    ld16o_PIPE_0_1 (qpse1_pse1_lsu_slot_2_lsu, IN(b2re_1_e23_e379_e379,0), JOIN(IN(o2_1_e24_e381_e380,1),BACK(o2_1_e24_e380_e381,1)));
    ld16o_PIPE_0_1 (qpse1_pse2_lsu_slot_2_lsu, IN(b2im_1_e26_e382_e382,0), JOIN(IN(o2_1_e27_e384_e383,1),BACK(o2_1_e27_e383_e384,1)));
    and            (cc_alu_slot_alu,           JOIN(IN(o0_1_e72_e386_e385,0),BACK(o0_1_e72_e385_e386,0)), JOIN(IN(mask e74 e387 e387,1),BACK(m
    inc            (qpse1_pse2_agu_slot_mau,   JOIN(IN(o1_1_e75_e390_e389,0),BACK(o1_1_e75_e389_e390,0)), SPLIT(SPLIT(SPLIT(SPLIT(EXIT(o
    inc            (qpse1_pse1_agu_slot_mau,   JOIN(IN(o2_1_e77_e392_e391,0),BACK(o2_1_e77_e391_e392,0)), SPLIT(SPLIT(SPLIT(SPLIT(EXIT(o
    add            (cc_agu_slot_2_mau,         JOIN(IN(o0_1_e69_e394_e393,0),BACK(o0_1_e69_e393_e394,0)), IN(step e71 e395 e395,1), SPLIT(SPLI
  )
```


Schedule - Machine view



AVISPA schedule example

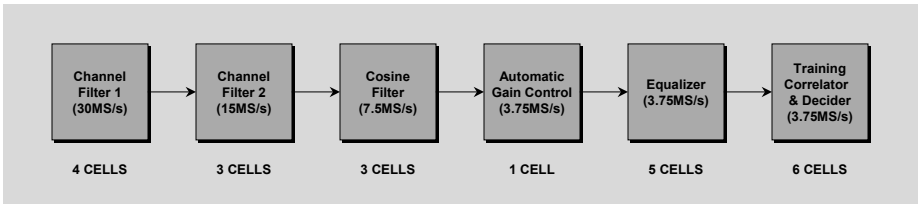
2k complex FFT application



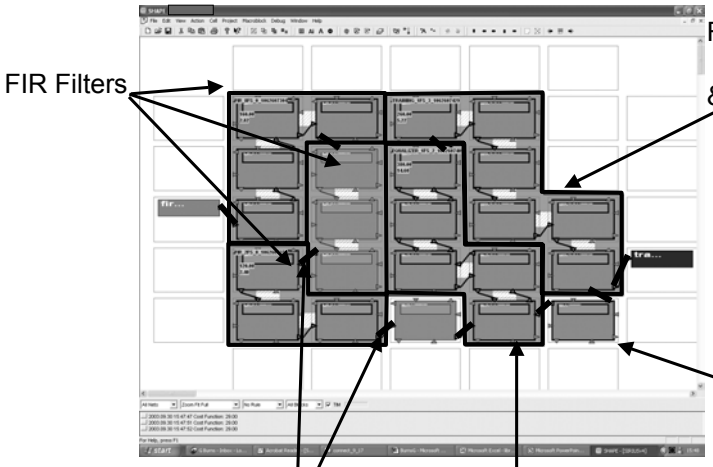
Silicon Hive

Example of BRESCA Chain

- Single Carrier Stream Processing



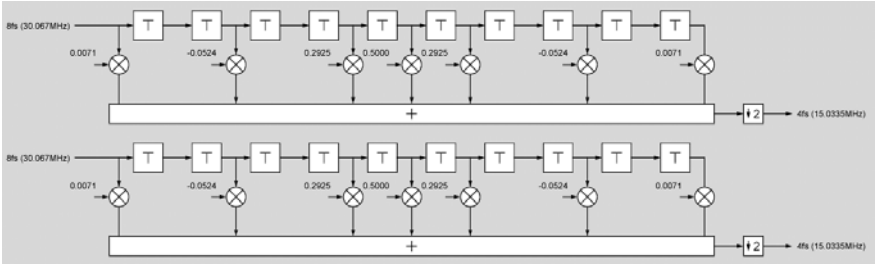
Silicon Hive



Silicon Hive

BRESCA Cell Mapping

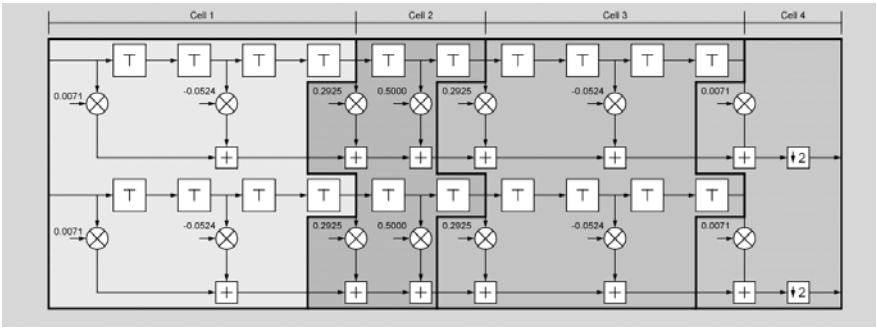
- 11 Tap, semicomplex halfband FIR (@30MS/s)



Silicon Hive

BRESCA Cell Mapping

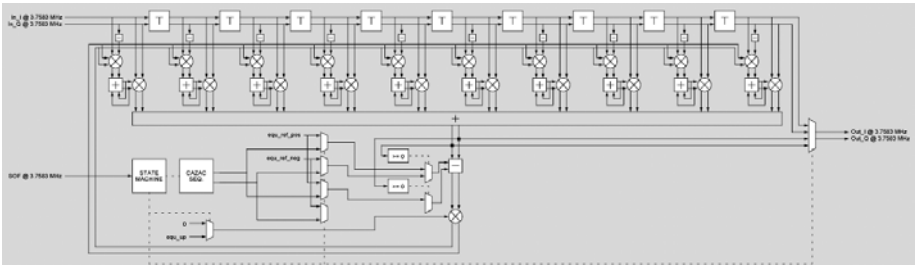
- FIR on 4 cells



Silicon Hive

BRESCA Cell Mapping

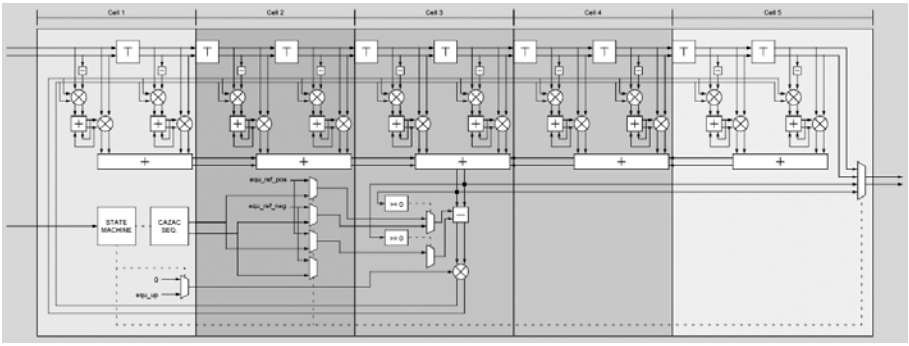
- 10 Taps complex Equalizer with Decision Feedback (@3.75MS/s)



Silicon Hive

BRESCA Cell Mapping

- Equalizer on 5 cells



Silicon Hive

Contents

- Introduction
- Issues in flexibility for SoC design
- SiHive Architecture overview
- Processor design
- Accelerator programming
- Applications

Silicon Hive

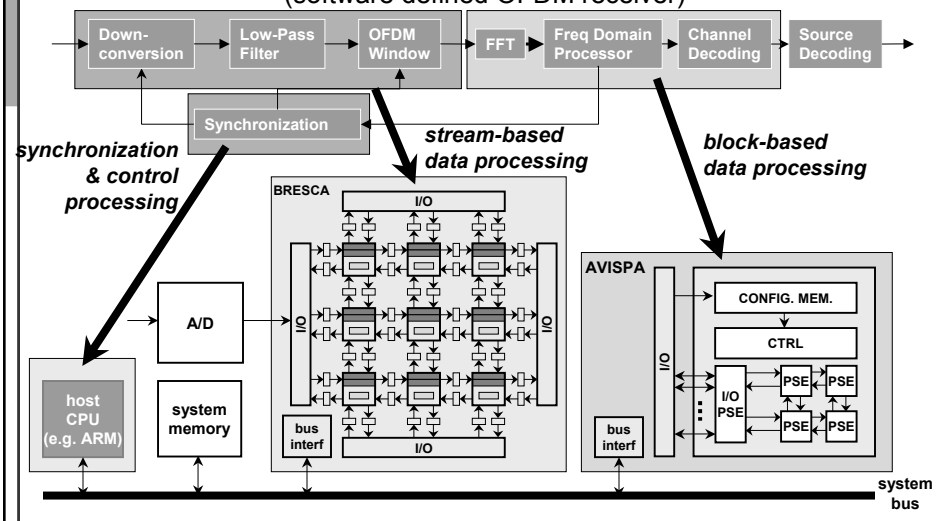
OFDM receiver

- Goals:
  - Flexibility for prototyping in an OFDM broadcast receiver for digital radio
  - Multi-standard consolidation
- Issues exploited by accelerator:
  - Timesharing enabled by high parallelism and efficiency of Avispa accelerator.
  - Imperfect ASIC module and memory utilization

Silicon Hive

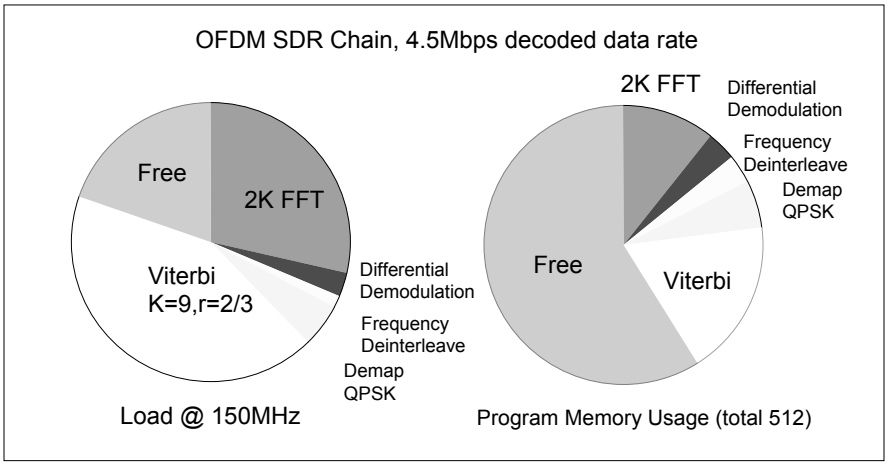
SoC design example

(software defined OFDM receiver)



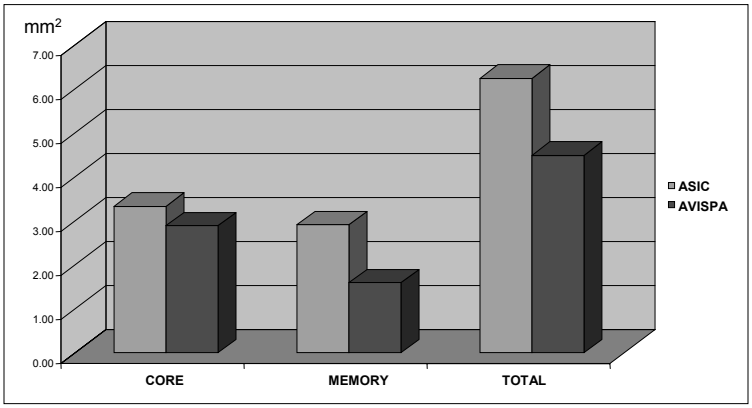
Silicon Hive

Avispa Mapping



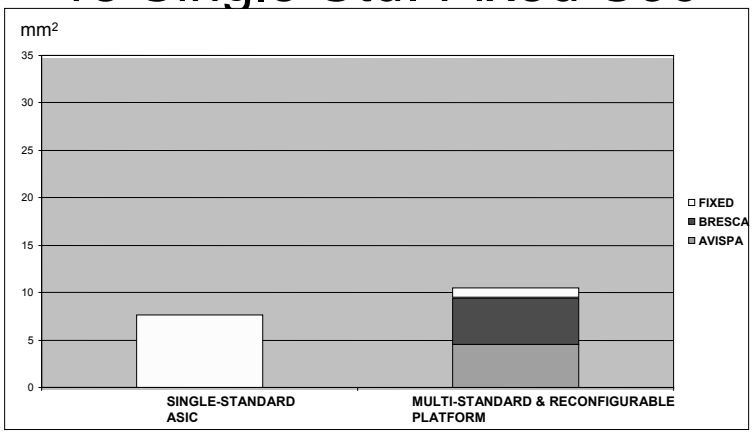
Silicon Hive

Avispa vs ASIC Area



Silicon Hive

Multi Std. Reconf. SoC vs Single Std. Fixed Soc



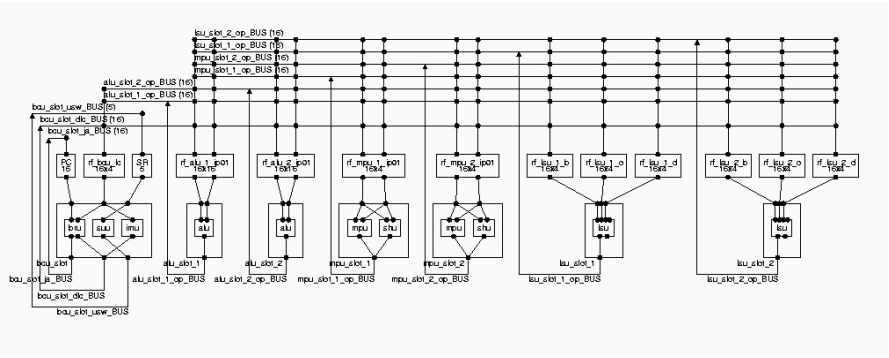
Silicon Hive

Moustique Accelerator

- Goals:
  - Flexibility for feature introduction in an MPEG4 video decoder with differentiating post processing features
  - Silicon area cost reduction
- Issues exploited by accelerator:
  - Timesharing enabled by modest processing rates relative to system clock
  - Imperfect ASIC module utilization
  - Easy deconfiguration if fixed configuration is desired

Silicon Hive

Architecture diagram



- 16 bit datapath
- 7 issue slots, 1 function units
- 78 register files

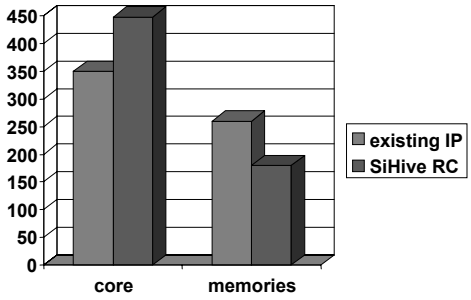
Silicon Hive

Processor Synthesis

- Programmable: instruction memory in RAM
  - Core: synthesized at 0.088 mm^2 @ 160 MHz
  - 0.13µ CMOS
  - Instruction RAM: 0.360 mm^2
    - 2 x 80 bits x 512 words
  - Data memory: 6 KB 0.18 mm^2
- Fixed: instruction memory in ROM
  - Instruction ROM: 0.07 mm^2

Silicon Hive

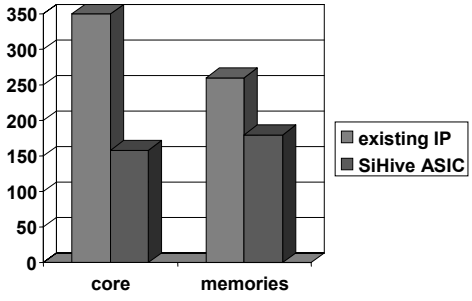
Results for scenario 1 (programmable)



- programmable solution competes with existing IP

Silicon Hive

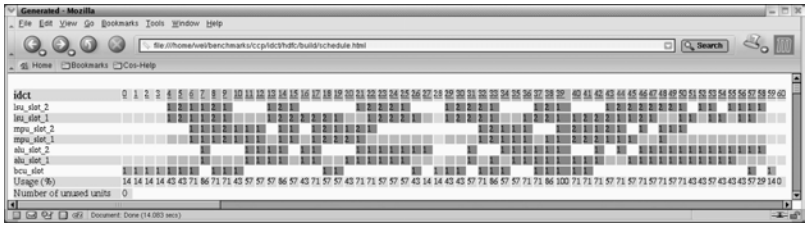
Results for scenario 2 (fixed)



- overall savings of 45% based on gates only

Silicon Hive

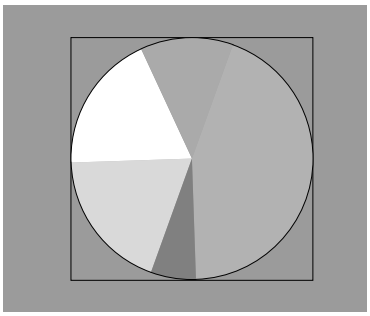
Scheduling example: IDCT



- 61 instructions
- requires only 30MHz operation

Silicon Hive

Results, processor load



- processor load dominated by DFD
- plenty of room for optimisation
- MPEG4 decoding + deblocking CIF 30 Frames/sec

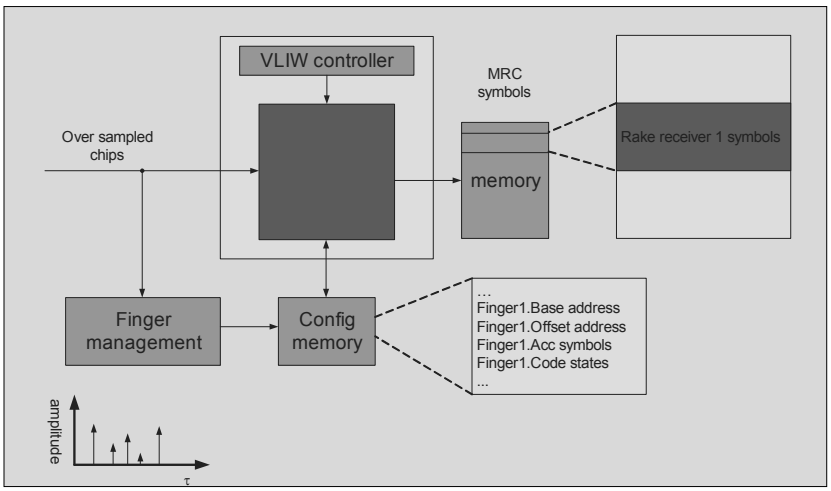
Silicon Hive

Scalable RAKE Receiver Core

- Goals
  - Multi-Standard Programmability
  - Scalability in time (fingers/RAKE)
  - Scalability in space (cores/SOC)

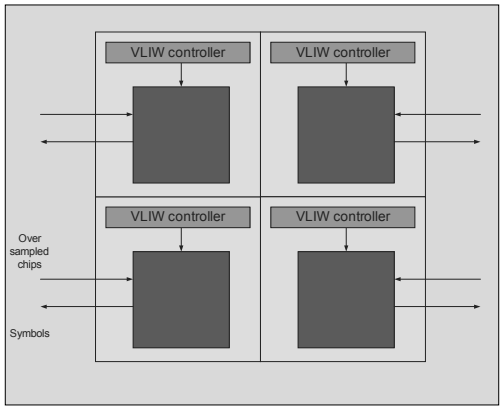
Silicon Hive

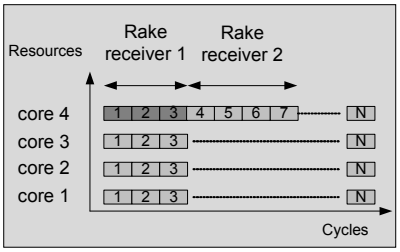
Rake receiver



Silicon Hive

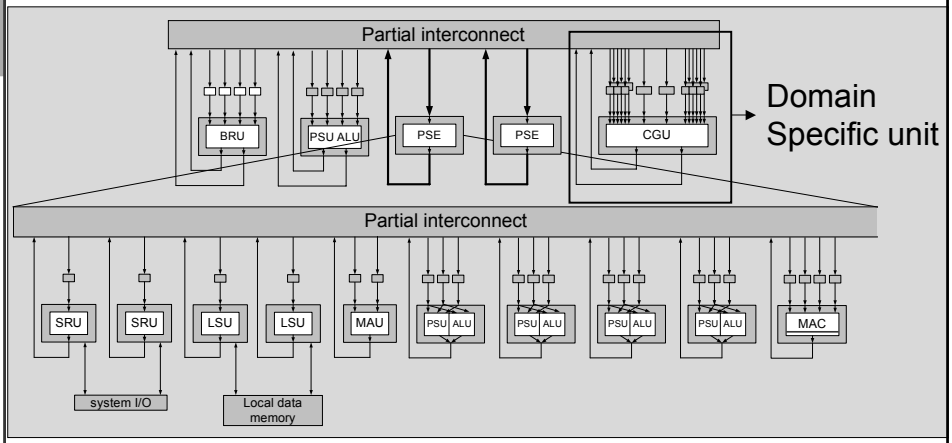
Rake receiver





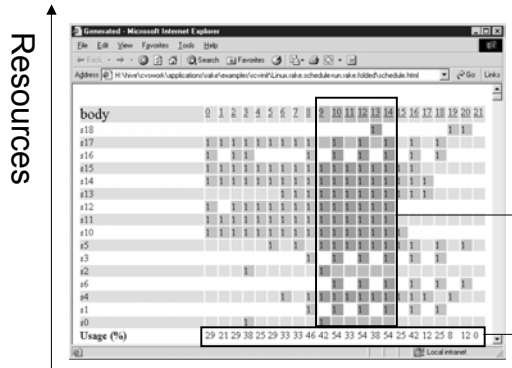
Silicon Hive

Architecture



Silicon Hive

Schedule example



- 2 cycles per finger

Body of loop

Issue slot usage per cycle

Silicon Hive

Wrap up

- Familiar Systems-on-chip platforms can be extended with programmable accelerators, to enable software-definition of applications that have been previously impractical for high-volume consumer price points. Programmable SoC's with efficiencies approaching ASICs are possible.
- Technology to automatically generate both accelerators and tools from the same representation, enables tradeoffs between flexibility and efficiency in hardware design.
- ***Both software programmability, and processor design flexibility, can be used to reduce costs of design and maintenance for SoC's throughout the product lifetime.***