ETH zürich

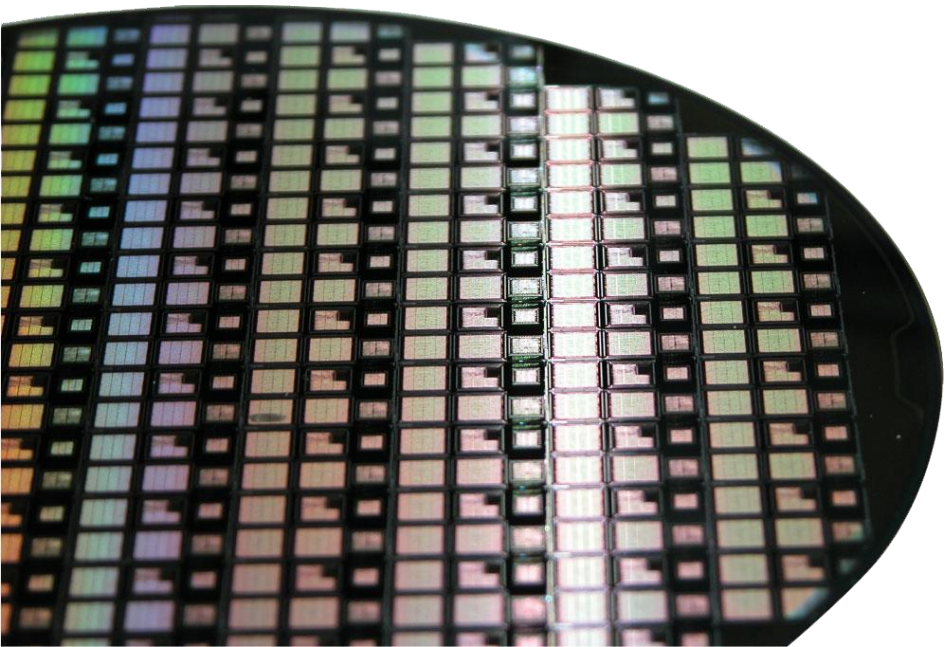
ASIP: Application Specific Instruction-Set Processor

Advanced System-on-Chip Design

17.05.2016

Michael Gautschi IIS-ETHZ

Luca Benini IIS-ETHZ



ETH zürich

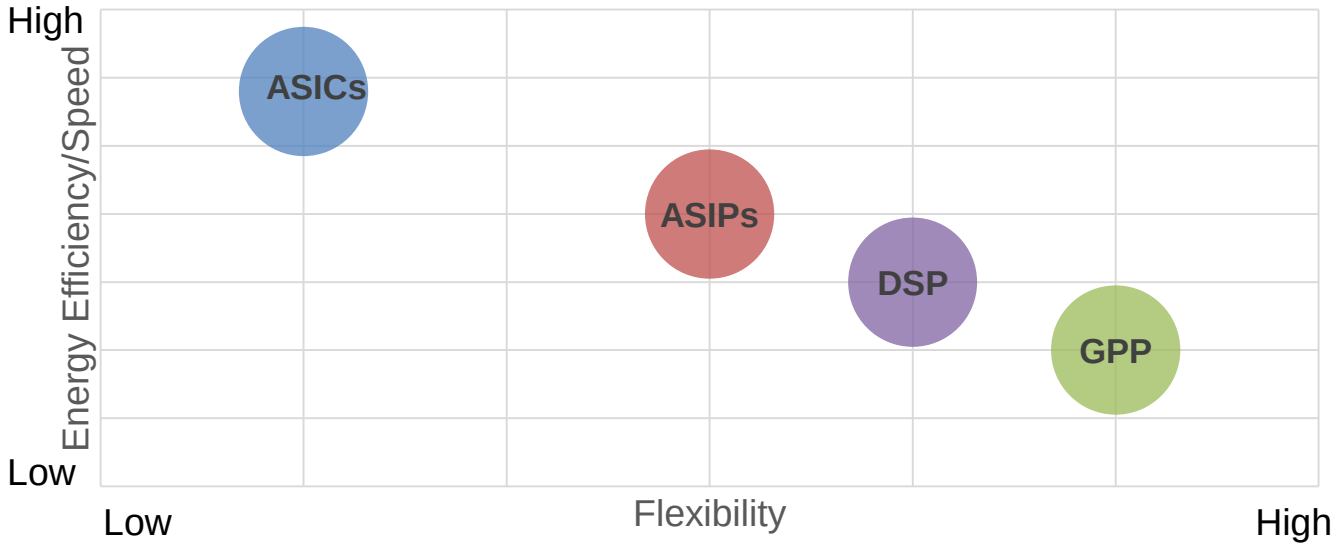
Integrated Systems Laboratory

Introduction

- Contents:
  - ASIP Introduction
    - Special function unit
    - Integration in SoC
  - Commercially available tools
    - Design flow
  - Exercise session about *Tensilica Xtensa Xplorer*
    - Introduction to *Xtensa Xplorer*
    - Exercise session
- Goals:
  - Knowing the pros and cons of ASIPs
  - Learning the general concepts of ASIPs
  - Being able to design and analyze ASIPs using *Xtensa Xplorer*

Motivation for ASIPs

- Problem:
  - GPP is too slow for some applications
  - Power consumption is not feasible for an embedded system



=> How do ASIPs achieve a better speed/energy efficiency?

ASIP Introduction

- ASIP = **A**pplication **S**pecific **I**nstruction-Set **P**rocessor
  - Compromise between **performance and flexibility**
  - Adding **custom instruction** to the CPU as dedicated hardware
  - Designed and optimized for a **specific application**
  - Commercial tool-suites available for **fast development**

- Example where ASIPs are used:
  - Digital signal processing
  - Audio processing
  - Mobile communication: encoding/decoding/synchronization
  - Cryptographic extensions
  - And many more

Weakness and strength of ASIPs

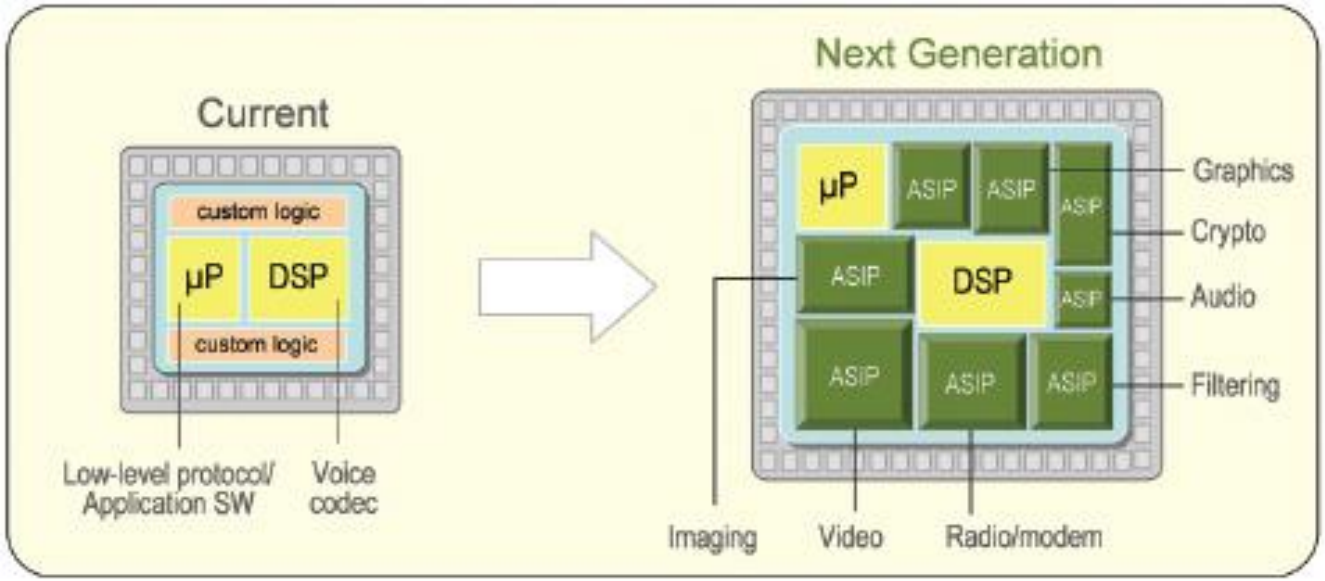
- An ASIP is a compromise between the two extremes ASIC and GPP
- ASIPs and ASICs are designed for a very specific application
  - But ASIPs are programmable and therefore almost as flexible as GPP
- ASIPs are great to be integrated in a embedded systems or SoC
  - Design effort is more on system level rather than on hardware

	GPP	ASIP	ASIC
Performance	Low	High	Very high
Flexibility	Excellent	Good	Poor
Power	Large	Medium	Small
HW design	Small	Large	Very large
SW design	Large	Large	None
Reuse	Excellent	Good	Poor
Market	Very large	Relatively large	small
Cost	Mainly on SW	SoC	Volume sensitive

Source: A.Nohl "*Application specific processor design: Architectures, design methods and tools*"

ASIPs in Multicore System-on-Chip

- SoC becomes **S**ea-**o**f-**C**ores
  - Many application specific processors integrated on a single chip
  - ASIPs as building blocks for heterogeneous designs



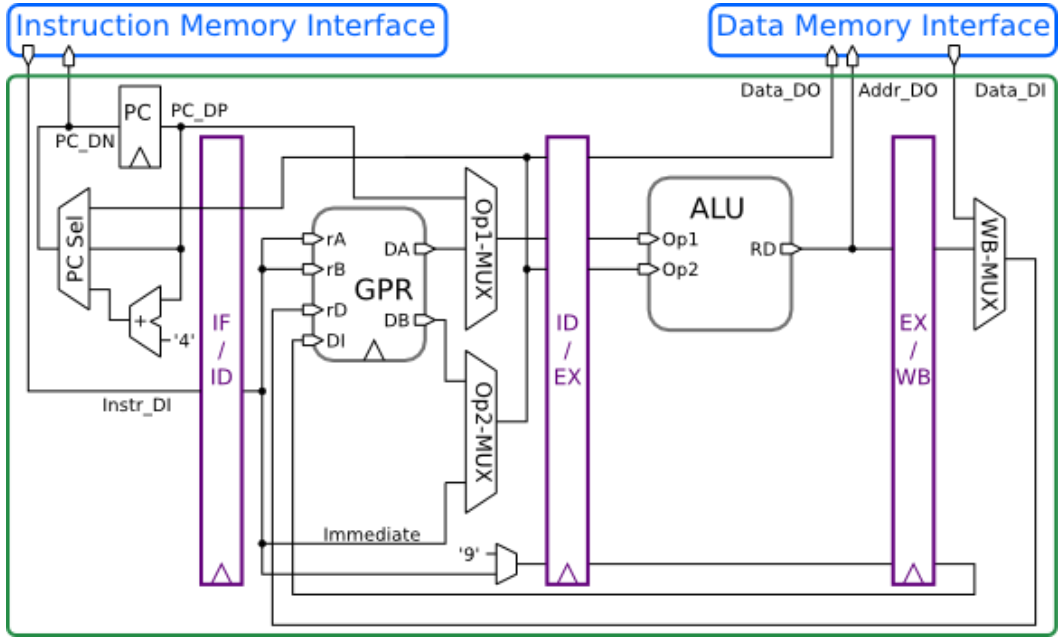
Processor Customization

- ASIPs are customized processors with hardwired functional units in the **data path.**
- Custom instructions are executed on a special function unit (SFU)
  - Compiler maps application source code to the extended instruction set
  - New instructions can be used intrinsically in C code

- Processor data path
  - 4 stage pipeline: { **I**nstruction **F**etch, **I**nstruction **D**ecode, **Ex**ecute, **W**rite-**B**ack}
  - 32 bit Instruction/ data interface

Different possibilities for data path integration:

a) SFU as comb. block in parallel of ALU
b) Comp. unit with internal state registers
c) Comp. unit with its own register file
d) Shared SFUs
e) Coprocessor with load/store interface

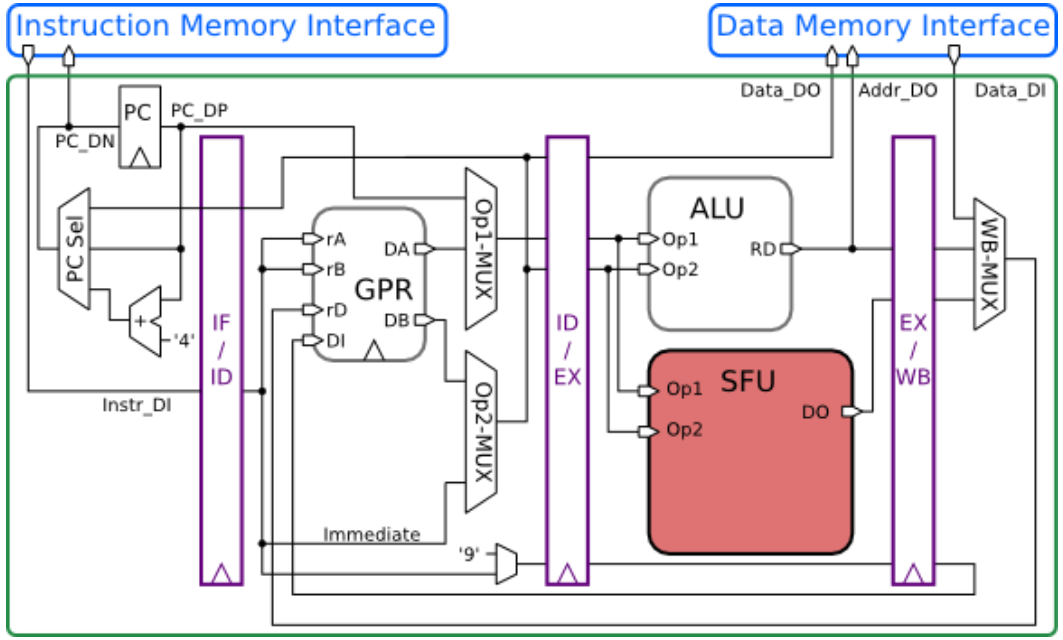


Special Function Unit

- Combinational unit, similar to ALU
  - Gets operands from general purpose registers (2 in this example)
  - Write back the result back to GPR
  - Multi-cycle instructions to map complex functions

Example:

- Modulo operation:
  - $c = a \% b$
- Square root:
  - $c = a + \mathrm{sqrt}(b)$

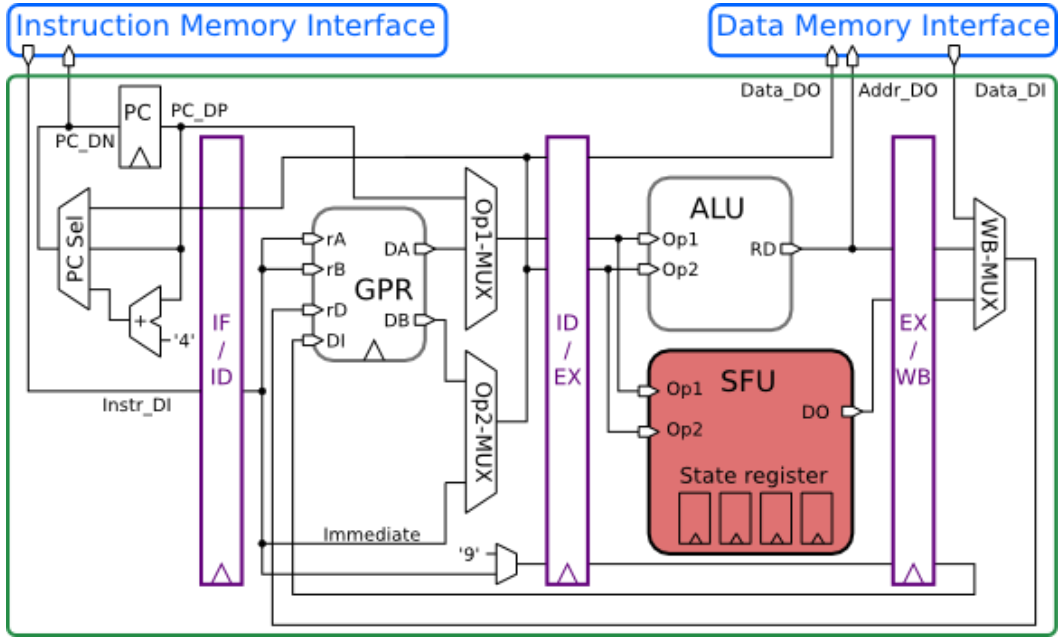


SFU with internal state registers

- Computation unit with internal state
  - State contains content which does not need to be stored in GPR
  - Read/write state with extra read/write instructions

Example:

- Accumulator
  - *a* += *b[i]* * *c[i]*
  - *a* stored in the state register (can be larger than 32 bit)
  1. initialize state to 0
  2. Compute MAC
  3. Read final state



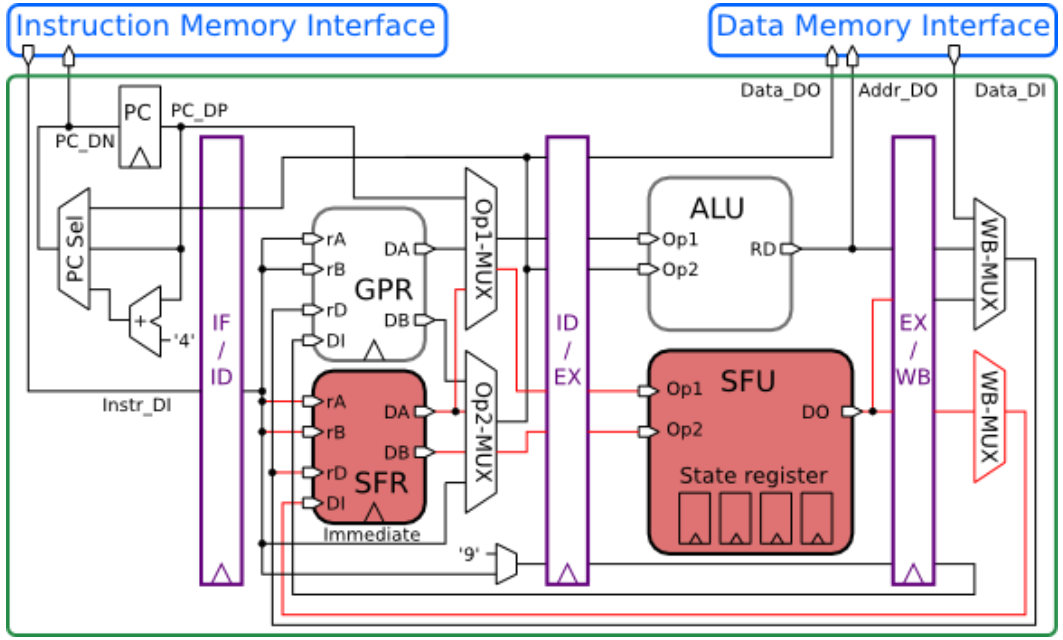
SFU with a separate register file

- SFU reads/writes from/to the *special register file* (SPR)
  - Compiler creates move/load/store instruction to SPR
  - Custom register width optimized for the SFU
  - Custom size and number of ports at the special register file

Example:

- Multiple accumulators
  - *a += b[i] * c[i]*
  - *d += f[j] * g[j]*

  => Store *d* and *a* in SPR with higher precision

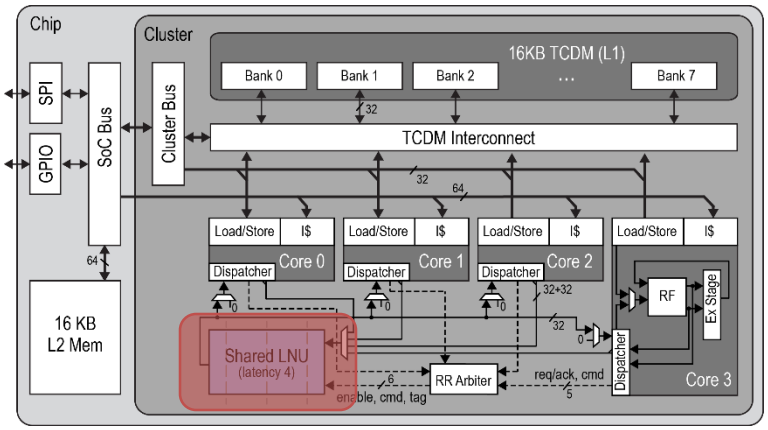


Floating-point Extensions in GPUs and Multi-core systems

- Expensive floating point operations can be computed in SFUs to approximate special functions:
  - sqrt(x), 1/x, $\log_2(x)$, 2^x, sin(x), cos(x), etc.

- Implemented with **function interpolators**
  - Speedup due to single cycle operation (no software emulations)
  - Limited precision (1ulp)
  - Implemented in GPUs [1,2]

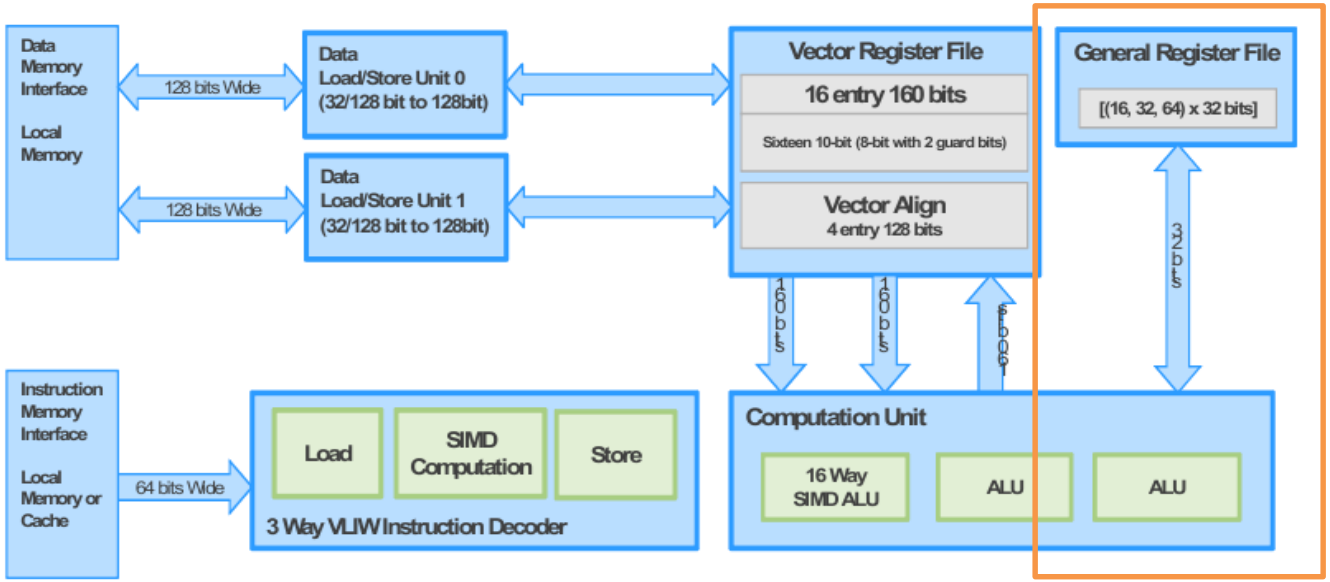
- **Shared** logarithmic number unit (LNU) [3]
  - Used to interpolate floating point additions and subtractions
  - 2^x, $\log_2(x)$, sin(x), cos(x), atan2(y,x)
  - Can be efficiently shared in a cluster of multiple processor cores



[1] Caro et. al, “High-Performance Special Function Unit for Programmable 3-D Graphics Processors”, TCAS 2009
[2] Oberman et. al, “A High-Performance Area-Efficient Multifunction Interpolator”, ARITH 2005
[3] Gautschi et. al, “A 65nm CMOS 6.4-to-29.2 pJ/FLOP@ 0.8 V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster”, ISSCC 2016

Customization with a Coprocessor

- Implement a set of instruction in a coprocessor
  - Customize the processor with more general instruction which can be used in several applications!
  - Share hardware resources for multiple instructions
  - Capable of issuing wider/more load/stores at the time
  - SIMD execution for maximum speedup
- Usage in software:
  - Program in standard C with the use of intrinsics

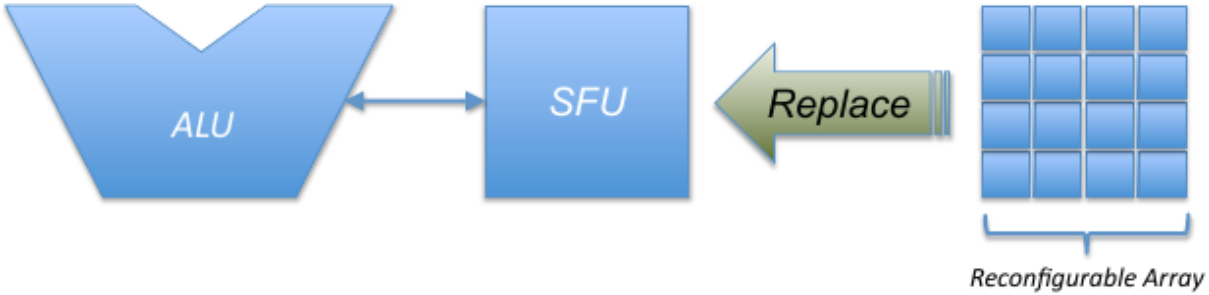


Data path of simple processor

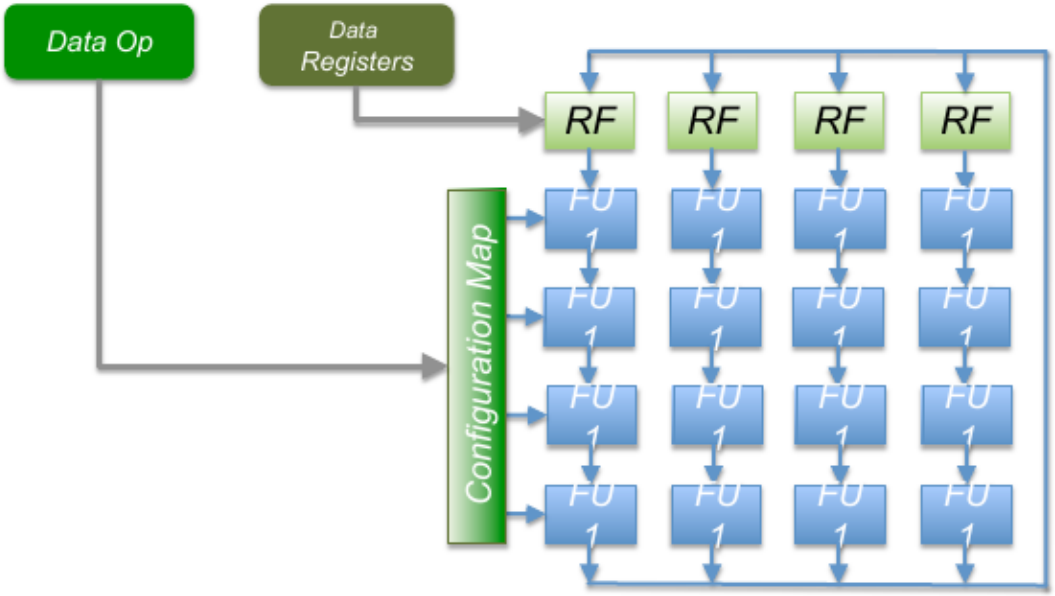
Tensilica ConnX Soft Stream Coprocessor Architecture

Reconfigurable ASIPs

- Design SFU for more than one single application!



- Replace SFU by configurable array of simple functions
- Configure the array to process the desired function
- FPGA-like approach
  - Efficient if not necessary to reconfigure very often



Source: Berekovic, TU-Braunschweig 2010

Commercially available ASIP/VLIW design tools

- LISA (Synopsys)

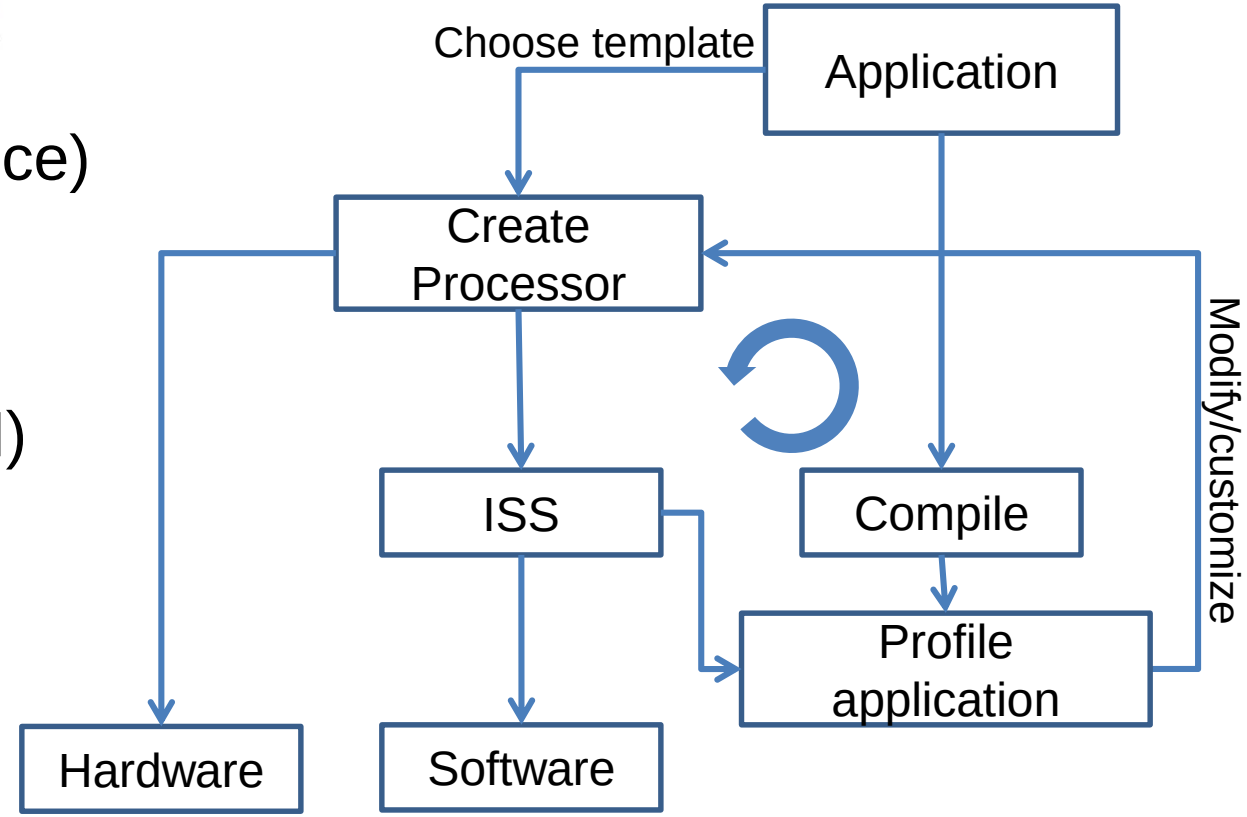
SYNOPSYS®
Silicon to Software™

- Xtensa (Tensilica/Cadence)

- OptimoDE (ARM)
- Codasip

codasip®

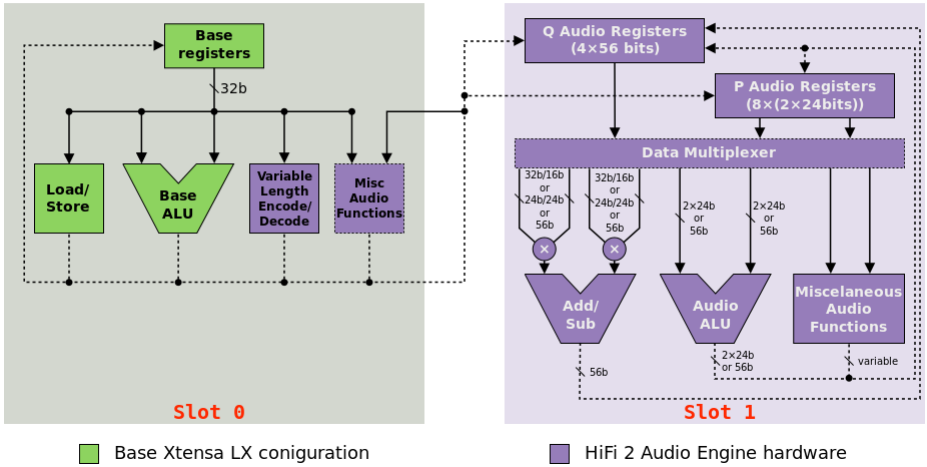
Design Flow:



Where is it used?

- **AMD Radeon R9-290**
  - High-end graphics card with TrueAudio
  - TrueAudio is a co-processor based on the HIFI core architecture of Cadence
  - AMD supports audio processing via TrueAudio in all its new chips (architecture generation GCN 1.1)

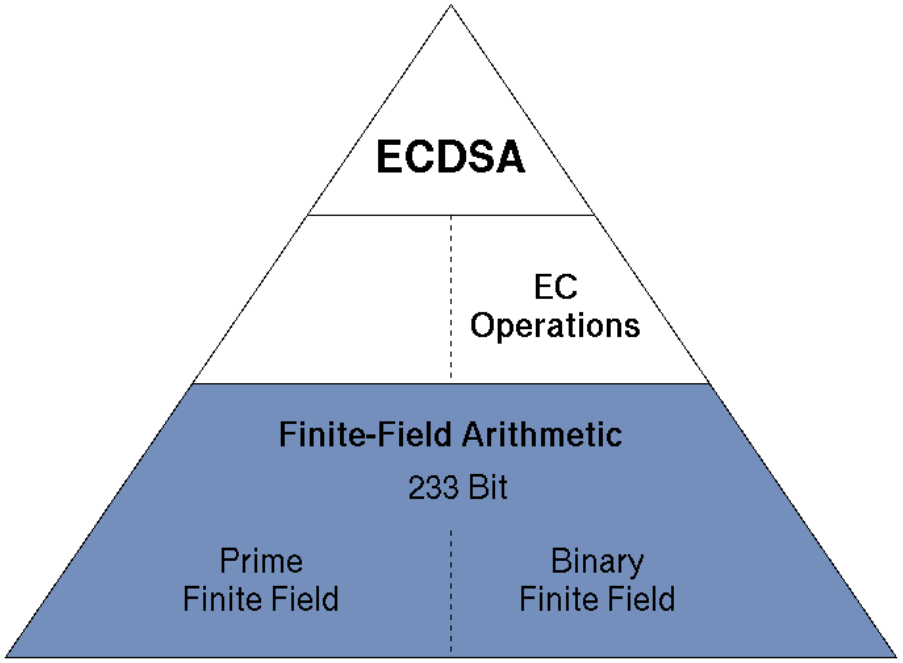
Datapath and registers of Cadence Tensilica's HiFi 2



- Samsung Galaxy S3
  - Voice recognition used by S-Voice
  - Uses low power HIFI xtensa core

Cryptographic Example: ECDSA signature verification

- **E**lliptic **C**urve **D**igital **S**ignature **A**lgorithm (ECDSA)
  - Signature validation with **E**lliptic **C**urves (EC)
- ECDSA requires prime and binary finite field operations
- Add, sub, mult, div, etc. defined in the finite-field arithmetic
  - We used the NIST B-233 curve



- Flexibility is good!
- Possible to support different algorithms and different standards with one hardware!

- Finite field operations are not suitable for general purpose architectures
  => hardware much more efficient!

Cryptographic Example: Base processor configuration

- Processor configuration:
  - 5 stage integer pipeline
  - 32 32bit general purpose registers
  - 16x16 bit multiplier
  - 2 KB I$, D$

- Estimated performance (65nm LP technology):
  - Max speed: 344 MHz (worst case)
  - Area: 83 kGE
  - Power : 20 mW (not verified)

- ECDSA algorithm ported from a former semester project [1]

[1] Semester Thesis by A. Traber, S. Stucki, 2014, A Unified Multiplier Based Hardware Architecture for Elliptic Curve Cryptography

ECDSA Signature Verification Algorithm

Profiling results using Xtensa Xplorer:

Operation	Total (cycles)	Total (%)	# function calls	Code size (bytes)
ECDSA Verification	46'674'997			6518
$GF(2^{233})$ multiplication 16x16 bit binary f. field mult.	42'063'643 34'199'189	90.1% 73.3%	2'309 591'104	366 157
$GF(2^{233})$ squaring	2'706'852	5.7%	2'472	446
others	1'904'502	21.0%	-	4'930

- Simple squaring operation on a 233 bit binary field:
  - Insert '0' bit between each input bit
    - e.g. '1101' => '01010001'
  - Reduce resulting 466 bit number to 233 bit
- Requires masking and shifting in C -> not efficient at all!
- Hardware architectures can do such operations in one cycle!
  => add custom instructions for 16bit mult, and squaring

Optimizing the multiplication in GF(2^{233})

Original C code

```
// shortened C-code: binary field 16 bit
// multiplication

uint16_t a, b;       // input
uint32_t prod = 0;   // output

for(i = 0; i < 16; i++) {
  if((a&(1<<i))
      prod ^= b << i;
}
```

~60 cycles

New C code

```
// intrinsic function call

prod = BinMul16(a,b);

// a, b, prod are stored
```

Speedup of factor 60!

"TIE" – Tensilica Instruction Extensions

```
// shortened BinMul operation
operation BinMul16 {out AR res, in AR a, in AR b}{}{
   wire [31:0] temp0 = (a & (1 << 0)) ? (b << 0)           : 32'b0
   wire [31:0] temp1 = (a & (1 << 1)) ? (temp0^(b << 1))   : temp0;
   wire [31:0] temp2 = (a & (1 << 2)) ? (temp1^(b << 2))   : temp1;
   // …
   wire [31:0] temp15 = (a & (1 << 15)) ? (temp14^(b << 15)) : temp14;
   assign res = temp15;
}
```

1 cycle only!

Results: Performance

- Profiling results of specialized circuit
- Reduced code size: 5.1 KB vs 5.7 KB

Operation	Total (cycles)	# cycles before	Speedup
ECDSA Verification		46’674’997	
$GF(2^{233})$ multiplication 16x16 bit binary f. field multiplication		42’063’643 34’199’189	
$GF(2^{233})$ squaring		2’706’852	
others		1’904’502	

- Comparison to HW-architecture:[2]
  - Coprocessor with 16 bit datapath requires ~1’850’000 cycles (12 kGE)
  - Factor 3.3 slower

[2] M. Gautschi, M. Mühlberghuber et.al. , SIRIOUS: A tightly coupled ECC Coprocessor for the OpenRISC

Results: Area and Timing

- Synthesis results with UMC 65 nm technology

Core comparison	Max Speed (MHz)			Area (kGE)			
	Synthesis		Estimation	Synthesis			Estimation
	TC	WC		TC	WC	344 MHz	
Basic configuration	857*	580*	344	103	107	98	83
With hardware extensions	850*	589*	344	107	111	100	85
Instruction extension				4.1	4.2	2.3	2.2

* Synthesized without memories; expected drop by 200-400MHz when synthesizing with a 2KB cache

- Conclusion:
  - Flexible architecture fully integrated in an application processor
  - 7.5x speedup at very low costs and design time (< 1 week)
  - Only 2.2 kGE hardware overhead, (datapath of co-processor 12 kGE)

Summary

- ASIPs are widely used in application specific domains
- ASIP can increase energy efficiency
- Possible to create complex SoCs with ASIPs
- ECDSA example is based on a mini-project completed by Sven Stucki

Q&A