Xtensa + TIE Feature Update

To RF.3 release – some key points of my choice ☺

Marcus Binning, Senior AE Manager, Europe (marcusb@cadence.com)
Tensilica User Day
Hanover
Feb 9th 2016

Agenda

- Overview
- Memory System
- TIE
- Tools / Debug

© 2016 Cadence Design Systems, Inc. All rights reserved.

© 2016 Cadence Design Systems, Inc. All rights reserved.
cādence®

Overview of features up to Release RF.3

- Not going to identify features by release, just where things stand today
  - The release notes will tell you when things were introduced
- We will cover some key (new) TIE language features
- First will cover some of the major “other” pieces of the Xtensa architecture

- NOTE – This talk will concentrate on GENERIC features, and will not describe all the DSP ISAs currently available
  - Lots of information on http://ip.cadence.com/ipportfolio/tensilica-ip

- Conceptually – LX6 is a development of all the LX that went before
  - No MAJOR discontinuities …. Yet ☺

© 2016 Cadence Design Systems, Inc. All rights reserved.

Memory System

© 2016 Cadence Design Systems, Inc. All rights reserved.
cadence®

Xtensa Architecture:
Local Memory Subsystem

IROM0
IRAM0
System Bus
Instruction Fetch and Decode
N-issue FLIX parallel pipelines
Base Execution Pipeline
Prefetch Unit
SDRAM
Register File
ALU
I-Fetch Buffer
Execution Units, Register Files and Interfaces
Execution Units, Register Files and Interfaces
Master Interface
MUL16/32, MAC16, NSA, MIN,MAX, etc
AXI\AHB-lite Bridge
Dev A
LD/ ST0
Slave Interface
SPFPU, DPFPU
Dev B
LD/ ST1
PIF
Additional ISA
DMA
DRAM0
DROM

Local Memory Subsystem

- Base ISA Feature
- Configurable Function
- Optional Function
- Optional & Configurable
- Designer Defined Extensions
- External RTL & Peripherals
- Memories & Caches

© 2016 Cadence Design Systems, Inc. All rights reserved.

Xtensa Architecture:
Dual Load/Store and Banking

Instruction Fetch and Decode
Base Execution Pipeline
Register File
ALU
MUL16/32, MAC16, NSA, MIN,MAX, etc
SPFPU, DPFPU
Additional ISA
N-issue FLIX parallel pipelines
Execution Units, Register Files and Interfaces
Execution Units, Register Files and Interfaces
I-Fetch Buffer
Instruction Cache
IRAM0
LD/ST0
Store Buffer
LD/ST1
Store Buffer
Data Cache
DRAM0
PIF
System Bus
SDRAM
Dev A
Dev B
DMA

Legend	
Base ISA Feature	Optional & Configurable
Configurable Function	Designer Defined Extensions
Optional Function	External RTL & Peripherals
	Memories & Caches

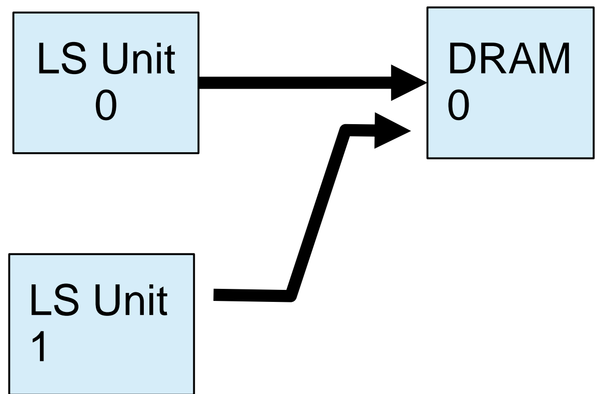
Local Memory with Dual Load/Store and Banking

© 2016 Cadence Design Systems, Inc. All rights reserved.

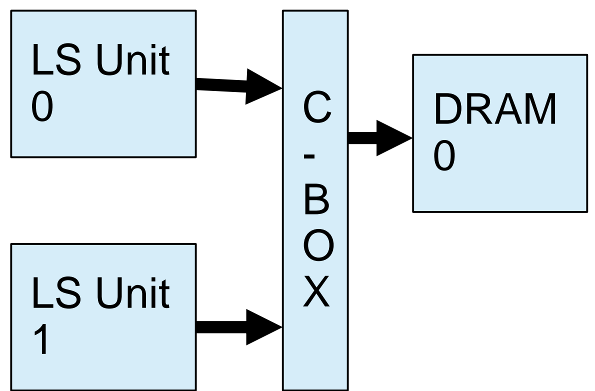
Parallelize Memory Accesses with Dual Load/Store

- Requires FLIX instruction to fully utilize dual Load/Store

Requires Dual Ported Memories



Configuration with C-Box → no dual ported memory required



- C-Box is an "arbiter-mux"
  - Multiple cycles are needed to satisfy multiple requests, if they are going to the same DRAMx address space
  - Loss some of FLIX efficiency when multiple Load/Store target same Data RAM

© 2016 Cadence Design Systems, Inc. All rights reserved.

Data Banking Example with Two Banks

Bank Select

Address		Data Width

31 ... 0

Bank 0

	Bank	Data
…000	0	0
…001	0	2
…002	0	4
…003	0	6
…		8

External address to bank0

Bank 1

	Bank	Data
…000	1	1
…001	1	3
…002	1	5
…003	1	7
…		9

External address to bank1

© 2016 Cadence Design Systems, Inc. All rights reserved.

Two Load/Store Units with Data Cache

- Configuration requirements
  - Must be a write-back D-cache
  - Must have at least 2 banks
  - Must have early restart (will be explained later)

- Each L/S Unit has its own copy of the DTag RAM.
  - Two simultaneous requests to different addresses require two DTag lookups for parallel access
  - DTag RAMs are written with the same data.

© 2016 Cadence Design Systems, Inc. All rights reserved.

DCache with Banking

Cache-lines of one way are spread over multiple banks
→ Parallel access of values in the same cache line

Way 0

	Bank	
…0000	0	Cache line 0, way 0
…0001	0	Cache line 1, way 0
…0002	0	Cache line 2, way 0
…0003	0	Cache line 3, way 0
…	0	Cache line 4, way 0

	Bank	
…0000	1	Cache line 0, way 0
…0001	1	Cache line 1, way 0
…0002	1	Cache line 2, way 0
…0003	1	Cache line 3, way 0
…	1	Cache line 4, way 0

Way 1

	Bank	
…0000	0	Cache line 0, way 1
…0001	0	Cache line 1, way 1
…0002	0	Cache line 2, way 1
…0003	0	Cache line 3, way 1
…	0	Cache line 4, way 1

	Bank	
…0000	1	Cache line 0, way 1
…0001	1	Cache line 1, way 1
…0002	1	Cache line 2, way 1
…0003	1	Cache line 3, way 1
…	1	Cache line 4, way 1

Benefits:

- Cache lines are loaded to smaller memories
- Software accesses only the bank it needs

© 2016 Cadence Design Systems, Inc. All rights reserved.

Memory Errors and ECC

- Memory protection available for Caches (tag and data) and local RAM (DataRAM and InstRAM)
- Can configure the memory error method for Instruction Side and Data side separately
- Error Correction
  - Correct single bit memory errors “on-the-fly”.
  - Correctable error will trigger a replay of the memory access with the corrected value
  - Xtensa does not write the error-corrected data back to the memory

Uncorrectable Error → ECC double bit error or Parity errors will trigger exceptions

Memory Type	Parity	ECC
Data Memories	1 bit per Byte	5 bits per Byte
Data Caches	1 bit per Byte	5 bits per Byte
Instruction Memories	1 bit per 32 bit word	7 bits per 32 bit word
Instruction Caches	1 bit per 32 bit word	7 bits per 32 bit word
Instruction and Data Cache tag array	1 bit per tag	7 bits per tag

© 2016 Cadence Design Systems, Inc. A

PIF on Xtensa

- The Processor Interface (PIF) protocol is a high bandwidth, full duplex system interconnect protocol
- PIF Port
  - Reads and write accesses are never speculative
  - Recommended interface for memory that require additional wait states
- PIF is a configuration option
  - If Cache is configured, PIF is required
  - If Inbound PIF is configured, PIF is required
- PIF is fully mapped to Xtensa's 4Gbyte address range, excluding address ranges of InstRam, InstRom, DataRam, DataRom and XLMI,
- PIF additional optional features to decrease cache miss latency
  - Critical word first
  - Early Restart
  - Prefetch

© 2016 Cadence Design Systems, Inc. All rights reserved.

PIF Protocol - Basic

- Split transaction interface
  - Separate Request & Response Channels
  - Master arbitrates for request channel
  - Slave arbitrates for response channel
- Simple flow-control for request and response
  - Initiator: Valid
  - Receiver: Rdy
  - When both Valid and Rdy are asserted at rising edge of CLK, a transaction completes
- Supports multiple outstanding requests
  - Responses can be out of order
  - 6-bits transaction ID used by requestor
- Write response for writes can be configured to identify bus data or address errors
  - 16 unique IDs to allow up to 16 write outstanding write responses
  - To ensure read/write memory ordering and synchronization use memw instruction

→ Xtensa stalls when memw is executed until all outstanding write responses have returned

© 2016 Cadence Design Systems, Inc. All rights reserved.

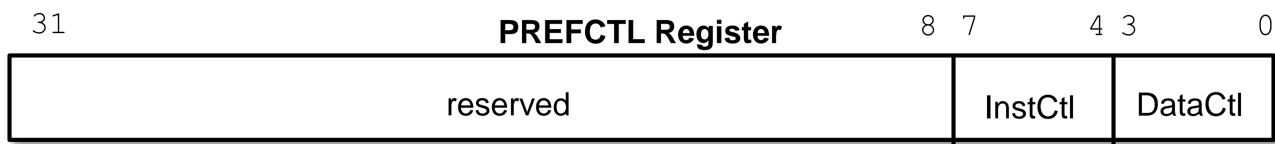
Cache Prefetch: Overview

- Problem: long cache miss latency
  - Xtensa pipeline will replay and then stall during cache miss
  - It may take many cycles to load a cache line from system memory
  - Load miss or Instruction Fetch miss waits for PIF block read response
  - Need to load many values stored in contiguous addresses
- Solution: cache Prefetch option
  - Prefetch additional cache line(s) from contiguous addresses ahead of a cache miss
  - Prefetch hardware Identifies a *stream*
  - Requires Data cache; Instruction cache is optional
  - Additional Cache lines are Stored in *Prefetch Buffer*
- Benefit: Miss penalty for long (50-100 cycles) system memory latency is reduced

© 2016 Cadence Design Systems, Inc. All rights reserved.

Hardware Prefetch Modes

- Prefetch Buffer (SRAM) is outside of the Xtensa core
- Buffer size configurable to 8 or 16 entries, of Cache Line size
- Hardware Prefetch mode is controlled with the PREFCTL register



Prefetch Control Nibble Value	"Stream Detected" when a cache miss occurs and …	Additional cache lines prefetched, when *no* stream is detected	Additional cache lines prefetched, when stream is detected	*Maximum Number of buffer entries per stream
0x0	-	none (disabled)	none (disabled)	-
0x4	Hit in Prefetch Buffer *OR* detected using *Miss History Table*	none	2*	2
0x5	Hit in Prefetch Buffer	1*	2*	2
0x8	Hit in Prefetch Buffer	2*	4*	4

© 2016 Cadence Design Systems, Inc. All rights reserved.
cādence®

Example of “Mode 0x4” Prefetch Algorithm (Animated)

Prefetch Buffer

cacheline *a+2*

cacheline *a+3*

DCache

cacheline *a*

cacheline *a+1*

cacheline *a+2*

HIT!

Prefetch Logic

Miss History Table

Miss *a* STREAM!

Miss *a+1*

Cache Control

Request ***a+3***

Request ***a+4***

PIF

Request ***a+1***

I R E M W

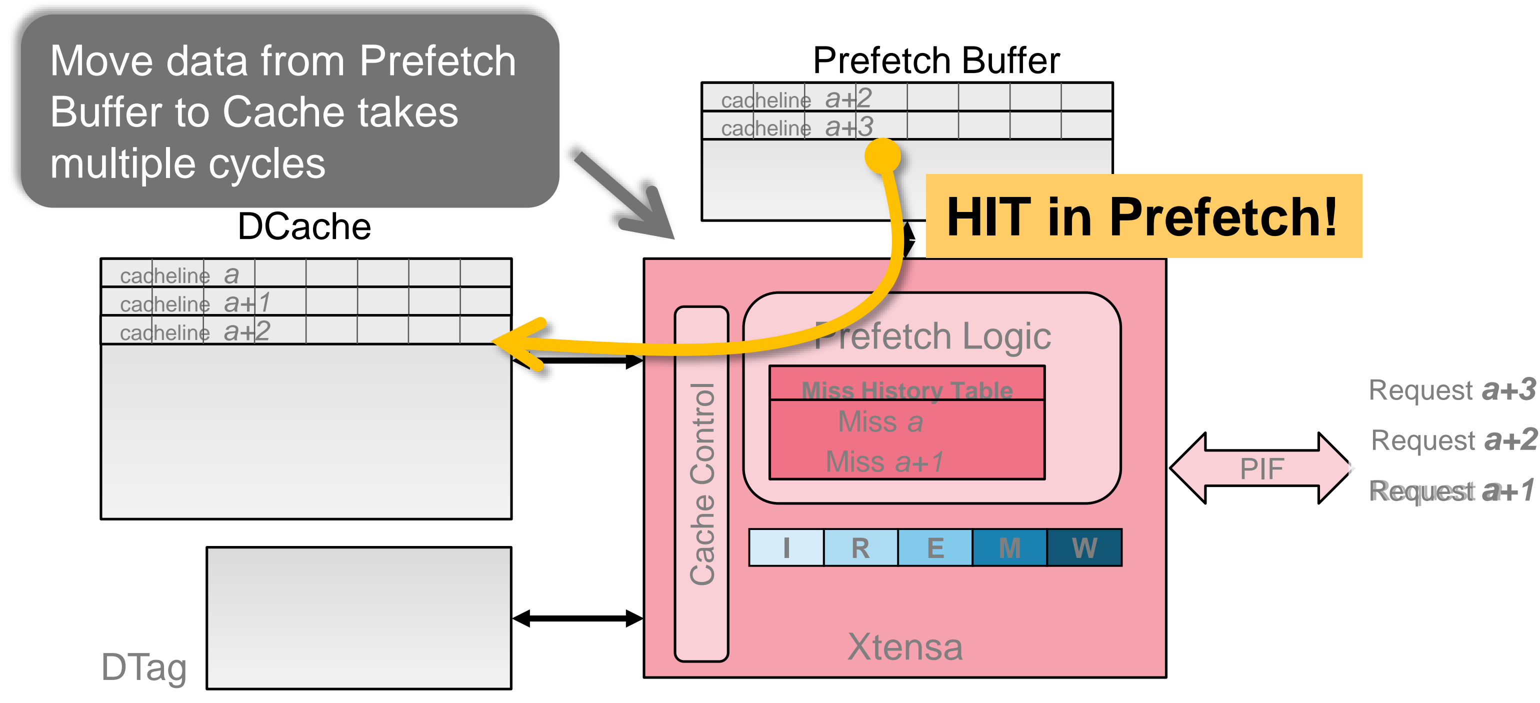
DTag

MISS!

Xtensa

© 2016 Cadence Design Systems, Inc. All rights reserved.

Why Prefetch direct to L1 ?



Assume for example 64-bit wide cache, and a 64-byte cache line

→ Takes 8 cycles to move 1 cache line from Prefetch to L1

- Benefit: Prefetch to L1 increases performance for lower latency (1-20 cycles) system memory accesses, but might thrash the cache

© 2016 Cadence Design Systems, Inc. All rights reserved.

Prefetch types in LX6 core

- Three prefetch types, one objective: avoid or reduce cache-miss latency
- All types use the same prefetch hardware
  - HW Prefetch
    - Automatically initiate prefetch upon stream detection or cache miss.
    - Predictive hardware prefetching of instruction and data
    - Alleviates penalty associated with large external memory latencies
  - SW Prefetch
    - User's software program initiates Prefetch
    - Prefetches single data cache-line into L1-Data Cache or Prefetch buffer.
  - Block Prefetch: *New Feature*
    - User's software initiates Prefetch
    - Allows prefetch of a block of data spanning multiple cache-lines
    - Block prefetches can be grouped and queued in user software
    - User can limit number of Block prefetch upgrade requests

© 2016 Cadence Design Systems, Inc. All rights reserved.

Prefetch HW

Prefetch Buffer

Blk_Prefetch Entry 0
Blk_Prefetch Entry 1
Blk_Prefetch Entry 7

Prefetch Entry 0
Prefetch Entry 1
Prefetch Entry N-1

Pref & Modify Prefetch Entry 0
Pref & Modify Prefetch Entry 1

Configuration Dependent:
8 or 16 Entries

Max # of entries: limit entries for block prefetch upgrade requests: software programmable

Block prefetch buffer:
Entries: 8

Dedicated special entries for “Prefetch & Modify” category Block prefetch

DCache

cacheline a
cacheline a+1
cacheline a+2

ICache

Cache Control

Prefetch Logic

Miss History Table

I R E M W

Xtensa

PIF

© 2016 Cadence Design Systems, Inc. All rights reserved.

Block Prefetch Operations

- Operations - supported by XTHAL APIs
  - Load a block of data into the L1 data cache (upgrade)
  - Write back (Castout & invalidate) a block of data from the L1 data cache (downgrade)
  - allocate a block in the L1 data cache whose content to be modified, without reading memory (Prefetch and modify upgrade)
  - cache block operations: cancel, wait, abort, start new group
- User needs to initiate the block prefetch of a data block in advance
  - Block prefetch can eliminate cache misses (both in the normal case and in the "prefetch and modify"). Make sure to enable DL1 option (prefetch to L1 DCache) in PREFCTL register.
  - In the case of certain cache line allocation conflicts with the Load/Store units, the prefetch logic can choose to keep the prefetch response data in the prefetch buffer. Such prefetched data will be brought to L1 cache when the actual data cache miss occurs.

© 2016 Cadence Design Systems, Inc. All rights reserved.

Memory copy example

Assuming config has:

- Total 16 prefetch entries
- Cache line size is 128 bytes
- Allocate 8 entries for block prefetch

```
#include <xtensa/core-macros.h>


test_fun(u8 *pdest, u8 *psrc)  {
    u32 size = 1024;                              // 8 lines of 128 bytes each
    // reserve 8 entries for block prefetch, Inst/Data Ctl: Prefetch 2 lines on stream
      detect
    xthal_set_cache_prefetch_long (XTHAL_PREFETCH_BLOCKS(8) | XTHAL_DCACHE_PREFETCH_MEDIUM |
      XTHAL_ICACHE_PREFETCH_MEDIUM | XTHAL_DCACHE_PREFETCH_L1);
                    /*- - - - - other code here - - - - - - */
    xthal_dcache_block_prefetch_for_read_grp (psrc, size);          // Group prefetch
      begins
    xthal_dcache_block_prefetch_modify (pdest, size)        // prefetch & modify; no PIF
      read.
                    /*- - - - - other code here - - - - - - */
     memcpy(pdest, psrc, size);         // 1024 bytes copy
                    /*- - - - - other code here - - - - - - */
```

Begins block prefetch and groups the subsequent block prefetch requests

Prefetch and modify !
No PIF request, no refill to L1 cache.
Just request an L1 Cache allocation

© 2016 Cadence Design Systems, Inc. All rights reserved.

Prefetch types – At a quick glance

	HW Prefetch	SW prefetch	Block Prefetch
Xtensa Core Architecture	LX 4 onwards	LX 4 onwards	LX 6 onwards
Prefetch initiated by	HW on stream detect	User's software program	User's software program
Prefetch quantity	1/2/4 cache lines – depending on aggressiveness. Software programmable.	Only one cacheline	One or Multiple cachelines
Cache supported	Both Dcache and Icache *	Only for DCache	Only for Dcache
HW configurability	Prefetch buffer & number of entries – Core configuration option	Prefetch buffer – Core configuration option	Prefetch buffer & number of entries – Core configuration option
Software programmable	Prefetch aggressiveness for I$ and D$ independently	Yes.	Can reserve #entries in prefetch buffer for block prefetch
Configuration register	PREFCTL	PREFCTL	PREFCTL

** Prefetch always applies to the data cache, and applies to the instruction cache for some configurations of line sizes. (Changed in LX6)*

© 2016 Cadence Design Systems, Inc. All rights reserved.

© 2016 Cadence Design Systems, Inc. All rights reserved.

cadence®

Important new TIE features

- Register file architectures
- FLIX design
- Hardware sharing (shared semantics)
- Protos – some notes
- Tc reports

© 2016 Cadence Design Systems, Inc. All rights reserved.

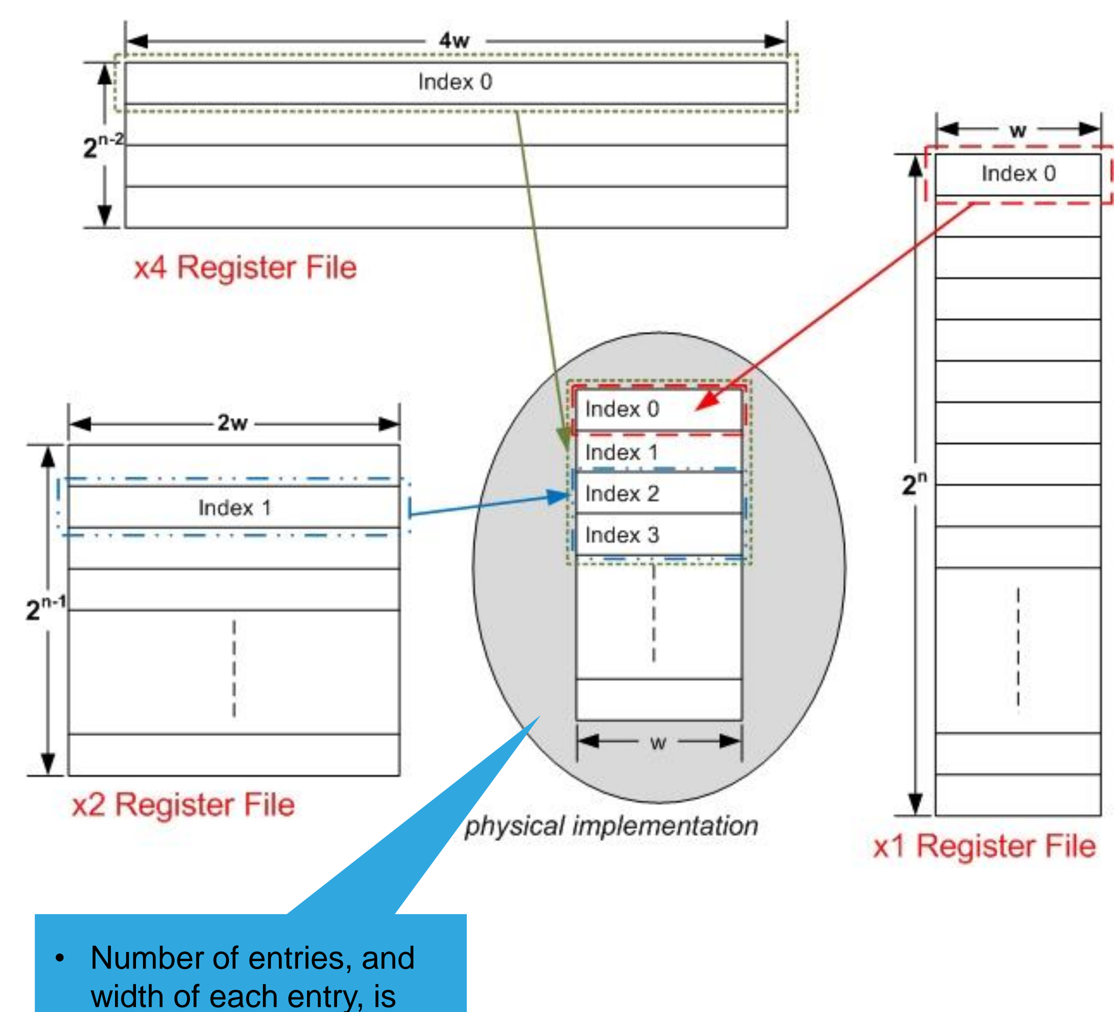
Register Files

- In a load-store machine the register file architecture is extremely important
  - Most operands to computation will come from them in one form or another
- Now there is great flexibility in the architecture, size and depth of regfiles
  - Can be very wide (> 1024bit)
  - Can have arbitrary numbers of registers > 1 … you want an 11-entry 73bit regfile ? You got it!
  - Can have up to 32 (count ‘em) read ports and 16 (“are you sure?!”) write ports
    - Remember that currently all read/write ports are fully bypassed … you would not actually want to reach these limits because the bypass networks would likely be enormous ☺
  - Can be linked into “virtual” register files in pairs or quads
    - Make a 16-entry 32-bit regfile usable as a 8-entry 64-bit or 4-entry 128-bit
    - Opens up new possibilities for DSP processing
- From the programmer’s viewpoint, register files are where “custom ctypes” live
  - The compiler will manage scheduling, load/spill, allocation ….
- You can have a total of 24 separate register files in an Xtensa

© 2016 Cadence Design Systems, Inc. All rights reserved.

Since (RE.2)

Register Groups: 1 implementation but 3 virtual views



- A user regfile can be used as three virtual regfiles of different widths
  - x1: 2^n entries, each w wide
  - x2: 2^{n-1} entries, each $2w$ wide
    - Implicitly refers to 2 consecutive entries of physical implementation
  - x4: 2^{n-2} entries, each $4w$ wide
    - Implicitly refers to 4 consecutive entries of physical implementation
- All three virtual regfiles use same physical hardware
  - XCC ensures no overlaps when allocating register indices for variables
- x1 virtual regfile is always available
- x2 and x4 virtual regfiles need to be enabled explicitly in TIE
  - Either or both can be enabled

© 2016 Cadence Design Systems, Inc. All rights reserved.

Register Groups
Syntax

```
regfile CR1 8 8 CR CR2=2 CR4=4

// ctype definition for CR register file
ctype xx_char 8 8 CR1

// ctype definition for CR2 register file
ctype xx_short 16 16 CR2 { xx_char hi, xx_char lo }

// ctype definition for CR4 register file
ctype xx_int 32 32 CR4 { xx_char a, xx_char b, xx_char c, xx_char d }

// Operation that uses register groups
operation { out CR2 result, in CR2 dataA, in CR1 data } { } { .. }
```

Customer can optionally create register groups by specifying the number of consecutive entries. Syntax is:

group-name=[2|4]

- *At least one ctype that is as wide as the register group must be declared*
- *The ctype must be a "struct ctype" that references ctypes of primary register file.*

Operations can directly use register groups as operands

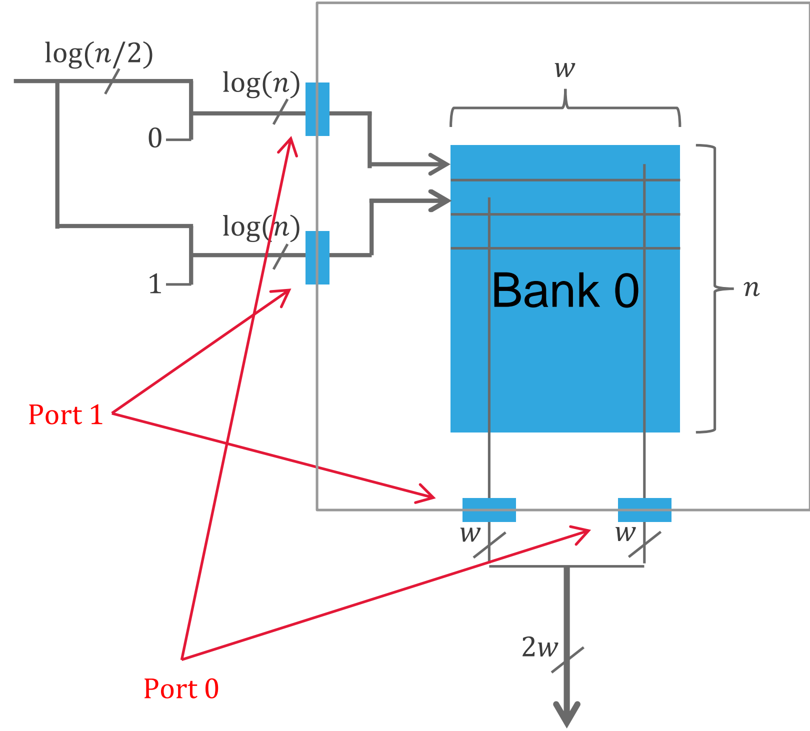
© 2016 Cadence Design Systems, Inc. All rights reserved.

Physical Implementation Choice

MULTI-PORT Implementation

Figure shows a single read port on a register file with x2 and x4 register groups

regfile CR1 8 32 cr CR2=2 CR4=4



Legend
w – physical width of each register
n – number of entries in register file

- The regfile is implemented as a single bank *w*-wide and *n*-deep
- Each regfile port is *w*-wide
- Regfile ports used per operand is variable
  - x1 regfile access uses 1 port
  - x2 regfile access uses 2 ports
  - x4 regfile access uses 4 ports
- Can potentially increase ports on regfile
  - Impacts area (regfile core and read/write pipelines)
  - Could adversely impact timing

© 2016 Cadence Design Systems, Inc. All rights reserved.

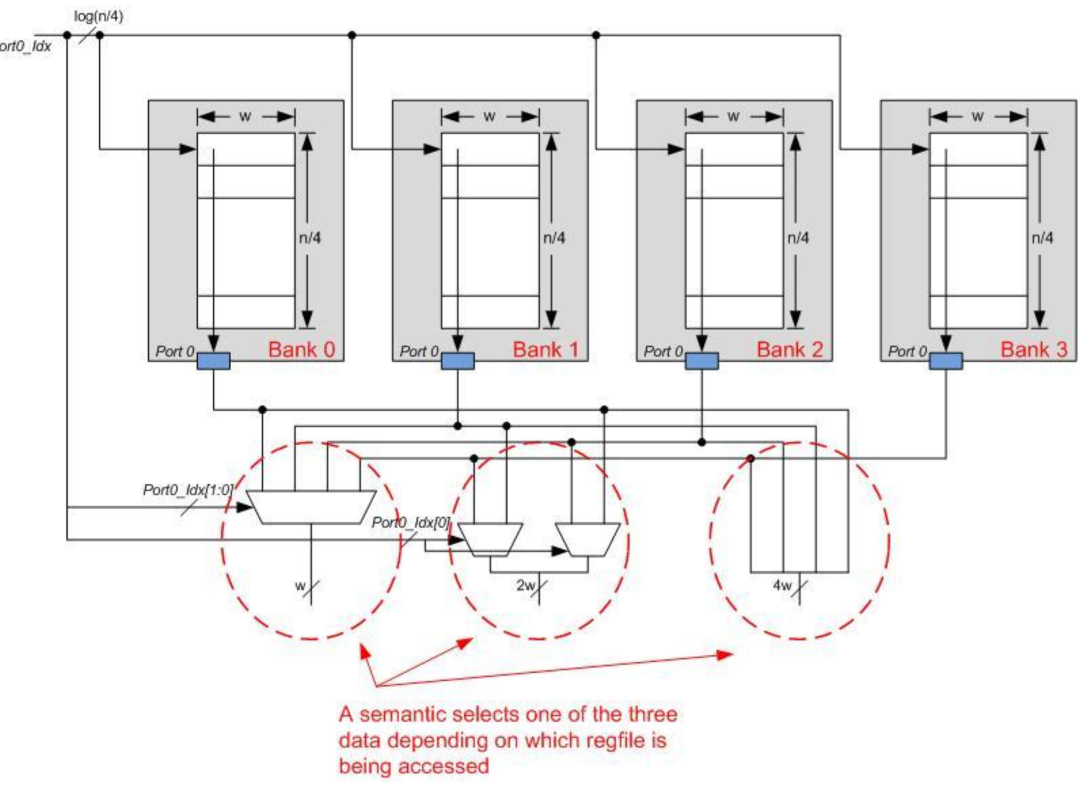
Physical Implementation Choice

VIEWS Implementation

- Regfile is composed of banks of equal depth
  - E.g. the regfile in figure has 4 banks, each 8-wide and 8-deep
- Regfile indices are interleaved across banks
  - E.g., Index 0 is in Bank 0, Index 1 is in Bank 1, and so on…
- Wider data is a concatenation of multiple banks
  - E.g., x2 data is a concatenation of data on Banks 0 and 1, or 2 and 3

Figure shows a single read port on a register file with x2 and x4 register groups

regfile CR1 8 32 cr CR2=2 CR4=4



© 2016 Cadence Design Systems, Inc. All rights reserved.
cadence®

FLIX Support

Flexible Length Instruction eXtension

```
format f64 64 {slot0, slot1}
slot_opcodes slot0 {ADD, AND}
slot_opcodes slot1 {SUB, XOR}
```

- First iteration (circa 2003) – Single 64-bit FLIX length allowed
- Since then …
  - Gradual relaxation of limits, shorter FLIX (32b) → longer FLIX (128b), more FLIX (2 lengths allowed, then more …)
- Implementation Limits (rarely reached):
  - Max Formats → 24
  - Max slots → 64
  - Max slots (in 128b format) → 30
  - Format lengths → 32 <= Format Length <= 128 (integer #bytes)
- It's pretty Flexible ☺
- Watch out – you can create a monster of an instruction decode unit
  - Remember KISS !

© 2016 Cadence Design Systems, Inc. All rights reserved.

Sharing Hardware

- Shared functions
  - Very easy to use, can share hardware "per slot" or globally
  - Useful for quick "what-if" analysis, but not the best for "final implementations"
    - there isn't time here to dig into "why", just take my word on it ☺
- Semantics are the way to go
  - What we use ☺
  - Can be multi-cycle, can implement the HW for 100's of instructions
    - Which means they can be very complicated and difficult to write/debug → need a structured design flow
- NOW Semantics can be SHARED across slots in a FLIX machine
  - Still only one copy of the hardware
  - More flexibility in HW design for the "mix" of instructions the compiler uses
  - More efficient than using shared functions.
  - See section 20.6 in RF.3 TIE Reference Manual

© 2016 Cadence Design Systems, Inc. All rights reserved.

Protos – what are they and why do they matter ?

- Essentially the mechanism for connecting xt-xcc to your hardware
- Used to define the programming model
  - E.g. operator overloading, type conversions, default types, instruction fusing (technically that's a "imap")
- Can be single instructions, or sequences

```
proto BBE_SBMULSTATE { in xb_vecNx16 * a }{xb_vecNx16 t0, xb_vecNx16 t1, xb_vecNx16 t2, xb_vecNx16 t3}{
    BBE_MOVVBMULSTATE t0, 0;
    BBE_SVNX16_I t0, a, 0;
    BBE_MOVVBMULSTATE t1, 1;
    BBE_SVNX16_I t1, a, 32;
    BBE_MOVVBMULSTATE t2, 2;
    BBE_SVNX16_I t2, a, 64;
    BBE_MOVVBMULSTATE t3, 3;
    BBE_SVNX16_I t3, a, 96;
}
```

- Greatly enhance the programmer's experience – read up on them

© 2016 Cadence Design Systems, Inc. All rights reserved.

TIE compiler reports

- When you compile some TIE code, tc creates some reports
  - And lots of other things … they are all in the tdk/ directory

- Area estimates
- Power estimates
- Report on operand, regfile and FLIX design

- All useful, note the choice of word … “estimate”
  - Do not start writing your marketing material based on these numbers ☺

© 2016 Cadence Design Systems, Inc. All rights reserved.

Tdk .report – some highlights

- Very useful regfile information - #ports and operand schedules
  - There is now an advanced TIE training class that goes into why this is important

```
### TIE Report File

# These are the instructions in each slot
slot <Inst> instructions: nop=NOP
    MB_MOVMBVMBV
slot <fusion_slot0> instructions: nop=NOP
    MB_L64_I MB_S64_I
slot <fusion_slot1> instructions: nop=NOP
    POP_COUNT128_64 POP_COUNT64_16 POP_COUNT64_32 POP_COUNT64_8


### Register File port information
# These are the read and write ports for each regfile.
regfile AR:

        port rd0: width( 32) stage( 1 2) operands( {ars,0} {opnd_ae_sem_ar_to_dr_a0,0})
        port rd1: width( 32) stage( 1 2 3) operands( {art,0} {opnd_MB_L64_I_address,0} {opnd_MB_S64_I_address,0} {opnd_ae_sem_ar_to_dr_a1,0} {opnd
        port rd2: width( 32) stage( 1 2) operands( {art,1})
        port rd3: width( 32) stage( 1 2) operands( {ars,1})
        port wr0: width( 32) stage( 1 2) operands( {ar0,0} {ar12,0} {ar4,0} {ar8,0} {arr,0} {ars,0} {ars_entry,0} {art,0} {opnd_ae_sem_dr_to_ar_a,
        port wr1: width( 32) stage( 1 3) operands( {stage3_ar_operand,0} {arr,1} {art,1} {opnd_AE_CALCRNG3_a,1} {stage3_ar_operand,1})
regfile BR:
        port rd0: width( 1) stage( 1) operands( {bs,0})
        port rd1: width( 1 4 8 16) stage( 1 2) operands( {brall,0} {bs4,0} {bs8,0} {bt,0})
        port rd2: width( 1 2 4) stage( 2) operands( {bt2,0} {bt,1} {bt2,1} {bt4,1})
        port wr0: width( 1 2 4 16) stage( 1 2) operands( {br,0} {br2,0} {brall,0} {bt,0} {br,1} {br2,1} {br4,1})
regfile AE_DR:

        port rd0: width( 64) stage( 2) operands( {frs,0} {opnd_ae_sem_arithmetic_v,0} {opnd_ae_sem_arithmetic_v0,0} {opnd_ae_sem_cmov_v0,0} {opnd_
        port rd1: width( 64) stage( 2 3) operands( {frt,0} {opnd_ae_sem_arithmetic_v1,0} {opnd_ae_sem_cmov_v,0} {opnd_ae_sem_cmpp_v0,0} {opnd_ae_s
        port rd2: width( 64) stage( 2) operands( {frs,1} {opnd_ae_sem_mul_x2_S1_d0,1})
        port rd3: width( 64) stage( 2) operands( {frr,0} {frr,1} {opnd_AE_ZEROB_v0,1} {opnd_ae_sem_arithmetic_v1,1} {opnd_ae_sem_arithmetic_va,1}
        port wr0: width( 64) stage( 2 3 5) operands( {frr,0} {opnd_ae_sem_ar_to_dr_v,0} {opnd_ae_sem_arithmetic_v,0} {opnd_ae_sem_cmov_v,0} {opnd_
        port wr1: width( 64) stage( 2 3 5) operands( {frr,1} {opnd_POP_COUNT128_64_res,1} {opnd_ae_sem_ar_to_dr_v,1} {opnd_ae_sem_arithmetic_v,1}
```


© 2016 Cadence Design Systems, Inc. All rights reserved.

cadence

Tdk .report – flops inferred in semantics

- Will be due to multi-cycle semantics
- Typically "wide" pipeline flops should be only one pipe stage deep
- Useful tool for spotting "inefficient" schedules …

```
### Report the generated flops in each semantic

Semantic: MB_L64_I
    32 * 1  opnd_MB_L64_I_offset: 0 -> 1

Semantic: MB_S64_I
    32 * 1  opnd_MB_S64_I_offset: 0 -> 1

Semantic: POP_COUNT128_64
    1 * 2   POP_COUNT128_64: 0 -> 2

Semantic: popc_sem
    1 * 2   POP_COUNT64_8: 0 -> 2
    1 * 2   POP_COUNT64_16: 0 -> 2
    1 * 2   POP_COUNT64_32: 0 -> 2
    1 * 2   TIE_inst_cp1: 0 -> 2
```

- There are many other useful pieces of info in the .report files …

© 2016 Cadence Design Systems, Inc. All rights reserved.

TIE constructs – Honourable Mentions

Interesting features to read up on

- TIE property “shared_or” and “ignore state output”
  - Allows creation of “sticky bits” in your FLIX design (TIE RM section 4 and 20.2)
- Bitkill – allows partial update of output operands
  - no need to create inouts where not needed (TIE RM 7.4, 7.5)
- Specialized_op – tell the compiler two instructions only differ e.g. by immediate
  - Useful when you have a “large immediate” and “small immediate” version of an instruction
  - Compiler will select the most efficient based on immediate value and code optimisation (TIE RM 20.1)
- Header_include – embed useful header info in the TIE file
  - TIE RM 20.9 – Provide a header “automatically” along with your TIE headers
- TIEprint() – diagnostics for ISS simulation
  - This is purely a simulation artefact – useful for debugging (TIE RM 26)

© 2016 Cadence Design Systems, Inc. All rights reserved.

Tools / Debug

© 2016 Cadence Design Systems, Inc. All rights reserved.

Performance Monitors

Non-intrusive counting of events in a real HW target

- Performance Monitors - hardware counting of events on the updated trace port
- When to use:
  - When ISS/XTSC system model is not enough to measure performance
    - Not enough of the system is modeled
    - Model is not accurate enough
    - Anything you have to have Silicon or FPGA
  - In-Field performance analysis, debugging
- Use Cases:
  - Statistical Profiling provides an overview
    - Overall performance profiling of target software with large data sets
    - Identify inefficient functions or sections in the code
  - Details function analysis with direct counter access
    - Detailed analysis of individual functions or sections in the code
    - Identify inefficient constructs
    - Identify reasons for inefficiency due to stalls, cache misses,…

© 2016 Cadence Design Systems, Inc. All rights reserved.

What is counted ?

Up to 8 32-bit counters software programmable

- There are almost 100 countable hardware events, in **`SELECT`** groups listed below
- Select Groups:
  - Always Increment (counts cycles)
  - Overflow of Counter N-1
  - Successfully Retired Instructions
  - D-Side Global Stalls
  - I-Side Global Stalls
  - Exceptions and Replays
  - Holds and other Bubbles
  - I-TLB Access
  - I-Side Memory Accesses
  - D-TLB Access
  - Data Memory Load Instruction
    - LS0,LS1
  - Data Memory Stores Instruction
    - LS0,LS1
  - Data Memory Accesses
    - LS0,LS1
  - Multiple Loads/Stores
  - Outbound Prefetch/Castout
  - Inbound I or D Target
  - Prefetch

© 2016 Cadence Design Systems, Inc. All rights reserved.

Use the provided HAL APIs

- Always best to use the (supported) HAL APIs to access these types of functions

- See debug guide chapter 8
- See Software Toolkit userguide chapter 6

- The above give descriptions how to do profiling using these counters

- Remember → They are on the trace port, they are NON-INTRUSIVE

© 2016 Cadence Design Systems, Inc. All rights reserved.

Code Coverage

Built into Xplorer

popcount_TIE.c | pop_count2.tie | utils.c | »10

```
{
    int i, retval =0 ;
    for(i=0; i<count; i++)
    {
        if(goldPtr[i] != dataPtr[i])
        {
#if (VERBOSE > 1)
            printf("CV8:: Iter %d  Gold--> %d  Data--> %d\n", i, gold
#endif
            retval++ ;
        }
    }
    return retval ;
}
int compare_vec16(uint16_t *goldPtr, uint16_t *dataPtr, int count)
{
    int i, retval =0 ;
    for(i=0; i<count; i++)
    {
        if(goldPtr[i] != dataPtr[i])
        {
#if (VERBOSE > 1)
            printf("CV16:: Iter %d  Gold--> %d  Data--> %d\n", i, go
#endif
            retval++ ;
        }
    }
    return retval ;
}
int compare_vec32(uint32_t *goldPtr, uint32_t *dataPtr, int count)
{
    int i, retval =0 ;
    for(i=0; i<count; i++)
    {
        if(goldPtr[i] != dataPtr[i])
        {
```

Outline

- VERBOSE
- stdint.h
- stdio.h
- compare_vec8(uint8_t*, uint8_t*, int) : int
- compare_vec16(uint16_t*, uint16_t*, int) : int
- compare_vec32(uint32_t*, uint32_t*, int) : int
- init_vec8(uint8_t*, int) : void
- init_vec16(uint16_t*, int) : void
- init_vec32(uint32_t*, int) : void

Help

Contents | Search | Related Topics | Bookmarks | Index

Search expression:

Code Coverage view | Go

Scope Default

Local Help (3 hits)

Code Coverage View

Code Coverage View The Code Coverage View displays code coverage statistics (lines of code in each file or function, lines of code executed

Code Coverage | File | Fun

File	% Covered	Total LOC	Lines Executed	Lines Missed
<TOTAL>	79.2	192	152	40
D:/customers/Fingerprint_Systems/workshop_jan2016/fuse_wsf3/...	53.8	39	21	18
D:/customers/Fingerprint_Systems/workshop_jan2016/fuse_wsf3/...	59.3	54	32	22
D:/customers/Fingerprint_Systems/workshop_jan2016/fuse_wsf3/...	100.0	43	43	0
D:/customers/Fingerprint_Systems/workshop_jan2016/fuse_wsf3/...	100.0	24	24	0
D:/customers/Fingerprint_Systems/workshop_jan2016/fuse_wsf3/...	100.0	32	32	0

© 2016 Cadence Design Systems, Inc. All rights reserved.

Quick points on code coverage

- Provides statistics on lines of source code executed over one or more runs
- Good for verifying complete exercising of all lines of source code (e.g. testing) or any areas of dead code that need to be removed
- Only supported in Xplorer, no command-line version is available
- Uses feedback data collection (code instrumenting) to check coverage
- Coverage can be checked using either software or hardware feedback mechanisms
- Better results at lower optimization levels
- Coverage under IPA not supported
- Runs are cumulative, so multiple runs may be done under different scenarios to verify complete coverage
- See Xplorer online help for details

© 2016 Cadence Design Systems, Inc. All rights reserved.

Summary

- Hopefully some points of interest – thanks for listening!
- Xtensa is a wide topic – there's a lot to know
- There is an increasing amount of training going to the iLS system
- There are user groups
- There will be lunch … shortly ☺

© 2016 Cadence Design Systems, Inc. All rights reserved.

cadence®