cādence®

Processor Configuration Build

Processor: hifi3_hikey960
ISA: LX6.0
Release: RG-2017.5
Build ID: 464161

Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Xtensa Release: RG-2017.5
Issue Date: 02/2018

Modification: 464161

Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Contents

List of Tables

List of Figures

1. Processor Configuration Overview

Topics:

- *Summary*

Overview summary

Concise summary in the same form as the configuration HTML page.

1.1 Summary

Table 1: Overview

Name	Value
Configuration Name	**hifi3_hikey960**
Description	hifi3 hikey960 RF2 build
Built on XPG	2018-02-10 00:36:11 PST
XPG Release	RG-2017.5
XPG Build ID	464161
Target HW Build ID	448360
Target Hardware Version	LX6.0.2
Platforms Built	linux,win32

Table 2: User TIE

Name	Value
TIE Name	No user TIE

Table 3: HiFi 3 AudioEngine Options

Option	Selection
HiFi3 Audio Engine DSP coprocessor instruction family	Selected
> HiFi3 Vector FP	Not Selected

Table 4: Target Software Options

Option	Selection
C Libraries	Newlib C Library
Software ABI	windowed
Hardware Floating Point ABI	Not Selected
Xtensa Tools should use Extended L32R	Not Selected
Build with reset handler at alternate reset base	Not Selected

Table 5: Implementation Options

Option	Selection
Global Clock Gating	Selected
Functional Unit Clock Gating	Selected
Asynchronous Reset	Selected
Register file implementation block	Flip-flops
Full scan	Selected
Size of L0 Loop Buffer (in bytes)	256
Semantic Data Gating selection	All
Enable Memory Data Gating	Selected
Power Shut-Off Domains selection	None
Power Shut-Off Core Retention selection	None

Table 6: Memory Management Options

Option	Selection
Memory Protection/MMU	Region protection with translation

Table 7: Arithmetic Options

Option	Selection
MUL32 implementation selection	Pipelined + UH/SH
MUL16	Selected
16-bit MAC with 40 bit Accumulator	Selected
CLAMPS	Selected
32 bit integer divider	Selected
Single Precision FP (coprocessor id 0)	Not Selected
Single+Double Precision FP (coprocessor id 0)	Not Selected
Non-IEEE Double Precision Floating Point Accelerator	Not Selected

Table 8: ISA Instruction Options

Option	Selection
NSA/NSAU	Selected
MIN/MAX and MINU/MAXU	Selected
SEXT	Selected
DEPBITS	Not Selected
Enable Density Instructions	Selected
Boolean Registers	Selected
Enable Processor ID	Selected
TIE arbitrary byte enables	Selected
Zero-overhead loop instructions	Selected
Synchronize instruction	Selected
Conditional store synchronize instruction	Selected
Number of Coprocessors	2
Miscellaneous Special Register count	2
Thread Pointer	Not Selected

Table 9: ISA Configuration Options

Option	Selection
Number of AR registers for call windows	64
Byte ordering (endianness)	Little Endian
Generate exception on unaligned load/store address	Handled by hardware
Max instruction width (bytes)	8
L32R hardware support option	Normal L32R
Pipeline length	5

Table 10: BUS / Bridge Options

Option	Selection
Processor Interface + External Bus selection	AXI3
Asynchronous AMBA bridge	Not Selected

Option	Selection
Request Control Depth	4
Request Data Depth	8
Response Depth	8

Table 11: PIF Configuration Options

Option	Selection
Write buffer entries	16
Prioritize Load Before Store	Not Selected
Enable PIF Write Responses	Selected
Inbound PIF request buffer depth	4
PIF Request Attributes	Not Selected
Enable PIF Critical Word First	Selected
PIF Arbitrary Byte Enables	Selected
Enable Early Restart	Selected

Table 12: Prefetch Options

Option	Selection
Cache Prefetch Entries	8
Enable Prefetch Directly to L1	Not Selected

Table 13: Interface Width Options

Option	Selection
Width of Instruction Fetch interface	64
Width of Data Memory/Cache interface	64
Width of Interface to instruction cache	64
Width of PIF interface	64

Table 14: Port and Queue Options

Option	Selection
GPIO32: 32-bit GPIO interface	Selected
QIF32: 32-bit Queue Interface	Not Selected

Table 15: Memory Error Selections

Option	Selection
Instruction Memory Error type	None
Data Memory Error type	None

Table 16: Instruction Cache

Option	Selection
Instruction Cache size (Bytes)	65536
> Associativity	4
> Line size (Bytes)	128
> Line Locking	Selected
> Instruction Cache memory error	Not Selected
> Dynamic Way Disable	Not Selected

Table 17: Data Cache

Option	Selection
Data Cache (Bytes)	65536
> Associativity	4
> Line size (Bytes)	128
> Write Back	Selected
> Line Locking	Selected
> Data Cache memory error	Not Selected
> Number of Data Cache Banks	1
> Dynamic Way Disable	Not Selected

Table 18: Local Memories

Memory	Size	Address	Inbound PIF	Busy
Instruction RAM 0	32K	0xe8080000	Selected	Not Selected
Instruction RAM 1	16K	0xe8088000	Selected	Not Selected
Data RAM 0	32K	0xe8058000	Selected	Not Selected
Data RAM 1	128K	0xe8060000	Selected	Not Selected

Table 19: Load / Store

Option	Selection
Count of Load/Store units	1
Connection box	Not Selected

Table 20: DataRAM Options

Option	Selection
iDMA	Not Selected

Table 21: Debug

Option	Selection
Debug	Selected
> Instruction address breakpoint registers	2
> Data address breakpoint registers	2
> On Chip Debug(OCD)	Selected
> Enable APB Debug Access	Selected
> External Debug Interrupt	Selected
> Number of Performance Counters	4

Table 22: Trace

Option	Selection
Trace port (address trace and pipeline status)	Selected
Add data trace	Not Selected

Option	Selection
TRAX Compressor	Included
Size of trace memory (bytes)	4096
TRAX ATB data interface	Not Selected
Enable sharing of TRAX memories	Not Selected

Table 23: Interrupts Overview

Option	Selection
Interrupt count	32
> Count of interrupt priority levels	5
> Timer count	2
> EXCM priority level (highest priority of efficiently C-callable handlers)	3
> Debug interrupt level	5

Table 24: Interrupts Details

Interrupt	Type	Level	BInterrupt Pin
0	nmi	nmi	0
1	sw	3	
2	level	3	1
3	level	3	2
4	level	3	3
5	timer.0	3	
6	timer.1	4	
7	level	3	4
8	level	2	5
9	level	2	6
10	level	2	7
11	level	2	8
12	level	1	9

Interrupt	Type	Level	BInterrupt Pin
13	level	1	10
14	level	1	11
15	level	1	12
16	level	1	13
17	level	1	14
18	level	1	15
19	profiling	3	
20	level	1	16
21	level	1	17
22	level	1	18
23	level	1	19
24	level	1	20
25	level	1	21
26	level	1	22
27	level	1	23
28	level	1	24
29	writeerr	3	
30	level	1	25
31	level	1	26

Table 25: System Memories

Memory	Base Address	Size
System RAM	0xc0000000	256M
System ROM	0xd0000000	16M

Table 26: Vector Options

Option	Selection
Automatically position vectors	Not Selected
Vector Layout Style	Xtensa Relocatable

Option	Selection
Enable Relocatable Vectors	Selected
Alternate Static Vector Base Address	0xc0000000
Default Dynamic Vector Group VECBASE	0xe8080400

Table 27: Static Vectors

Vector	In Memory	Address	Prefix Bytes	Size Bytes
Reset vector	Instruction RAM 0	0xe8080000	0x0	0x300

Table 28: Dynamic Vectors

Vector	In Memory	Address	Prefix Bytes	Size Bytes
Window vector base	Instruction RAM 0	0xe8080400	0x0	0x178
Level 2 vector	Instruction RAM 0	0xe8080580	0x8	0x38
Level 3 vector	Instruction RAM 0	0xe80805c0	0x8	0x38
Level 4 vector	Instruction RAM 0	0xe8080600	0x8	0x38
Level 5 vector (Debug)	Instruction RAM 0	0xe8080640	0x8	0x38
NMI vector	Instruction RAM 0	0xe80806c0	0x48	0x38
Kernel vector	Instruction RAM 0	0xe8080700	0x8	0x38
User vector	Instruction RAM 0	0xe8080740	0x8	0x38
Double vector	Instruction RAM 0	0xe80807c0	0x48	0x40

2. Processor Configuration Options

Topics:

Descriptions of configured processor configuration options

This includes all options configured for this particular LX6.0 processor. Refer to the Xplorer help for descriptions of all available configuration options.

2.1 Processor Selections

Processor and Coprocessor Options

The main set of processor and coprocessor options.

Coprocessor selections for this build are included in this section.

2.1.1 HiFi 3 Audio Engine option

HiFi 3 Audio Engine DSP coprocessor instruction family

HiFi3 Audio Engine DSP coprocessor instruction family	Selected
HiFi3 Vector FP	Not Selected

Cadence HiFi 3 Audio Engine is a highly optimized audio processor geared for efficient execution of audio and voice codecs and pre- and post-processing modules. It goes beyond the two MAC, two issue, HiFi 2/EP architecture with four multipliers, three VLIW slots, good support for 32x16-bit and 32x32-bit multiplication, a true 64-bit data path and native support for ITU-T/ETSI intrinsics. The extra resources provide for significant performance improvements compared to HiFi 2/EP, particularly on pre/post-processing algorithms as well as voice codecs. The support for 32-bit audio as well as ITU-T/ETSI intrinsics, including automatic vectorization, provides much better performance on out-of-the-box C programs and voice algorithms.

HiFi 3 is backward compatible at the C/C++ source level with HiFi 2/EP. Any algorithm written in C/C++, including all HiFi 2/EP packages from Cadence, can simply be recompiled on HiFi 3 and will get modest performance improvements. For maximum performance, key kernels may need to be retuned for the HiFi 3 architecture.

All HiFi 3 Audio Engine operations can be used as intrinsics in standard C/C++ applications. In addition, when compiling with automatic vectorization or with the -mcoproc option, the compiler will automatically use HiFi 3 operations when compiling standard C code.

Cadence HiFi 3 Audio Engine consists of two main components: a DSP subsystem and a subsystem to assist with bit stream access and variable-length (Huffman) encoding and decoding.

2.2 Software Configuration Options

Configuration options which only affect the target software

Software configuration options require a new processor configuration, but do not affect the generated hardware. Therefore, options can be selected after a hardware design has been completed. The initial choices set in the software pane of the processor generator when creating a configuration will used to generate matching diagnostics with the HW package.

You can easily explore alternatives by building the HW + SW one way, and then building "software upgrades" of the original configuration with different combinations of target software options. "Upgrades" can be variants built with the same XPG release, or they can be built with newer XPG releases.

2.2.1 C and Math Libraries

Cadence offers the choice of three C and math libraries: `newlib` from Red Hat, Inc., the Xtensa C library and `uClibc`

C Libraries Selection	Newlib C Library

You choose between the libraries when building your software configuration through the Xtensa Processor Generator. The libraries cannot be mixed.

- The `newlib` library is more complete, fully documented, higher performance and supports reentrancy for multi-threaded environments.
- The Xtensa C library has similar performance to newlib and is smaller. It strictly implements the C library as defined by the C standard and hence may not implement all the extensions supported by `newlib`. The philosophy of the library is standards compliance and simplicity. So, for example, the malloc routine is simple and hence fast but might cause more memory fragmentation on programs that extensively malloc and free. The Xtensa C library places no open source restrictions *on the C user (there are minor restrictions for the C++ user)*.
- `uClibc` is significantly smaller. `uClibc` can be configured with or without support for floating point. Without floating point support, it is not possible, for example, to print floating point numbers and it is not possible to use C++ I/O streams, but the resultant library is significantly smaller still. Note that `uClibc` comes with more restrictive open source licensing requirements than even `newlib`.

Review your contract or the files in the `XtensaTools/misc` directory for details about the various licensing requirements.

2.2.2 Application Binary Interfaces

AR Registers Count	64
ABI Selection	windowed

Cadence offers the choice of two Application Binary Interfaces (ABIs) for Xtensa X and LX processors: the windowed ABI and the CALL0 ABI. Xtensa TX processors only support the CALL0 ABI because they only have 16 AR registers. With the windowed ABI, each function call is implemented using a CALL4, CALL8 or CALL12 instruction that rotates the Xtensa register windows and thereby immediately gives the called function a set of extra scratch registers. Without the windowed ABI, each function call is implemented using a CALL0 instruction and the compiler must typically save and restore to memory scratch variables used by the callee. Application code compiled using CALL0 is typically 5-10% larger than application compiled using the windowed ABI. Performance of loop intensive code is marginally slower with CALL0 while more call intensive code is up to 10% slower. At time of writing, CALL0 is only supported with the ThreadX RTOS from Express Logic or with the XTOS runtime from Cadence.

Given these characteristics, most will use the windowed ABI. However, there are also advantages to the CALL0 ABI. The CALL0 ABI enables hardware configurations with only 16 AR registers, thereby allowing significantly smaller hardware configurations. Interrupt and context switching latency is lower with CALL0 than with the windowed ABI. Using the CALL0 ABI, you can manually rotate the register files in a single cycle in special code or interrupt handlers for very fast specialized context switching.

An application cannot mix the two ABIs. However, it is possible to use the windowed ABI for an application and CALL0 for certain high priority interrupts. The use of CALL0 in this context enables interrupt handlers to be written in C without the higher overhead of saving and restoring all the AR registers.

Related Links

AR Registers Count on page 33
Number of physical AR registers. Setting to 16 registers means windowed calls are not supported

2.2.3 Build with Reset Handler at Alternate Reset Base

Use Alternate Reset Base option	Not Selected
Alternate Static Vector Base Address	0xc0000000

For processor configurations that support relocatable vectors, at configuration time, a primary and alternate "static vector group base address" can be configured. This address is a base from which the reset vector and memory error vector (if configured) are offset. Which address (primary / alternate) is used at processor reset is controlled by an input pin which can be asserted to select the alternate base. By default, the software configuration build will assume the primary static base is used, and will generate reset code for those addresses. This option chooses whether to build software with the reset code at the alternate address.

2.2.4 RTOS Compatibility Option

Generic RTOS Compatibility - ensures selection of a set of features required by many RTOSes

RTOS Compatibility option	Not Selected

This option does not have any direct effect on processor software or hardware; it is a compatibility checking option to help avoid configuration omissions that might have later impact on what software can run on the processor.

2.3 Implementation Options

Options which affect the physical implementation

2.3.1 Global Clock Gating

Select whether global clock gating should be enabled

Global Clock Gating	Selected

Enables first level of clock-gating that is based on global conditions, which can turn off most Xtensa clocks for low-power applications. Please refer to the appropriate *Xtensa Microprocessor Data Book* for more information

2.3.2 Functional Unit Clock Gating

Select whether clock gating should be enabled at the functional unit level

Functional Unit Clock Gating	Selected

Allows a second level of clock gating in which individual units, that are not used, are turned off via clock-gating while the Xtensa processor is still active.

2.3.3 Asynchronous Reset

Asynchronous Reset	Selected

The Xtensa processor can optionally be configured to use asynchronous reset registers. In the default case (synchronous), the BReset input is used to synchronously reset flip-flops in the processor core. If this option is selected, then the BReset input is used to asynchronously reset flip-flops in the processor core.

2.3.4 Full Scan option

Creates a scan-enabled design that supports scan insertion

Full Scan option	Selected

When a core is configured with "Full scan" option, a TMode pin and a TModeClkGateOverride pin are added to the RTL at top-level.

TMode is expected to be asserted during entire scan testing and it enables testability in 4 areas:

1. It enables async. reset pin to bypass synchronization logic so tester can directly control the reset of flops during entire scan test.
2. It inverts the clock(the JTAG clk) of the falling-edge triggered flip-flop in JTAG logic during testing so that there are only rising-edge triggered flops (i.e no falling-edge edge triggered flops) in the design during entire scan test.

3. It overrides the enable pin of latches in latch-based register files so that latches are transparent (only applicable to older Xtensa cores with latch-based register file) during capture phase of scan test.
4. It bypasses the reset that is generated by test-logic-reset state of the TAP, to use JTRST instead, if JTAG TAP is configured. This allows direct control (instead of going through sequential logic) of the reset pin of flops that uses FSM-generated reset.

The TModeClkGateOverride is typically only asserted during shift phase of scan test and it enables testability as follows: It overrides the clock-gating enable pin of clock-gating cell. I.e. it disables clock-gating so clock is always turned on during shift phase of scan test.

2.3.5 Size of L0 Loop Buffer option

Configures an L0 Loop Buffer to cache loop instructions to save loop power by avoiding I-memory accesses

Size of L0 Loop Buffer option	256

This creates an L0 Instruction Loop Buffer that captures instructions from the body of a Zero Overhead Loop, and then executes subsequent iterations of the loop from this buffer. This allows the instruction memories to not be enabled during most of the loop execution. Power savings will depend on what portion of code is being executed as Zero Overhead Loops.

Related Links

Zero-Overhead Loops Option on page 32
Enable zero-overhead loop instructions (eliminates loop pipeline overhead)

2.3.6 Semantic Data Gating option

Allows the insertion of data gates on the inputs of TIE semantics to prevent unnecessary toggling on semantics not currently in use by the core

Semantic Data Gating option	All

In both user-written and core TIE blocks, the output of register files and states fan out to several parallel TIE semantic logic blocks. At any given time, not all of these semantics are in use. If the semantic data gating option is configured, data gates with the appropriate enables are inserted in front of these semantic logic blocks to prevent unnecessary toggles. Two selections enable data gating:

1. If "all" is selected, then all user-defined TIE semantics and a Cadence-determined optimal list of core TIE semantics are data gated.
2. If "user" is selected, then only user-defined TIE semantics with the data_gate property are gated, as well as a Cadence-determined optimal list of core TIE semantics.

Although the Xtensa core will lower memory enable signals when a particular memory is not in use, the outbound address and data lines will still toggle. Data gating these signals can save idle-cycle memory dynamic power. If the memory data gating option is configured, data gates with the appropriate enables will be inserted on all instruction and data memories.

2.3.7 Memory Data Gating option

Gates the Addr and WrData inputs to the memories

Memory Data Gating option	Selected

Allows the insertion of data gates on the outbound address and data lines of both instruction and data memories, saving idle-cycle memory dynamic power. Memory data gating applies to all configured memories equally.

2.4 Instruction / ISA Options

Instruction and ISA options

Sub-sections describe the available instruction and ISA configuration options.

2.4.1 Memory Management Selection

Region protection options provide coarse protection at a low gate count The Full MMU provides resource management capabilities sufficient for running Linux

Memory management selection	Region protection with translation

LX and Xtensa processor configurations support several types of memory management:

- XEA2 Region protection
- XEA2 Region protection with translation
- XEA2 Full MMU with TLBs
- XEA2 MPU (Memory Protection Unit)

Refer to the *Xtensa Microprocessor Data Book* for more information on the XEA2 memory management options.

2.4.2 Arithmetic Instruction Options

The Xtensa Processor Generator offers a series of optional arithmetic configuration options.

These are all selected from the Instructions page of the Xplorer Configuration Editor. Refer to the Architectural Options chapter in the *Xtensa Instruction Set Architecture (ISA) Reference Manual* for more details.

2.4.2.1 MUL32 Option

MUL32 selection	Pipelined + UH/SH

This option selects a standard 32-bit multiplier, which is used by the compiler whenever multiplying signed or unsigned variables of type int, short or char.

Fully Pipelined Implementation

This option creates a MULL instruction that implements a 32-bit times 32-bit multiplication into a 32-bit product using fully-pipelined hardware. This instruction takes two cycles but the processor will only stall if the result of the multiplication is needed in the next cycle. The compiler will attempt to schedule other instructions, including other multiplies, in between the multiply and any use of its result.

Pipelined Plus UH/SH

In addition to the fully-pipelined MULL instruction, this option contains two fully pipelined instructions, MULSH for signed values and MULUH for unsigned, to compute the high 32-bits of a 32-bit times 32-bit into 64-bit product integral multiplication. These instructions will be automatically inferred by the compiler to aid in the multiplication of two signed or unsigned variables where at most one of the input variables is 64-bits. For other uses, you may access these instructions directly using intrinsics as follows.

```
#include <xtensa/tie/xt_mul.h>
int a, b, c;
unsigned ua, ub, uc;
...
c = XT_MULSH(a, b);
uc = XT_MULUH(ua, ub);
```

2.4.2.2 MUL16 Option

16-bit multiplier (signed/unsigned)

MUL16 option	Selected

MUL16 supports signed and unsigned 16-bit multiplication.

The MUL16 implementation is essentially free because MAC16 is selected. because MUL32 is selected.

Related Links

MUL32 Option on page 27

MAC16 DSP Instruction Family on page 28
16-bit Multiply/Accumulate with 40 bit accumulator (instruction family)

2.4.2.3 MAC16 DSP Instruction Family

16-bit Multiply/Accumulate with 40 bit accumulator (instruction family)

MAC16 DSP option	Selected

The MAC16 instruction family is a series of instructions allowing a 16-bit multiply accumulate into a 40-bit accumulator in parallel with two 16-bit updating loads. It allows a full iteration of a 16-bit dot product every cycle. Note that the instructions in this family that perform loads in parallel with the multiply accumulate are specialized and are not inferred by the C compiler. The only way to use these instructions is with compiler intrinsics or with hand-coded assembly. Note that using intrinsics, the specialized m registers are accessed by passing in their index, 0 to 3, directly into the intrinsic. The compiler is able to infer use of the multiply accumulate instruction that does not execute in parallel with a load. However, this instruction is typically no faster than what is enabled by the MUL16 option.

With no other multiplication options, the compiler will emulate 32-bit multiplications using the MAC16 instructions.

2.4.2.4 CLAMPS Option

Signed CLAMPS instruction (for saturating arithmetic)

CLAMPS option	Selected

The CLAMPS instruction tests whether a signed integral variable fits into a fixed number of bits ranging from 7 to 22. If so, the input value is returned. If not, the largest value with the same sign that does fit into the fixed number of bits is returned. This option is particularly useful for implementing saturating arithmetic. The compiler will automatically infer this instruction from the pattern.

`x = min(max((x,-2`^{n}`),2`^{n}`-1))=`

Min and max are first inferred from standard C conditional operations.

Consider the following example:

```
if (a < -1024) {
    result = -1024;
} else if (a > 1023) {
    result = 1023;
} else {
    result = a;
}
```

When compiled with optimization, the compiler will generate a CLAMPS instruction with an immediate value of 10.

2.4.2.5 32-Bit Integer Divider

32-bit integer divider that completes in a variable number of cycles depending on the operands

32-bit integer divider	Selected

This option adds four instructions that are used to perform 32-bit integer division. The instructions compute the quotient or the remainder, for signed or unsigned numbers respectively. These instructions take from two to 13 cycles depending on the bit patterns of the dividend and the divisor. The divide instructions are implemented using iterative or non-pipelined hardware, which means that instructions subsequent to the divide will not begin execution until the divide operation is complete. The compiler will infer the use of these instructions for all 8-, 16- and 32-bit integer division and modulo computations.

2.4.3 Miscellaneous ISA Instruction Options

These can all be selected from the Instructions page of the Xplorer Configuration Editor. Refer to the Architectural Options chapter in the *Xtensa Instruction Set Architecture (ISA) Reference Manual* for more details.

2.4.3.1 NSA/NSAU Option

Normalize shift amount

NSA/NSAU option	Selected

The NSA (normalize shift amount) instruction returns the number of contiguous sign bits in the most significant bits of a 32-bit signed value. The NSAU instruction returns the number of contiguous zero bits in the most significant bits of a 32-bit unsigned value. These instructions are not inferred by the compiler. However, library routines provided by Xtensa, in particular integer division and modulo, make heavy use of these instructions when present. The instructions can be accessed via intrinsics as follows:

```
#include <xtensa/tie/xt_misc.h>
int a, b;
unsigned ub;
...
a = XT_NSA(b);
a = XT_NSAU(ub);
```

2.4.3.2 MinMax Option

MinMax option	Selected

This performs a signed or unsigned integral minimum or maximum operation. These instructions require very little hardware and will be automatically inferred by the compiler.

2.4.3.3 SEXT (Sign Extend To 32-bits) Option

Sign extend to 32-bits (SEXT)

SEXT	Selected

This replicates one of bits 7 to 22 of an integral value into all high order bits of the 32 bit integer. The compiler must perform sign extension whenever converting values of type int into variables of type short or signed char. Since the semantics of C and C++ frequently promote short or char types into ints, the compiler frequently has to sign extend when you use short or char variables.

2.4.3.4 Density Instructions

Enable use of density (16-bit) instructions

Density option	Selected

This option enables the use of 16-bit instructions. The compiler will automatically use 16-bit variants of the core 24-bit instructions whenever possible to minimize code size. On average, enabling this option will reduce code size by 10% to 20%.

2.4.3.5 Boolean Registers option

Enable use of Boolean Register File (for TIE)

Boolean Registers option	Selected

This option adds a set of 16 single-bit registers and instructions that operate on these registers. The instructions perform boolean operations on the registers and can branch based on the value of one or more of these registers. Note that no core instructions set the boolean registers. They are only set by custom TIE instructions or by Cadence coprocessors such as floating point, ConnX Vectra LX and HiFi 2. Therefore, this option is useless unless you have one of these coprocessors or custom TIE instructions. However, this option is absolutely required if you do have one of these coprocessors or custom TIE instructions. To utilize the booleans registers in your C or C++ program, you must include the xtensa/tie/xt_booleans.h file. This file defines new datatypes xtbool, xtbool2, xtbool4, xtbool8 and xtbool16, corresponding to single bit boolean variables or SIMD style sets of booleans. You may branch on xtbool conditions using standard C control flow constructs. The other operations on booleans are accessible via intrinsics.

2.4.3.6 Processor ID Option

Processor ID option	Selected

Processor ID enables external logic to set a unique identifier for each processor in a system. This option is useful when a single piece of code executing on multiple processors wants to know which processor is invoking the instruction. The value of the id can be accessible in C as follows:

```
#include <xtensa/tie/xt_core.h>

int my_id = XT_RSR_PRID();
```

2.4.3.7 TIE Arbitrary Byte Enables Option

Selecting the TIE arbitrary byte enables option allows you to use the StoreByteDisable interface in your TIE code

TIE arbitrary byte enables option	Selected

This option is necessary for TIE files and Cadence coprocessors that contain instructions that disable part of a memory store. A TIE developer might use this feature for developing predicated, SIMD, memory references. Example usage: the ConnX Vectra LX and ConnX D2 coprocessors use this feature for implementing unaligned data accesses. HiFi 2 uses this feature to do conditional, bit-stream writes.

2.4.3.8 Zero-Overhead Loops Option

Enable zero-overhead loop instructions (eliminates loop pipeline overhead)

Zero overhead loops option	Selected

These instructions enable looping with no per-iteration cycle overhead. The use of the loop instructions set up a loop trip count, beginning PC and ending PC. Whenever the hardware executes the last instruction in the loop and the loop trip count has not been exceeded, control is automatically transferred back to the first instruction without needing any explicit branch instructions. Because every taken branch on Cadence processors requires 3 to 5 cycles to execute, the zero overhead loop instructions are extremely useful for code that spends time in loops. When compiling with optimization, the compiler will typically use these instructions for every loop that does not contain a function call.

2.4.3.9 Synchronize Instruction

Synchronize instruction	Selected

This option adds load-acquire and store-release instructions (L32AI and S32RI) to ease multiprocessor synchronization.

Related Links

Conditional Store Sync option on page 32

2.4.3.10 Conditional Store Sync option

Conditional Store Sync option	Selected

This option adds the S32C1I instruction to ease multiprocessor synchronization by providing an atomic compare and store operation.

. This processor includes the conditional store synchronization instruction for multiprocessor synchronization, so you must implement PIF support for the RCW transaction. Further note, that the AXI and AHB bridges supplied by Cadence come with built-in support for this transaction that is implemented by locking the bus. Refer to the *Xtensa Microprocessor Data Book* for details.

Related Links

Synchronize Instruction on page 32

2.4.3.11 Number of Coprocessors option

Maximum number of coprocessors that can be defined in TIE to support efficient (lazy) register save/restore

Number of Coprocessors option	2

Can be set from 0 to 8. Designating a coprocessor in TIE creates a bit in a special coprocessor enable register. When that bit is set, any access to the coprocessor's registers causes an exception. The associated exception-handling routine can then save the state of the coprocessor's registers before they are used, for example, in a new context.

2.4.3.12 Misc Special registers option

Misc Special registers option	2

The miscellaneous special registers option provides zero to four scratch registers within the processor readable and writable by RSR, WSR, and XSR. These registers are privileged. They may be useful for some application-specific exception and interrupt processing tasks in the kernel. The MISC registers are undefined after Reset.

2.4.4 ISA Configuration Options

These can all be selected from the Instructions page of the Xplorer Configuration Editor. Refer to the Architectural Options chapter in the *Xtensa Instruction Set Architecture (ISA) Reference Manual* for more details.

2.4.4.1 AR Registers Count

Number of physical AR registers. Setting to 16 registers means windowed calls are not supported

AR Registers Count	64

The Cadence windowed ABI allows you to have more physical AR registers than the 16 that are directly accessible by instructions in the ISA. On a call, the system rotates the AR register file, automatically giving the called function access to new scratch registers. This windowing mechanism allows for faster and smaller code. Whenever the number of physical registers is exceeded, an exception is thrown and the exception handler automatically saves and restores the excess registers to and from the memory stack. See the Xtensa Instruction Set Architecture (ISA) Reference Manual for more details.

Cadence allows 16, 32, or 64 physical registers. The selection of 16 registers necessitates the use of the CALL0 ABI. The use of 32 versus 64 physical registers does not affect application software. The choice is a trade-off between application performance and hardware area. The extra registers add approximately 5,000 gates to the processor but minimize the number of exceptions. If the number of gates is significant to your application, it is recommended that you profile your application and search for the functions beginning with _WindowUnderflow and _WindowOverflow. The more time spent in these handlers, the greater the value of having 64 physical registers.

Related Links

Application Binary Interfaces on page 23

2.4.4.2 Byte Ordering Option

Little or Big Endian

Byte ordering option	Little Endian

Cadence offers the choice of little or big endian as a configuration option. Every built processor configuration supports only one endianness. Cadence hardware and software supports both endianness equally. Mixing endianness in the same multiprocessor system is difficult.

2.4.4.3 Unaligned Load / Store Action Selection

How unaligned loads / stores are handled.

Unaligned Load / Store action selection	Handled by hardware

Traditional RISC processors expect that variables are aligned to their natural boundaries. For example, a 32-bit int variable is expected to be aligned to 32-bits. The C and C++ compilers will always align variables appropriately. However, through the use of casts, parameters or pointers might point to unaligned data. The compiler will assume that all data is properly aligned unless it's obvious to the compiler that it is not. For example, ((int *) 0x1), is obviously unaligned.

Cadence offers three configuration options to deal with circumstances where unaligned accesses occur. With Align address, the hardware will zero the bottom bits of the address before performing the memory access. This is rarely the desired behavior and is mainly provided for compatibility with earlier Cadence processors that only offered this option. This option is also unsupported on configurations with a data memory interface of greater than 128-bits. With Take exception, the hardware will throw an exception whenever an unaligned access is attempted. Typically, unaligned accesses should be treated as programming errors and the exception is an aid to debugging. However, for those running Linux on Xtensa processors, the exception handler in Linux will emulate an unaligned hardware access using multiple-aligned accesses. Using the exception handler is slow, but is useful when running open source drivers that are not performance critical and assume support for unaligned accesses. Note that writing handlers that emulate unaligned accesses is not easy or supported for configurations with FLIX. The third option, Handled by hardware, has the hardware automatically support unaligned accesses. Note that the hardware for handling these accesses takes several cycles so that performance-critical code should still only issue aligned accesses. The advantage of having the hardware handle the unaligned accesses is that it is faster than the emulation routine available in the exception handler and is also able to work together with FLIX. The advantage of the exception is that it makes it easier to identify and fix unaligned accesses, leading to more efficient and reliable code. Diamond processors provided by Cadence use the Take exception option.

2.4.4.4 Max Instruction Width Option

Max instruction width. Xtensa core instructions are 2 or 3 bytes wide.

Max instruction width option (bytes)	8

Cadence supports modeless intermixing of multiple instruction sizes. All configurations support 24-bit instructions. The use of 16-bit instructions is almost always recommended to save code size. The 16-bit instructions are equivalent variants of the most commonly used 24-bit instructions so that the compiler will always use them when possible. Additionally, Cadence supports customer defined FLIX (VLIW) instructions of any multiple of eight length from 32 to 128 bits. You can partition the wide instructions into custom slots, each of which is capable of executing one of a set of operations. The compiler automatically packs operations into the instructions. To utilize larger instructions, set the maximum instruction width appropriately.

2.4.4.5 L32R Hardware Support Option

Select what hardware support options to include for L32R.

L32R hardware support option	Normal L32R

The normal L32R instruction does a PC-relative load using a 16-bit offset to load literals. Literals are used to hold addresses of global variables and functions in addition to user specified literal constants. L32R is broadly supported by all Xtensa software and tools and is usually the best selection. However, on systems with instruction caches but no data caches, an L32R instruction might only be able to reach literals placed in system memory. Loading a literal from system memory on a core without a data cache will be very slow. Choosing L32R + Const16 adds the ability to load literals using 2 operations with 16-bit immediates, eliminating the need for any loads. When Const16 is enabled, the compiler (assembler) will generate that by default. Const16 requires one extra instruction to generate a literal, typically requires more memory and is not supported by all Xtensa software; Linux, for example, is not supported. However, const16 is much faster than an L32R that needs to load a literal from system memory without a data cache.

Hardware support for Extended L32R is a legacy option which is supported through LX3/X8 processors only. Note that whether software makes use of normal L32R, extended L32R for loading literals is a separate software build option.

Related Links

Use Extended L32R Instruction (for Legacy Hardware)

2.4.4.6 Pipeline Options

The default pipeline is 5 stage; LX processors can optionally be configured to have a 7 stage pipeline.

Pipeline length	5
Instruction Memory Fetch Latency	1
Cycle of Execute stage	1
Cycle of Modify stage	2
Cycle of Write-back stage	3

The base Cadence processor is a 5-stage pipeline micro-architecture with a single stage dedicated to data memory and another stage dedicated to instruction fetch. For large local memories (caches or on chip local memories) on configurations being clocked aggressively, the speed of the memory can limit the speed of the processor core. For such configurations, Cadence offers the option of adding two extra stages to the pipeline, one to the instruction fetch and another to the data memory. The use of these extra stages allow for larger and slower local memories without impacting the processor clock rate. These extra stages increase the branch penalty by one cycle and the load-use delay by one cycle. Therefore, they potentially slow down an application's speed, as measured in number of clocks required to complete the application. Some, applications, mostly DSP, will not slow down appreciably. However, control type code can slow down 15% or more.

2.5 Interface Options

Options which affect interfaces.

2.5.1 Bus and Bridge Selections

These can all be selected from the Interfaces page of the Xplorer Configuration Editor.

2.5.1.1 PIF / Bus Selection

PIF interface to external memories with optional bus bridge

PIF / Bus Selection	AXI3

The PIF is the processor's main interface to memory, and is required for an Xtensa processor that is configured with caches and/or system memories.

In addition to the PIF, an external bus bridge can be selected - either AHB-Lite, AXI3 or AXI4.

2.5.1.2 AXI Bridge Options

2.5.1.2.1 AXI Slave Request Control Depth option

Bridge Slave Request Control Depth option	4

AXI slave Request Control Buffer Depth can be set to one of 1,2,4,8,16 entries. The default is 4 entries. Each entry consists of the AXI request control signals used by the bridge viz address, length, size, burst type and ID bits. This selection applies to both Read Address and Write Address channels.

2.5.1.2.2 AXI Slave Request Data Depth option

Bridge Slave Request Data Depth option	8

AXI slave Request Data Buffer Depth can be set to one of 1,2,4,8,16 entries. The default is 8 entries. Each entry consists of AXI write request data signals used by the bridge viz data, strobes, and ID bits. This selection only applies to Write Data channels.

2.5.1.2.3 AXI Slave Response Depth option

Bridge Slave Response Depth option	8

AXI slave Response Buffer Depth can be set to one of 1,2,4,8, or 16 entries. The default is 8 entries. This parameter changes the number of entries in the read response buffer. Each entry in the read response buffer stores read data bits along with error control and ID bits. Note that the selection of this parameter only applies to Read Response channel. The depth of AXI slave write response buffer is not configurable and is set to 16 entries.

2.5.2 PIF Options

These can all be selected from the Interfaces page of the Xplorer Configuration Editor.

2.5.2.1 Count of PIF Write Buffer Entries

Count of PIF write buffer entries	16

Write buffer is used to mask the latency of large bursts of writes, data-cache dirty-line castouts, and register spills.

2.5.2.2 Inbound PIF Request Buffer Depth

Inbound PIF request buffer depth	4

Inbound PIF allows an external PIF master to read/write to Xtensa's internal memories. When the inbound-PIF request option is enabled, an inbound-PIF request buffer is added to the processor. The depth of the inbound-PIF request buffer limits the maximum block-read or -write request that the Xtensa processor can handle. Therefore, the inbound-PIF request buffer size should be configured to be greater than, or equal to the largest possible inbound-PIF block-request size. Note that this control is enabled when the inbound-PIF option is selected for one or more local memories.

2.5.2.3 PIF Write Responses option

Enable PIF Write Responses. If enabled, a "Write Error" interrupt may be configured

PIF write responses option	Selected

A PIF transaction is composed of a request and a response. For a PIF write, the Xtensa processor can be configured to not count the write response. In that case, any write transaction that has been requested by the Xtensa processor, a PIF master, is assumed to be completed in the future. Any write response from a slave is accepted but ignored. Write responses may be important for memory ordering or synchronization purposes. However, on long latency systems, enabling write responses might significantly impact application performance.

When the write-response configuration option is selected, 16 unique IDs are allocated to write requests, and no more than 16 write requests can be outstanding at any time. In addition, store-release, memory-wait, and exception-wait instructions will wait for all pending write responses to return before those instructions commit.

Note: If the inbound-PIF configuration option is selected, the write responses option causes the PIF to send write responses to external devices that issue inbound-PIF write requests.

2.5.2.4 PIF Critical Word First option

Enable loading of critical word first on a block request

PIF Critical Word First option	Selected

The Xtensa processor can be configured to issue block-read transactions as Critical Word First Transactions. Any block-read can be issued such that a specified PIF width of the block will arrive first, with the remaining PIF widths arriving in sequential order and wrapping around to the beginning of the block e.g. a block-read of 8 PIF widths could arrive as 5, 6, 7, 0, 1, 2, 3, 4.

This requires that the Early Restart option be selected. Selecting both Critical Word First and Early Restart will improve the processor miss penalty by approximately 1 + (Cache_line_size/PIF_size) cycles. If only Early Restart is selected, you will get the same benefit if a miss is to the first element of a cache line but as little as one cycle of benefit if the miss is to the last element of the cache line.

Related Links

Early Restart option on page 39
Enable Early Restart as soon as the critical word has been filled to a cache-line

2.5.2.5 PIF Arbitrary Byte Enable option

PIF Arbitrary Byte Enable option	Selected

The Xtensa processor can be configured to use Arbitrary Byte Enables on write requests. This may allow more efficient execution when Arbitrary Byte Enable TIE is used that generates stores that have arbitrary byte enable patterns.

Note: LX5 and X10 processors will only generate Arbitrary Byte Enables on write single transactions.

2.5.2.6 Early Restart option

Enable Early Restart as soon as the critical word has been filled to a cache-line

Early Restart option	Selected

This option enables early restart on both instruction cache line misses and data cache line misses. For data, the instruction will wait in the M-stage of the pipeline and the data for the cache line miss data will be forwarded to it there. For instruction fetch, the fetch will stall in the I-stage and the cache line miss data will be forward to it there.Refill of the cache lines occurs in the background while the pipeline proceeds. This improves cache miss performance, at the cost of area.

Selecting both Critical Word First and Early Restart will improve the processor miss penalty by approximately 1 + (Cache_line_size/PIF_size) cycles. If only Early Restart is selected, you will get the same benefit if a miss is to the first element of a cache line but as little as one cycle of benefit if the miss is to the last element of the cache line.

Related Links

PIF Critical Word First option on page 38
Enable loading of critical word first on a block request

2.5.3 Prefetch Options

These can all be selected from the Interfaces page of the Xplorer Configuration Editor.

Selecting a non-zero count of cache prefetch entries option selects whether prefetch is included; the other options modify its behaviour.

Xtensa processors can prefetch to instruction and data caches if the relevant cache configuration options are compatible. If not, then prefetch will only be supported to the data cache. The requirements for prefetching to both instruction and data caches are as follows:

- Xtensa LX5 and X10 processors and earlier: The instruction and data cache line widths must be the same, plus the instuction cache and data cache access widths must be the same
- Xtensa LX6, LX7 and X11 processors and later: The instruction and data cache line widths must still be the same, but the access width restriction is relaxed as described in the appropriate Data Book. Note that the conditions have changed, so make sure to check with the Data Book corresponding to the specific release you will use to generate hardware.

2.5.3.1 Cache Prefetch Entries

Controls how many cache lines can be held in the prefetch buffer

Cache Prefetch Entries	8

Select > 0 entries to include the cache prefetch unit.

2.5.4 Interface Width Options

These can all be selected from the Interfaces page of the Xplorer Configuration Editor. The following simplified diagram shows the memory interface widths:

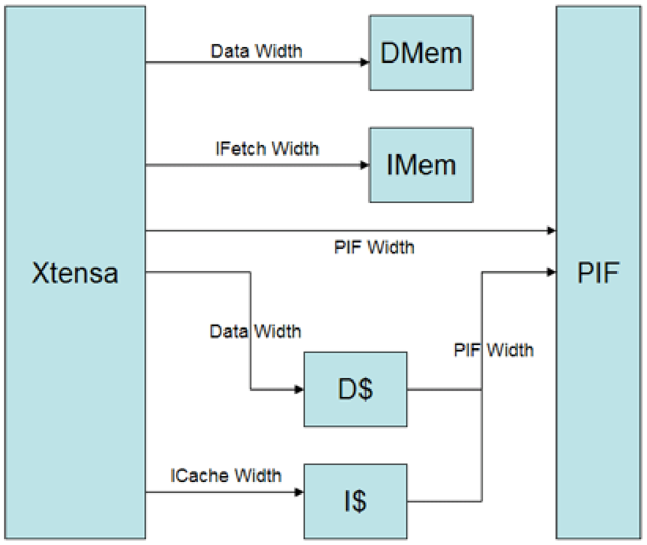

Figure 1: Memory Interface Widths

Cadence allows you to separately control the instruction fetch width, the data cache/memory width, the instruction cache/memory width and the width of the PIF inter- face. In general, wider widths give higher performance at a higher cost in area. The instruction fetch width controls how many bits are fetched in a cycle from the Icache or local memory into holding buffers. This parameter can be set to 32, 64 or 128. Configurations with 64-, 96- or 128-bit FLIX instructions are required to set this parameter to 64- or 128-bits respectively. For configurations without wide instructions, the use of a 64 or 128-bit fetch may still have two potential advantages. First, a taken branch on Cadence processors suffers a minimum two cycle penalty (three on configurations with a 7-stage pipe) assuming that the entire target instruction can be fetched with a single load from the local instruction memory. The fetch unit always fetches at least as much data as the size of the largest supported instruction. However, the fetch unit always fetches aligned data. If the target instruction crosses a 32-bit boundary assuming a 32-bit fetch width or a 64- or 128-bit boundary assum-ing a 64- or 128-bit fetch width, then there is one additional cycle of penalty. Fewer instructions cross 64- or 128-bit boundaries than 32-bit boundaries. Therefore a processor with a wider fetch will suffer fewer branch bubbles. Second, for straight line code, the use of a 64- or 128-bit fetch implies that the fetch unit needs to fetch fewer times. Fetching a single 64- or 128-bit chunk of instructions consumes less power than fetching multiple, 32-bit chunks. Of course, fetching

64- or 128-bits consumes more power than fetching 32-bits, and if the application branches sufficiently frequently, the use of a 64- or 128-bit fetch will not cut the number of fetches in half. Therefore, which configuration option consumes less power depends on how frequently branches are taken.

The data cache or memory width controls how many bits are transferred from external memory into the cache per cycle as well as how many bits can be loaded or stored from the cache or local data memory every cycle. The width must be at least as large as the largest load or store instruction and must be at least as large as the PIF. The PIF width is the width of the memory interface from external memory to the local memories or caches. It is also the data transfer width for non-local, uncached memory references. Larger PIF widths enable faster handling of cache misses. It is typically better to make your PIF width match your system bus width rather than externally bridge the processor to a smaller system bus width.

The Xtensa processor supports an optional hardware and software prefetch mechanism for systems with large memory latency. When the processor detects a stream of cache misses (either data or instruction), it can speculatively prefetch ahead up to four cache lines and place them in a buffer close to the processor. In addition, the user can explicitly control prefetching using the DPFR instruction. Prefetch is enabled by setting the number of Cache Prefetch Entries.

The interactions between the widths and other parameters are fairly complex, and are enforced by the Xplorer Configuration Editor. The basic rules are as follows:

- ICache width >= IFetch width
- ICache width <= max(IFetch, PIF)
- IFetch width >= Max instruction width
- Data width >= PIF width

2.5.4.1 Width of Instruction Fetch Interface

Width of Instruction Fetch interface	64

The instruction fetch width must be at least as large as the maximum instruction size.

Related Links

Interface Width Options on page 40

2.5.4.2 Width of Data Memory/Cache Interface

Width of interface to Data RAM, ROM, XLMI, Data Cache

Width of Data Memory/Cache interface	64

The maximum width of data for a load or store instruction. Core instructions are 32 bits or less; TIE instructions can access up to this width in a single operation. Most Cadence DSP coprocessors make heavy use of wide loads and stores.

2.5.4.3 Width of Instruction Cache Interface

Width of Instruction Cache interface	64

Normally the instruction cache width should be the same as the instruction fetch width, either 32-, 64-, or 128-bits. The instruction fetch width is the amount of data fetched from the cache on each instruction fetch read access, and there is rarely any benefit to having the instruction cache width wider than this, since wider memories usually consume more power. One exception is when instruction cache refill time is critical, since a wider instruction cache reduces this refill time, assuming a wider PIF interface is also used.

Must be at least the width of Instruction fetch Width and cannot exceed max(PIF, InstFetch).

2.5.4.4 Width of PIF Interface

Width of PIF interface	64

The PIF can be configured to be 32, 64, or 128 bits wide. The PIF read and write data buses are the same width.

2.5.5 Port / Queue Options

2.5.5.1 GPIO32 Option

32-bit General purpose Input output TIE port interface. Contains a 32-bit output port (EXPSTATE) and 32-bit input port (IMPWIRE)

GPIO32 option	Selected

The GPIO32 option provides a pair of preconfigured output and input ports that allow SOC designers to integrate the processor core more tightly with peripherals. For example, these GPIO ports can be used to receive and send out control signals from/to other devices and RTL blocks.

2.5.6 Caches and Local Memories

Subsections describe the individual cache and local memory options in more detail.

Cadence allows up to six local memory interfaces on each of the instructions and data sides. Each interface might be a local RAM, local ROM or cache. Each way of a set-associative cache counts as one interface. The caches can be anywhere from 1 Kilobyte to 128 Kilobytes, from direct-mapped to 4-way set associative, with line sizes from 16 bytes to 256 bytes.

Caches allow reasonably robust performance with minimal effort. Local memories potentially allow higher performance and efficiency, but not always. Local memories support external DMA engines through the processor’s inbound PIF port. DMA allows you to work on one block of data while loading another block in the background. DMA potentially completely avoids the penalties of a cache miss. Of course, this only works if the working set sizes of the current block plus the block being loaded in parallel together are small enough to fit inside the local memory.

Caches work well when the total memory being used is significantly larger than the local memory size but the working set at any given time is sufficiently small. Local memories are much harder to use in such scenarios. Data must be explicitly and manually moved into and out of the local memory. Partitioning code is not always easy. You may try to use both local memories and caches, putting your frequently used data or code in local memories while leaving caches to handle the rest. This can be very effective if some code or data is small and used frequently, and other code or data is very large and is being streamed into the processor. Frequently, however, making such a clean partition is difficult; hardware does a better job of dynamically allocating memory to caches than you can statically. Local memories require less power to access than equivalently sized caches. Direct-mapped caches require significantly less power than set associative caches. Direct-mapped caches can perform well, but performance can be less robust. Small changes to an application can have a dramatic performance impact if two pieces of code or data suddenly fall into the same cache location. With direct-mapped caches, be certain to utilize some of the performance tuning and measuring methodologies described in Chapter 2 to make sure that you are not thrashing the cache. In particular, the Cache Explorer allows you to automatically simulate performance and power usage for various cache systems on your actual application, and the Link Order tool allows you to rearrange your code to minimize instruction cache misses.

Two local memories of size n/2 require less power than one local memory of size n. Two local memories can also increase the performance of DMA because the DMA engine writing into one memory will not compete for bandwidth with the processor trying to access the other memory. However, with two local memories, you must partition the data or code between the two local memories. Cadence also supports line locking of all but one way in a set associative cache. Line locking provides some of the benefits of local memories in a cache. In order to effectively utilize line locking, you must explicitly identify data or code that is small and frequently used. As with local memories, it is often hard to statically partition as well as the hard- ware caching mechanism is able to automatically partition.

Caches and local data memories can be divided into one to four banks. The data memory is divided into banks so that successive data memory width sized accesses go to different banks. At most one load or store can go to any one bank in a cycle. On configurations that support multiple loads or stores per cycle or on systems with DMA, using more banks will minimize the number of stalls due to bank conflicts.

See the appropriate Data Book for more detailed information.

2.5.6.1 Instruction Cache Details

Instruction Cache size bytes	65536
Instruction Cache ways	4
Instruction Cache line size bytes	128

Instruction Cache line locking	Selected
Instruction Cache memory errors	Not Selected
Dynamic Way Disable	Not Selected

- **Associativity**: - 1 through 4 way associativity is supported.
- **Size**: Total cache size of all configured ways. Minimum size of a way is 512 bytes; minimum cache size is 1KB.
- **Line Size**: Size of a cache line in bytes. The *critical word first* option can be used to reduce the penalty when a line is being filled from external memory.
- **Line Locking**: keeps a line in the cache until it is unlocked. Once locked, the line behaves similar to local memory. This is useful when working with a small piece of code without incurring an expense of having a local memory. Refer to the Local Memory Usage and Options chapter in the appropriate *Xtensa Microprocessor Data Book* for more information.

The dynamic cache way disable capability gives the ability to disable and re-enable the use of cache ways in both Instruction- Cache and Data-Cache independently to facilitate power savings. New and modified instructions enable the user to clean cache ways before disabling them and to initialize cache ways while enabling them. When a Cache Way is disabled, it removes that cache memory block from service. Therefore it reduces total cache capacity by 1/(number of ways in service).

Restriction: LX6/X11++ only.

Related Links

PIF Critical Word First option on page 38
Enable loading of critical word first on a block request

Early Restart option on page 39
Enable Early Restart as soon as the critical word has been filled to a cache-line

2.5.6.2 Data Cache Details

Data Cache size bytes	65536
Data Cache ways	4
Data Cache line size bytes	128
Data Cache line locking	Selected
Data Cache memory errors	Not Selected
Data Cache write-back	Selected
Number of Data Cache Banks	1

Dynamic Way Disable	Not Selected

- **Associativity**: - 1 through 4 way associativity is supported.
- **Size**: Total cache size of all configured ways. Minimum size of a way is 512 bytes; minimum cache size is 1KB.
- **Line Size**: Size of a cache line in bytes. The critical word first option can be used to reduce the penalty when a line is being filled from external memory.
- **Write-back**: The data cache is a write-through cache by default. If this option is selected, the data cache can be programmatically toggled between write-back and write-through.
- **Line Locking**: The data cache allows line locking, which keeps a line in the cache until it is unlocked. Once locked, the line behaves similar to local memory. This is useful when working with a small piece of code/data without incurring an expense of having a local memory. Refer to the Local Memory Usage and Options chapter in the appropriate *Xtensa Microprocessor Data Book* for more information.
- **Banks**: If multiple banks are configured, then the data cache is divided into banks so that successive data memory width sized accesses go to different banks. At most one load or store can go to any one bank in a cycle.

  Restriction: Multiple banks are LX5++ only

The dynamic cache way disable capability gives the ability to disable and re-enable the use of cache ways in both Instruction- Cache and Data-Cache independently to facilitate power savings. New and modified instructions enable the user to clean cache ways before disabling them and to initialize cache ways while enabling them. When a Cache Way is disabled, it removes that cache memory block from service. Therefore it reduces total cache capacity by 1/(number of ways in service).

Restriction: LX6/X11++ only.

Related Links

PIF Critical Word First option on page 38
Enable loading of critical word first on a block request

Early Restart option on page 39
Enable Early Restart as soon as the critical word has been filled to a cache-line

2.5.6.3 Local Memories

Local Memory Details

Table 29: Local Memories

Memory	Size	Address	Inbound PIF	Busy
Instruction RAM 0	32K	0xe8080000	Selected	Not Selected
Instruction RAM 1	16K	0xe8088000	Selected	Not Selected
Data RAM 0	32K	0xe8058000	Selected	Not Selected
Data RAM 1	128K	0xe8060000	Selected	Not Selected

Local Memory Options

Note that the "normal" L32R instruction which is used to load literals has a range of 256K bytes preceding the current PC, so the default positioning of instruction / data puts data memories before instruction memories so the data memory can be used for literal storage. If there is no data memory within range, the editor warns because the compiler may have problems generating literals if compiling code into that memory.

Attributes that can be selected for local memory interfaces (inbound PIF, busy and memory error) must be consistent for each memory type. E.g. if you configure 2 data RAMs then either both must have inbound PIF configured, or neither.

Selecting **Inbound PIF** allows an external PIF master to read/write to Xtensa's internal memories. When the inbound-PIF request option is enabled, an inbound-PIF request buffer is added to the processor.

Selecting **Busy** will create an external interface to the processor that the processor will check before writing to the memory.

Banks: Data RAM and Data ROM can optionally be configured in 2 or 4 banks. If multiple banks are configured, then the data RAM is divided into banks so that successive data memory width sized accesses go to different banks. At most one load or store can go to any one bank in a cycle.

Split Read-Write port: For multiple load-store configurations, this brings out the interfaces for all load-store units so you can handle the multiplexing and banking ouside of the core. The alternative is to select CBOX which handles the multiplexing inside the core.

See the appropriate Data Book for more detailed information.

Related Links

Automatically Select Memory Addresses on page 47
Xplorer can automatically position local memories for alignment and proximity needs

2.5.6.4 Automatically Select Memory Addresses

Xplorer can automatically position local memories for alignment and proximity needs

Automatically Select Local Memory Addresses	Not Selected

By default the option **Automatically select memory addresses** is enabled. In this mode, the Configuration Editor chooses appropriate local memory addresses to keep them naturally aligned and close together such that literals can be loaded from data memory with L32R.

If you want to ensure exact compatibility with addresses of some other processor configuration it is appropriate to uncheck the auto-placement option so you can enter specific addresses. Consider also the memory regions in which system and local memories are placed such that your cache attributes and power considerations are met.

Note also that exception vectors need to be placed inside valid memories. If vectors are marked as being placed in an instruction memory, Xplorer's default behaviour is to automatically match the vector location with that of the containing memory.

Local Memory root address: If the full MMU is selected, then the configuration editor requires that local memories be "auto-placed" to ensure that there are no address space collisions. This option lets you choose whether that local memory auto-placement occurs in kernel space or user space.

Related Links

Automatically Position Vectors on page 54

Memory Management Selection on page 27
Region protection options provide coarse protection at a low gate count The Full MMU provides resource management capabilities sufficient for running Linux

2.5.6.5 Load/Store Units

LoadStore units	1

Cadence supports the use of one or two load/store units. Dual load/store units potentially allow your application to issue two load/stores every cycle. The processor will only take advantage of the second load/store unit if you have custom FLIX TIE instructions with loads or stores in multiple slots, or if you utilize the dual load/store variant of a ConnX processor, which come pre-built with two load/store units.

Data caches on dual load/store configurations must be at least 2-way set associative and must have at least two banks. The memory is banked so that successive data memory access width size references go to different banks. The processor can not issue multiple memory references to the same bank in the same cycle. The compiler will try to compile code to avoid bank conflicts.

Local data memory can be optionally banked. In addition, the user can select the connection box option. With this option, if two memory references go to different local data memories, the processor will not stall. If two memory references go to the same bank of the same local data memory, the connection box will stall the processor for one cycle. With no banking and no connection box, dual load/store configurations require the use of dual-ported memory.

Restriction: Xtensa X and TX processors support just one load / store unit.

2.6 Debug and Trace Options

Debug and Trace options.

2.6.1 Debug option

Debug option	Selected

The Debug option implements instruction-counting and breakpoint exceptions for debugging by software or external hardware. The option uses a high-priority interrupt level.

The sub-options described here require that *Debug* is selected. The Debug features are enabled either through a JTAG or APB slave port on the Xtensa processor. In each case, there is a connection to the Access Port, which provides access to the different debug functions. The Access Port implements a TAP for JTAG access, and an asynchronous peripheral slave port for APB.

2.6.1.1 Count of HW Instruction Traps

Count of HW Instruction Traps	2

These registers are additional hardware to support instruction breakpoints in ROM. When debugging with the ISS these are not needed, but they are required with real hardware.

2.6.1.2 Count of HW Data Traps

Count of HW Data Traps	2

These registers are additional hardware to support data watchpoint capability. When debugging with the ISS these are not needed, but they are required with real hardware.

2.6.1.3 On-Chip Debug option

On-chip debug module with JTAG compatible interface

On-Chip Debug option	Selected

Cadence recommends that OCD is selected; it is required in most debugging scenarions on hardware.

OCD support provides access to and control of the software-visible state of the processor through an IEEE 1149.1 Test Access Port (TAP), also known as JTAG (from the Joint Test Action Group that originated the standard). Through this TAP, an external debug agent can:

- Generate an interrupt to put the processor in the debug mode.
- Gain control of the processor upon any debug exception.
- Read and write any software visible register and/or memory location.
- Resume normal mode of operation.
- Communicate with a running system via the DDR register.

The OCD support feature requires an external TAP controller, and Cadence provides an example TAP controller that implements OCD support. See the *Xtensa Debug Guide* for more information.

Note: Starting with LX5/X10, the OCD option includes two new instructions LDDR32.P and SDDR32.P to speed up memory download/upload through the Debug Module. This is a replacement for the Debug Instruction Register Array option of LX4/X9 and earlier processors.

2.6.1.4 APB Debug Access option

Configure APB access to the Access Port module of the processor

APB Debug Access option	Selected

The Enable APB Debug Access option adds an APB slave interface as specified by the AMBA 3 Advanced Peripheral Bus (APB) protocol. In addition to JTAG, the APB slave interface can be used to access debug functionality, for instance to read and write OCD registers, TRAX control, and Performance Counter registers. The APB slave interface operates on its own clock (PBCLK) which is asynchronous with respect to the CLK signal of the processor core and OCD.

With this option, Xtensa becomes a CoreSight-compatible component as viewed/accessed through the APB. The CoreSight registers and functionality are as described in the *Xtensa Debug Guide*

2.6.1.5 Break-in Break-out option

Add external debug interrupt at debug level (break-in/break-out)

Break-in Break-out option	Selected

This capability enables one Xtensa processor to selectively communicate a break to other Xtensa processors in a multiple-processor system. Refer to the appropriate Xtensa Microprocessor Data Book for more detail.

2.6.1.6 Performance Counters Option

Enable hardware based performance monitoring

Performance Counters option	4

Configures how many counters are available for hardware based performance monitoring - to count events such as cache misses. Each configured counter comprises a pair of registers: control and status.

The Xplorer Profile launch can make use of either performance counters or a hardware timer for hardware based performance monitoring. Timer based monitoring can only count cycles, but is available with earlier processor versions. Performance counters provide for more sophisticated monitoring, and options within the launch dialog allow assignment of different types of events to the available counters. Note that programs have to be linked with a specific option (-hwpg) to make use of hardware performance monitoring (see Build Properties -> Linker Options).

Remember: This option requires that a profiling interrupt is configured.

2.6.2 Trace option

Trace port (address trace and pipeline status)

Trace option	Selected

Xtensa processors support a traceport which can be configured for instruction trace, and optionally data trace. A trace-buffer module (TRAX) can also be configured and generated with the processor configuration. The TRAX module implementation requires that instruction trace, OCD and break-in / break-out are selected.

2.6.2.1 TRAX Memory Size

Size of TRAX memory (bytes), or 0 for none

Size of trace memory (bytes)	4096

Selecting a non-zero memory size generates the TRAX module with the processor RTL.

2.7 Interrupt Options

Interrupt, timer and exception options.

2.7.1 Interrupt Configuration

Count of interrupts	32
Count of interrupt levels	5
Count of timers	2
Debug level	5
EXCM level	3

Table 30: Interrupts Details

Interrupt	Type	Level	BInterrupt Pin
0	nmi	nmi	0
1	sw	3	
2	level	3	1
3	level	3	2
4	level	3	3
5	timer.0	3	
6	timer.1	4	
7	level	3	4
8	level	2	5
9	level	2	6
10	level	2	7
11	level	2	8
12	level	1	9
13	level	1	10
14	level	1	11
15	level	1	12
16	level	1	13
17	level	1	14
18	level	1	15
19	profiling	3	
20	level	1	16
21	level	1	17
22	level	1	18
23	level	1	19
24	level	1	20
25	level	1	21
26	level	1	22

Interrupt	Type	Level	BInterrupt Pin
27	level	1	23
28	level	1	24
29	writeerr	3	
30	level	1	25
31	level	1	26

Interrupt Information

The following sections contain basic information for using the processor interrupts. Setting interrupts requires detailed understanding of the SOC design and the related devices. Refer to the *Xtensa Microprocessor Programmer's Guide* and the appropriate *Xtensa Microprocessor Data Book* for more information on the behavior and support of different levels of interrupts.

Interrupt Types

Interrupt types can be any of the values listed in the table below. The column labeled "Priority" shows the possible range of priorities for the interrupt type. The column labeled "Pin" indicates whether there is an Xtensa core pin associated with the interrupt, while the column labeled "Bit" indicates whether or not there is a bit in the INTERRUPT and INTENABLE Special Registers corresponding to the interrupt. The last two columns indicate how the interrupt may be set and how it may be cleared.

Table 31: Interrupt Types

Type	Priority	Pin?	Bit?	How Interrupt is Set	How Interrupt is Cleared
Level	1 to N	Yes	Yes	Signal level from device	At device
Edge	1 to N	Yes	Yes	Signal rising edge	WSR.INTCLEAR '1'
NMI	N+1	Yes	No	Signal rising edge	Automatically cleared by HW
Software	1 to N	No	Yes	WSR.INTSET '1'	WSR.INTCLEAR '1'
Timer	1 to N	No	Yes	CCOUNT=CCOMPAREn	WSR.CCOMPAREn
Debug	2 to N	No	No	Debug hardware	Automatically cleared by HW
WriteErr	1 to N	No	Yes	Bus error on write	WSR.INTCLEAR '1'
Profile	1 to N	No	Yes	Profiling interrupt	Clear in profiling logic

Interrupt Levels

Low-Level Interrupts

Level 1 interrupts are intended for non real-time interrupts. These interrupts are slower at interrupt handling due to sharing of general dispatch handlers. Level 1 interrupts will go to either the UserExceptionVector or the KernelExceptionVector. The EXCCAUSE register will identify the exception as a level-one interrupt, and software handlers can respond accordingly.

Mid-Level Interrupts

Mid-level interrupts are faster than Level 1 interrupts because they have a dedicated handler (Level<N>InterruptVector). Also, because they are C-callable, they are easy to program. Note: To program a mid-level interrupt you need to save/restore the state when entering/exiting the handler. Mid-level interrupts go to a dedicated vector. For example:

```
Level2InterruptVector
Level3InterruptVector
...
```

High-Level Interrupts

High-level interrupts, which are written in assembly, are the fastest interrupts (with the lowest latency) because they have a dedicated handler. Supporting Interrupts Service Routines (ISRs) in assembly only requires that the handler saves/restores the registers that it uses and issues RFE/RFI when done. Also, because latency of high-level interrupt is very important, designers should understand the latency of the execution of the handler (including memory latency).

Sometimes, it is possible to have a high-level interrupt trigger a lower level interrupt in which the handler is written in C.

2.8 Vector and System Memory Options

Vector memory placement and system memory options.

2.8.1 System Memories

System RAM and ROM

Table 32: System Memories

Memory	Base Address	Size
System RAM	0xc0000000	256M
System ROM	0xd0000000	16M

System memory covers the entire 32-bit address space that is not mapped to any configured local memory port. Generally, system memories are not part of the processor configuration, but:

- The XPG build process makes use of system memory for diagnostics if PIF is configured requiring that memories are configured.
- Vectors are, by default, automatically placed into memories and will default into system memory in most cases.

Beyond the limitations noted above, system memories can be added, removed, resized and repositioned as desired in your final system without rebuilding the processor on the XPG. See the *Xtensa Linker Support Packages (LSPs) Reference Manual* for information on changing and regenerating linker scripts to match the memory you intend to use with software support.

2.8.2 Automatically Position Vectors

Automatically Position Vectors	Not Selected

Vectors have to be in valid memories. This option helps keep the vector addresses current as memories are moved and sized during the configuration process. Vectors are automatically positioned according to the selected *Vector Layout Style*.

Related Links

Vector Layout Style on page 54

2.8.3 Vector Layout Style

Vector Layout Style	Xtensa Relocatable

This selection controls how Xplorer automatically places vector addresses when the option *Automatically Position Vectors* is selected. It also controls style compatibility checking (e.g. the required order of vectors for compatibility with the *Relocatable Vectors* option) regardless of whether *Automatically Position Vectors* is selected. It is not a processor hardware option; it is used to adjust vector addresses which do affect the hardware.

Related Links

Automatically Position Vectors on page 54

2.8.4 Relocatable Vectors option

Relocatable Vectors Option	Selected

By default, vector addresses are fixed at processor configuration time and cannot be changed.

This option adds the following capabilities:

- **Static Vectors Group**: the 2 vectors in this group are the reset vector and, if configured, the memory error vector. As before, when a processor configuration is created, addresses are determined for all vectors based on the memory layout. If the relocatable vectors option is selected, an alternate address is also specified at configuration time. The auto-placement algorithm (if enabled) will try and choose an appropriate instruction memory. When the XPG builds the processor, both primary and alternate addresses will be in the generated output. By default, at reset, the processor will jump to the "primary" configured reset address. Alternatively a pin on the processor can be asserted to cause it to jump to the alternate reset address.

  When software is generated for the processor, you have the option of placing the reset handler code at the primary address or at the alternate address. If you plan to make use of this feature, you should build both software configurations so you can link binaries with the corresponding handler code.

  The hardware implementation of the relocatable vectors option places constraints on the ordering of vectors in memory (if memory errors are configured, the vector must be after the reset vector), and the vectors must be contained within a 4K block of memory.
- **Dynamic Vectors Group**: the other exception vectors are collectively referred to as the dynamic group because their location in memory can be changed at runtime. These vectors must all be in the order shown above (if configured), and must be contained within a 4K block of memory.

Consult the *Xtensa System Software Reference Manual* and *Xtensa Linker Support Packages (LSPs) Reference Manual* for more information on how to use these relocation features with XTOS and in your system software.

Note: If this is selected, carefully review the *Alternate Static Vector Base Address*. Xplorer attempts to choose a good default, but when memories are changed a good default for this is not necessarily obvious

2.8.5 Alternate Static Vector Base Address

Alternate address for the static vectors (reset and optionally memory errors) which will be used if the appropriate pin is asserted.

Alternate Static Vector Base Address	0xc0000000

Relocatable vectors are arranged with offsets within a 2K byte aligned block of memory. This address is the base of that block for the reset vector and if configured also the memory error vector. As an example, most of the Diamond processors have the primary reset in System ROM, and the alternate reset in Instruction RAM to support different kinds of processor instances.

Note: Xplorer will attempt to choose an appropriate location (e.g. IRAM if confgured), but you should review it carefully because your anticipated use of the alternate location cannot be predicted.

2.8.6 External Reset Vector

External Reset Vector	Not Selected

The option Enable Relocatable Vectors adds an alternate reset address to the processor such that an external pin decides which of two statically configured reset addresses is used. A typical use case for this is to have one reset in System ROM and an alternate reset in an IRAM or IROM.

The option External Reset Vector adds runtime configurability to the alternate reset logic by adding a set of external address pins such that the reset address can be driven to the processor as it comes out of reset.

2.8.7 Default Dynamic Vector Group Vector Base

Default Dynamic Vector Group VECBASE	0xe8080400

This value is presented for information only. As vector addresses are configured, this address is computed to be an appropriately aligned address such that all the dynamic vectors are contained within the maximum relocatable vector block size.

This VECBASE register can be either initialized by an LSP, or it can be written to dynamically. Either way, generally vectors will need to be built at the alternate locations using an appropriate LSP such that they can be appropriately loaded. See the *Xtensa Linker Support Packages (LSPs) Reference Manual* for more information.

2.8.8 Static Vectors

Table 33: Static Vectors

Vector	In Memory	Address	Prefix Bytes	Size Bytes
Reset vector	Instruction RAM 0	0xe8080000	0x0	0x300

The static vector group comprises the reset vector, and if configured also the memory error vector.

If Automatically Position Vectors is enabled (recommended), then these vectors will be automatically positioned at the start of an appropriate memory - typically System ROM, or if that is not available then Instruction RAM. In this mode, the address of vectors cannot be directly edited.

Related Links

Automatically Position Vectors on page 54

Dynamic Vectors on page 57

2.8.9 Dynamic Vectors

Table 34: Dynamic Vectors

Vector	In Memory	Address	Prefix Bytes	Size Bytes
Window vector base	Instruction RAM 0	0xe8080400	0x0	0x178
Level 2 vector	Instruction RAM 0	0xe8080580	0x8	0x38
Level 3 vector	Instruction RAM 0	0xe80805c0	0x8	0x38
Level 4 vector	Instruction RAM 0	0xe8080600	0x8	0x38
Level 5 vector (Debug)	Instruction RAM 0	0xe8080640	0x8	0x38
NMI vector	Instruction RAM 0	0xe80806c0	0x48	0x38
Kernel vector	Instruction RAM 0	0xe8080700	0x8	0x38
User vector	Instruction RAM 0	0xe8080740	0x8	0x38
Double vector	Instruction RAM 0	0xe80807c0	0x48	0x40

The dynamic vector group comprises the window vector group, the level vectors and the NMI, Kernel, User and Double vectors.

If Automatically Position Vectors is enabled (recommended), then these vectors will be automatically positioned at the start of an appropriate memory - typically System RAM, or if that is not available then Instruction RAM. In this mode, the address of vectors cannot be directly edited.

Related Links

Automatically Position Vectors on page 54

Static Vectors on page 56