Azurite Core Datasheet

Release: 1.1.0

Last update on : Tuesday 10th May, 2022

Table of contents

List of figures

List of tables

Chapter 1

Overview

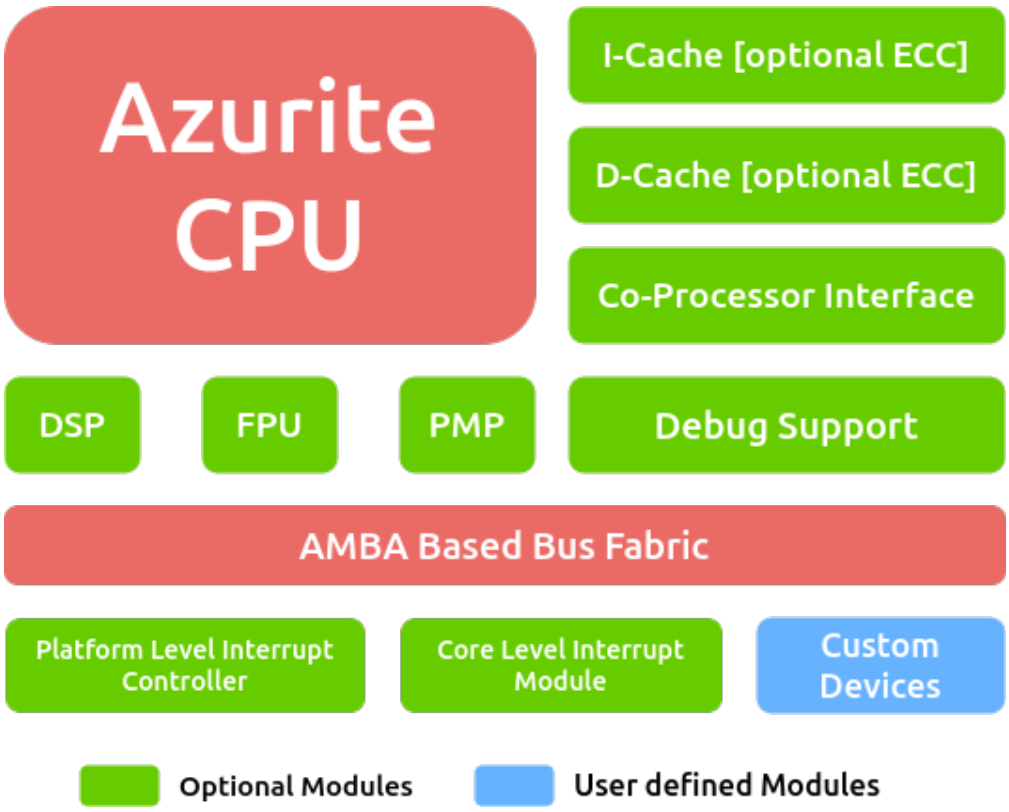

Fig. 1.1: Azurite Processing System

Azurite is a fully configurable and synthesizable Core Generator capable of generating production quality RTL based on the open RISC-V ISA. The core generator can produce variants of a commercial grade 2-stage in-order core supporting the RV[64/32]GCU (or its subsets) extensions of the RISC-V ISA, from the same high-level source code. Azurite leverages the high level abstraction offered by Bluespec System Verilog to build highly parameterized, compact and powerful library components (like arithmetic units, branch predictors, caches, mmu, etc) that can be seamlessly integrated to create a solution catered to your needs. Fig. 1.1 shows the system level diagram of the Azurite core.

The configuration to the generator is provided through a very simple and easy to modify YAML file. The core generator uses a python script to generate the necessary environment variables and macros required to generate the RTL. The generated RTL is completely synthesizable and can be directly integrated into conventional VLSI flows for FPGA, ASIC

or verification. Fig. 1.2 shows the basic flow of the Azurite core generator.

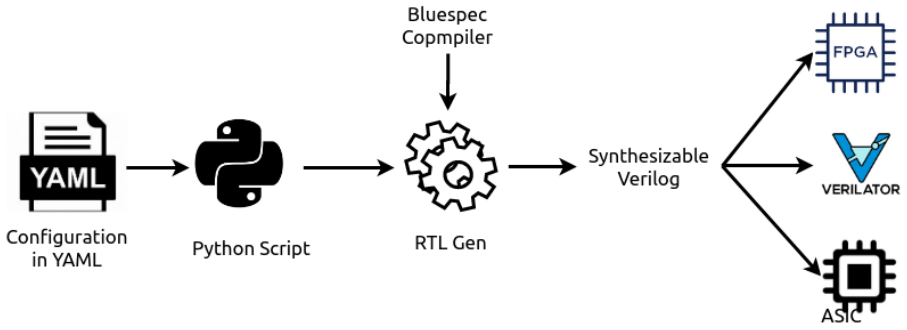

Fig. 1.2: Azurite Core Generator Flow

The various design instances generated through Azurite can serve domains ranging from sensor fusion, power management, motor control, Low-power MCUs, health wearable monitors, smart IoT, connected toys, etc. The extreme parameterization of the design in conjunction with using an HLS like Bluespec, it makes it easy to add new features and design points on a continual basis.

A summary of key features inlcude :

- Support for RV[32|64]IMAFCUZicsr_Zifencei (compliant with the latest release of the spec)
- Enhanced fetch micro-architecture to provide higher performance with higher code-density support
- Configurable Multiply/Divide Units. Can be configured for higher performance or low area.
- Floating Point Unit (FPU), an IEEE-754 compliance Floating-Point Unit with single precision support
- Highly configurable and Latency optimized Instruction and Data Caches
- Includes a light-weight, non-intrusive and a full-featured Debug support over JTAG
- Configurable Hardware Breakpoints triggered by instruction/data address/content match, interrupts, exceptions and instruction count.
- A fully configurable RISC-V Privileged architecture. Ability to define and control the behavior of every CSR without breaking compliance to the ISA.
- Physical Memory Protection support as per official ISA spec to provide protection across privilege levels.

Chapter 2

Core Level Features

- 2 stage in-order pipeline
- 64/32-bit variants available
- Configurable to support the following RISC-V ISA extensions: RV[64/32][IMAFDCUZicsr_Zifencei]
- Optional pre-decoded cache to reduce critical path in decode.
- Optional support for Bitmanipulation Extensions:
  - Ratified extensions : Zba, Zbb, Zbc and Zbs
  - Unratified extensions : Zbp, Zbe, Zbf, Zbm, Zbr and Zbt
- Optional support for Packed-SIMD Extensions
- Support for enhanced RISC-V privilege features:
  - Vectored interrupts and support for external interrupt controller
  - Programmable trap vector base
  - Atomic interrupt enable/disable feature support
- Supports IEEE-754 based single and double precision Floating Point operations in hardware.
  - The design of the FPU (Floating-Point-Unit) is based on Hard-float. Therefore leverages the berkely recoded-format for efficient implementation to support sub-normals.
  - Single unit per precision to handle fused-multiply-add, multiply, addition and subtraction ops. These units can be configured to meet high-frequency requirements since they are retime-enabled.
  - Runs at 1:1 core/FPU clock ratio.
  - A separate floating point registerfile is used to store floating point operands (in recoded format)
  - Control the number of in-flight instructions by controlling pipeline depths of each module.
- Integrated integer Multiply/Divide units
  - A fully pipelined multiplier. The multiplier is retime-enabled thereby allowing a configurable pipeline depth at build-time.
  - The divider implements an iterative algorithm, with an early out mechanism for corner cases.
- A flexible Co-processor interface to provide closely-coupled integration of custom accelerators.
  - based on on the RoCC specification
  - implements a fire-wait or a fire-and-forget protocol based on the instruction dependency in the program order.
- Supports AXI-4 and AXI-4 Litebus protocols
- Configurable L1 Caches
  - Separate instruction and data caches which can be configured with upto 4-ways and 32KiB
  - Write-back with Write-allocate caches
  - Data is implemented as RegFiles for higher throughput.
  - Can be disabled/enabled through software
  - Configurable replacement policies: Pseudo LRU, Random, Round-Robin
  - Data Cache has support for atomic extension operations.
- Physical Memory Protection Unit (PMP):
  - Configurable number of regions : upto 16 can be protected.

  - Can protect upto a granularity of 4 bytes for the 32-bit core and 8 bytes for the 64-bit core.
  - Certain regions can be locked for protection against machine level access as well
- Daisy Chained CSRs optimized for frequency and scaling
  - Allows adding of custom CSRs without having to compromise on frequency or features.
  - Can instantiate up to 32 performance counters
- Debug support based on the RISC-V Debug spec v1.0
  - non-intrusive and light-weight implementation
  - interrupt based indication mechanism
  - Includes a program-buffer to execute custom code when halted (in debug-mode)
- Trigger support for data and address.
- Includes Performance counters for :
  - statistics of different types of instructions based on extension
  - statistics of Caches (like number of hits, misses, writebacks, etc)
  - statistics of traps (number of interrupts, exceptions, etc)

Chapter 3

Benchmarking the Core

The max DMIPS of the Azurite core is **1.9 DMIPs/MHz.**

Note: Make sure you are using gcc 11.1.0 or above to replicate the following results.

3.1 Benchmarking Dhrystone

The following numbers have been obtained via simulation where the number of ITERATIONS was fixed at 10000. The riscv-gnu-toolchain was used to compile the program. The versions used have been populated in the table.

Flags used for compilation:

```
-mcmodel=medany -static -std=gnu99 -O2 -ffast-math \
-fno-common -fno-builtin-printf -march=rv64$(march) -mabi=lp64d \
-w -static -nostartfiles -lgcc
```

The following table provides the DMIPs/MHz numbers for varios configurations for 10K iterations of dhrystone:

HW ISA CONFIG	march	mabi	uArch Configs	gcc version	DMIPs/MHz
RV64IMASU	rv64ima	lp64	Default	(g5964b5cd727) 11.1.0	1.9

Chapter 4

Core Pipeline Architecture

This chapter discusses the micro-architecture and features of the core pipeline. All addresses are treated as physical address as supervisor mode not supported The core pipeline consists of the following stages:

- Stage 1
  - PC generation
  - Instruction Fetch
  - Instruction Decode
  - Execution (ALU)
- Stage 2
  - Memory operations
  - Write Back (a.k.a Commit)
  - Trap Handling
  - System Instruction
  - Multi-Cycle Operations

A block diagram of the pipeline is shown in Fig. 4.1

4.1 STAGE-1

The current value of the PC (Program Counter), is sent to the Instruction Cache to retrieve the relevant instruction. The instruction cache returns with a pre-decoded information of the instruction if pre-decode is enabled at build-time, else just the instruction is returned. In case if compressed-extension is enabled, then a compressed instruction is decompressed before returning from the instruction cache.

The decode unit decodes the 32-bit instruction received from the instruction cache and also performs operand fetch from the register-file. The core consists of 32 general purpose integer registers of XLEN (32/64 bit) size. If floating point support is enabled in hardware, then another 32 floating point registers of FLEN (32/64 bit) size are also available. The integer register-file consists of two read-ports and one write port. The floating point register-file on the other hand requires three read-ports and one write port (to support floating point multiply-accumulate operations). To reduce operation latency in the pipeline, the register file also supports full-bypass.

All interrupts (local or external) are detected in this unit. Illegal traps and traps received from the previous stage are captured here and processed for the next stage.

When a WFI (Wait for Interrupt) instruction is detected, the pipeline is stalled from the next cycle . The pipeline resumes functionality only when an interrupt (local or external) is detected.

If the decoded instruction points to an instruction from the arithmetic class of the base ISA, then its executed in this stage itself. All multi-cycle arithmetic ops (like integer mul/divide, floating point, etc) are offloaded to the relevant

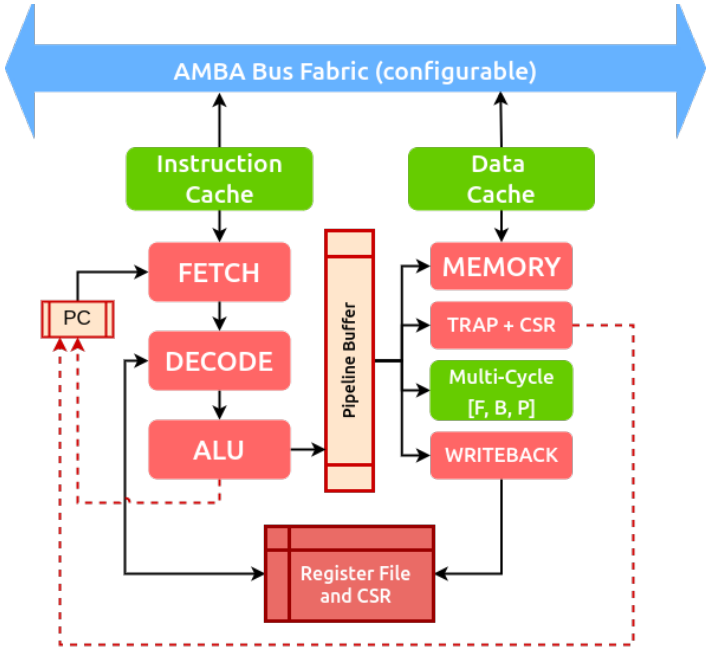

Fig. 4.1: Azurite Pipeline Diagram.

multi-cycle operation unit.

Control instruction resolution also occurs in this stage. The comparison logic of the Arithmetic ops is re-used to detect if a branch is taken or not. The target address for all control instructions is calculated using a dedicated adder. The ALU generates a re-direction signal for the PC either if a branch is taken or unconditional jumps. The re-direction also involves sending the correct target address to the PC register.

4.2 STAGE-2

All Memory ops are initiated and completed in this stage. Memory ops include : Load, Store, Atomic and Fence operations.

The result of arithmetic ops completed in this cycle or the previous stage are written back to the registerfile.

If the instruction from stage 1 is *Trap* type, a redirect signal is sent to PC generator along with the redirect address.

Chapter 5

Modes of Operation

The core supports the following standard modes of operations:

- **Debug**: This is highest level of operation which provides access to all features of the core and the system. Typically, this mode is used for software development and bring-up phases. This mode is available only if the debugging option is enabled at design time.
- **Machine**: This is the highest mode of software execution and is mandatory in all variants generated by the core. The code running in machine mode is inherently trusted and has access to all implementation resources.
- **User**: This is lowest mode of operation where user applications are executed.

Chapter 6

Trap Handling

Traps within the Azurite core can be either synchronous traps or interrupts. The synchronous traps are generated by exception causing instructions through various stages of the pipeline. Examples - Instruction Access Fault, Page Faults, Mis-aligned Faults, Illegal Instructions.

The Azurite core supports three basic classes of interrupts: timer interrupts, software interrupts and external interrupts. The timer interrupt is generated through an on-chip timer module. The software interrupt is generated by setting specific bits within the Core-Level-Interrupt (CLINT) module. The external interrupts are generated by a Platform Level Interrupt Controller (PLIC) after capture of external events.

Each of the above class of interrupts can be generated in any of the operation modes - Machine or User. Azurite cores also provide the ability to delegate handling of some of the traps/interrupts to a lower operation mode using standard delegation CSRs.

The core also supports vectored mode of interrupts, where the trap-vector is generated based on the cause of the trap, allowing fast access to interrupt routines.

Chapter 7

Memory Subsystem

The Azurite memory subsystem includes the L1 instruction and data caches.

The caches are designed to use single ported (i.e. 1RW) RAM structures for better delay, area and power. The access to the RAMs require two cycles: one cycle for the actual read of the RAM, and another cycle for tag comparison and hit determination

The caches also include an array of fill-buffers which hold the lines coming from the lower level memories. The entries in the fill-buffer are released into the RAMs either when the fill-buffer is full or when there is an opportunity where the cache is not receiving requests from the core pipeline.

In case of the data cache, the write-policies followed are write-back and write-allocate. The fill-buffers in the data-cache also hold the lines on which stores need to be performed.

7.1 Feature List

Following is a quick list of features for the L1 caches.

- The caches follow a write-back policy. This reduces the traffic to the next-level caches (if any).
- The caches are designed to use Single-ported SRAMs/BRAMs (1RW configuration) for data and tag arrays. This means that at any given cycle the SRAMs can either perform a read or a write. This choice improves the area,latency and power consumption of the entire cache since SP-SRAMs are the lightest-configurations available with a foundry/FPGA.
- The caches also assume that the SRAMs follow a *NO_CHANGE* policy for the read-ports, where only a read-access can cause a change in the output ports. More info on this can be found under the *Operating Mode* subsection of Chapter 3 of the *Block Memory Generator v8.3, LogiCORE IP Product Guide* from Xilinx.
- These are blocking caches. If a miss is encountered, the caches can latch only one more request from the core which will get served only after the previous miss has been served.
- On a cache-line miss, the caches expect the fabric/bus to respond with the critical word first.
- Round-robin, PLRU and Random replacement policies are supported (which need to be defined at compile time).
- The valid bits are stored as an array of registers instead. This enables a single-cycle flush operation.
- The caches without ECC do not generate any exceptions internally. Access exceptions are received from the fabric.

Chapter 8

Bus Interface and Fabric

The Azurite core can be configured to implement any of the following bus interfaces along with a choice of fabric (crossbar, simple-bus or Mesh).

- AXI-4: A high performance AMBA protocol for fast memory transfers and high bandwidth.
- AXI-4 Lite: A low overhead version of AXI-4 targeted for slow peripherals and devices.

Each of the above protocols and fabrics come with highly parameterized bridge adaptors to transact between different protocols and varying bus-sizes, thereby optimizing area, power and frequency.

Chapter 9

Interrupt Controller

The Azurite system also comes with a Platform Level Interrupt Controller (PLIC). The PLIC routes all the external interrupts to the core.

- The PLIC allows each interrupt to be enabled or disabled
- Each interrupt also has a runtime or design time configurable priority. Higher priority interrupts are serviced first
- The software application can also define a priority threshold. Interrupts with priority lower than the threshold will not latch an interrupt to the system
- PLIC supports both level and edge-triggered interrupts

This is an optional module and users can choose to connect their own interrupt controller to drive the external-interrupt source to the core.

Chapter 10

Debug Support

The Azurite core includes a JTAG based system level debugger, based on the RISC-V Debug spec [Spe20a]. It can be used for debugging of applications and kernel code. In addition to standard operating modes (machine and user), enabling debug support adds the debug mode of operation to the core.The Debug mode is entered through debug exceptions: *halt request from the host, executing ebreak instructions, single-step execution, breakpoints*. The debug mode can be exited by executing a **dret instruction** or through a resume request from the host.

Within the debug mode the host has access to all the integer registers, floating point registers (if F or D extensions are enabled at design time) and all the CSRs through a dedicated side-band access. Moreover, the host has access to all memory mapped registers of the devices that are in the SoC through the master bus interface.

The Debug support also includes host side GDB and OpenOCD software (ports of the standard source) to load and debug programs.

Chapter 11

Software Ecosystem

Since the Azurite core is based on the RISC-V ISA, the users can leverage the entire open source RISC-V toolchain which includes the following:

- Object toolchain:
  - Binutils
  - LLVM
  - Cranelift
- Debugging:
  - GDB
  - OpenOCD
  - Platform IO IDE
- Compilers and Libraries
  - GCC
  - Clang/LLVM
  - Glibc
  - Newlib
- OS and OS Kernels:
  - FreeRTOS
  - Zephyr RTOS
- Compilers and runtimes for other languages:
  - Go
  - OCaml
  - Rust
- Machine Learning/AI
  - TensorFlow Lite

Chapter 12

Build Time Configurations

The Azurite Core allows a wide variety of features to be customized based on the target application. Table 12.1 captures most of the key configurations that can be selected when the core is synthesized and implemented. Note that the choice of these options does not affect the software application porting.

Table 12.1: Azurite build time configurations

Parameter	Choices	Description
Core Level Configuration		
Processor Width (XLEN)	32/64	implement a 32-bit or a 64-bit core. Defines the size of the register files and interface to the caches.
ISA Extensions (ISA)	Combination of : A, M, F, C, U, Zifencei, Zicsr	• A: enable Atomic extension support • M: enable MulDiv extension support • F: enable Single precission support • C: enable Compressed extension support • U: enable User extension support • Zifence : enable fence for instructions • Zicsr : enable CSR operations
System Level Configurations		
Physical Address (PADDR)	Integer	Size physical address on the SoC bus
Debug Vector	PADDR sized integer	the address of the where the self-loop for the debugger lies
Reset PC	Integer	The Value of the Reset PC from where instructions will be fetched
Debug support	Enabled/Disabled	Debug support available on the system and core or not.
Co-Processor Interface	Enabled/Disabled	If enabled instantiates a cutom co-processor inter-face to the core.
Multiplier/Divider Configuration		

Continued on next page

Table 12.1 – continued from previous page

Parameter	Choices	Description
Multipler Stages	integer < XLEN	the number of cycles the multiplier takes for execution
Divider Stages	integer < XLEN	the number of cycles the divider takes for execution
Floating Point Configuration		
Inflight instructions	ordering_depth > 0	Number of inflight Floating point instructions in the pipeline
SPFMA Stages	in > 0 out > 0 configuration of each stage	Number of retime registers at input Number of retimr registers at output Configuration of each stage
DPFMA Stages	in > 0 out > 0 configuration of each stage	Number of retime registers at input Number of retimr registers at output Configuration of each stage
Instruction Cache Configurations		
Enable	True/False	if the I$ should be instantiated within the design or not.
On Reset	Enabled/Disabled	if the I$ should be enabled or disabled on reset.
Sets	Integer	Number of sets per way of the I$
Block Size	Integer	Number of 32-bit words per cache-block
Ways	Integer	Number of ways in the I$
Replacement Policy	PLRU, RoubdRobin, Random	The replacement policy to be implemented within the I$
Banks for Data RAMs	Integer	Number of banks the data blocks should be divided into
ECC	Enabled/Disabled	Is ECC enabled or disabled for the I$
Data Cache Configurations		
Enable	True/False	if the D$ should be instantiated within the design or not.
On Reset	Enabled/Disabled	if the D$ should be enabled or disabled on reset.
Sets	Integer	Number of sets per way of the D$
Block Size	Integer	Number of words per cache-block
Ways	Integer	Number of ways in the D$
Replacement Policy	PLRU, RoubdRobin, Random	The replacement policy to be implemented within the D$
Banks for Data RAMs	Integer	Number of banks the data blocks should be divided into
ECC	Enabled/Disabled	Is ECC enabled or disabled for the D$
PLIC Configurations		
Number of interrupts	Integer < 1024	Number of interrupts that can be processed by the PLIC
Number of Priority Levels	Integer	Number of priority levels within the PLIC
Number of NMI	Integer	Number of NMI interrupts connected to PLIC
Phyiscal Memory Protection (PMP)		
Enable	True/False	If PMP support is present or not

Continued on next page

Table 12.1 – continued from previous page

Parameter	Choices	Description
Entries	Integer < 16	Number of regions for pmp
Granularity	Integer >= 4	Granularity in terms of bytes. Minumim is 4 for 32-bit cores and 8 for 64-bit cores

Bibliography

[Spe20a] RISC-V ISA Debug Specification. *https://riscv.org/specifications/debug-specification/*, 2020.

[Spe20b] RISC-V ISA Privileged Specification. *https://riscv.org/specifications/privileged-isa/*, 2020.

[Spe20c] RISC-V ISA Unprivileged Specification. *https://riscv.org/specifications/isa-spec-pdf/*, 2020.