ZAP Processor Core
User Guide and Datasheet

NOTE
The project is in an experimental state.

Contents

1 Introduction....2
1.1 CPU Clock and Reset....2
1.2 Block Diagram....2
1.3 Pipeline Overview....2
1.4 Features....4
1.5 Branch Prediction Mechanism....4
1.7 Cache/TLB Overview....5
1.8 Running Simulations....5
1.8.1 Creating test cases....5
1.8.2 Simulating test cases....6
1.9 Deliverables....6
2 IO Ports and Processor Configuration....7
2.1 Interface Ports....7
2.2 Configuration....7
3 CP #15 Commands....9

1 Introduction

ZAP is a synthesizable open source 32-bit RISC processor core capable of executing ARM®v4T binaries at both the user and supervisor level. The processor features a 10-stage pipeline that allows it to reach reasonable operating frequencies. The processor supports standard I/D cache and memory management that may be controlled using coprocessor #15. Both the cache and TLB are direct mapped. Caches, TLBs and branch memory are implemented as generic fully synchronous RAMs that can efficiently map to native FPGA block RAM to save FPGA resources. To simplify device integration, the memory bus is fully compliant with Wishbone B3. A store buffer is implemented to especially improve cache clean performance.

NOTE
Please use pipeline retiming during synthesis for maximum timing performance.

1.1 CPU Clock and Reset

ZAP uses a single clock called the *core clock* to drive the entire design. The clock must be supplied to the port I_CLK. ZAP expects a rising edge synchronous I_RESET (active high) to be applied i.e., the reset signal must change only on the rising edges of clock. The reset must be externally synchronized to the core clock before being applied to the processor.

1.2 Block Diagram

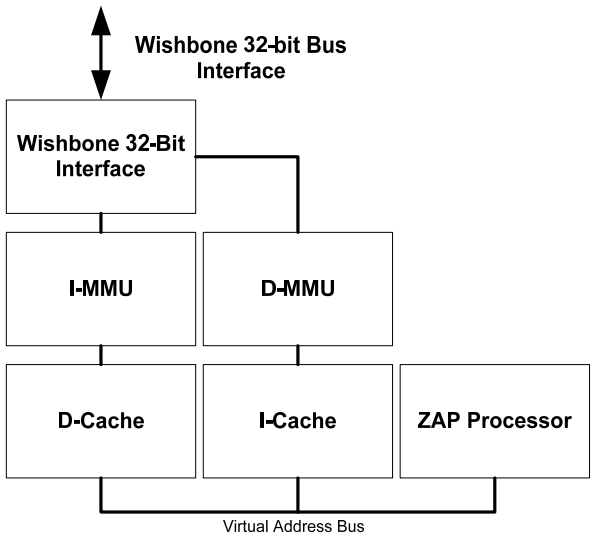

Figure 1. ZAP Block Diagram

1.3 Pipeline Overview

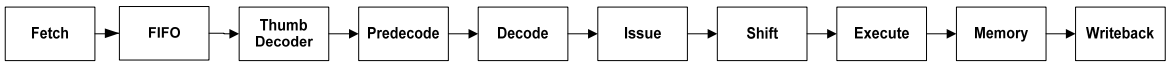

Figure 2. ZAP Pipeline Architecture

The table below briefly describes each stage of the pipeline.

Table 1. Pipeline Description

Stage	Detail
Fetch	Clocks data from I-cache into instruction register. Also branch predictor memory is read out in this stage.
FIFO	Instructions and corresponding PC+8 values are clocked into a shallow buffer.
Thumb Decoder	Converts 16-bit instructions to 32-bit ARM instructions. Instructions predicted as taken cause the pipeline to change to the new predicted target.
Predecode	Handles coprocessor instructions, SWAP and LDM/STM.
Decode	Decodes ARM instructions.
Issue	Operand values are extracted here from the bypass network. In case data from the bypass network is not available, the register file is read.
Shift	Performs shifts and multiplies. Contains a single level bypass network to optimize away certain dependencies. Multiplication takes multiple clock cycles.
Execute	Contains the ALU. The ALU is single cycle and handles arithmetic and logical operations.
Memory	Clocks data from the data cache into the pipeline. Aligns read data as necessary.
Writeback	Writes to register file. Can sustain 2 writes per clock cycle although the only use for the feature is accelerate LDR performance in the current implementation.

ZAP features a 10 stage pipeline. The pipeline has an extensive bypass network to minimize pipeline stalls. A load accelerator allows data to be forwarded from memory a cycle early. Most non-multiply instructions can be executed within a single clock tick with no stalls. Exceptions to this rule are when multiplies or non-trivial shifts are used.

The following code takes 3 cycles to execute because R1 needs to be shifted and is not available until the first instruction enters the ALU:

ADD R1, R2, R3
ADD R4, R5, R1 LSL R2

If the second register is not source shifted by a register that depends on the previous instruction, a data dependency check may be relaxed (Register R9 for the second instruction can be obtained in issue itself so nothing is blocking the second instruction from using the shifter) and thus the following code takes 2 cycles to execute:

ADD R1, R2, R3 LSL R5
ADD R4, R1, R9 LSL R2

Another feature of the pipeline is that it can issue memory operations with writeback in a single cycle. The following instructions takes 2 cycles to execute assuming a perfect cache.

LDR R0, [R1, #2]!
ADD R1, R3, R4 LSL R1

The pipeline feedback unit is heavily optimized to minimize pipeline stalls. The sophisticated feedback network results in a slight reduction in maximum frequency. However, net performance should improve since unwanted stalls are eliminated.

1.4 Features

- Fully synthesizable Verilog-2001 core.
- Store buffer for improved performance.
- Can execute 32-bit ARMv4 and 16 bit Thumb® v1 code. Thumb support is experimental.
- Wishbone B3 compatible interface. Cache unit supports burst access.
- 10-stage pipeline design. Pipeline has bypass network to resolve dependencies.
- 2 write ports for the register file to allow LDR/STR with writeback to execute as a single instruction. Note that the register file is implemented using flip-flops.
- Branch prediction supported. Uses a 2-bit state for each branch.
- Split I and D writeback cache (Size can be configured using parameters).
- Split I and D MMUs (TLB size can be configured using parameters).
- Base restored abort model to simplify data abort handling.

1.5 Branch Prediction Mechanism

ZAP uses a relatively simple branch prediction mechanism (Note that the branch table block RAM is read in Fetch):

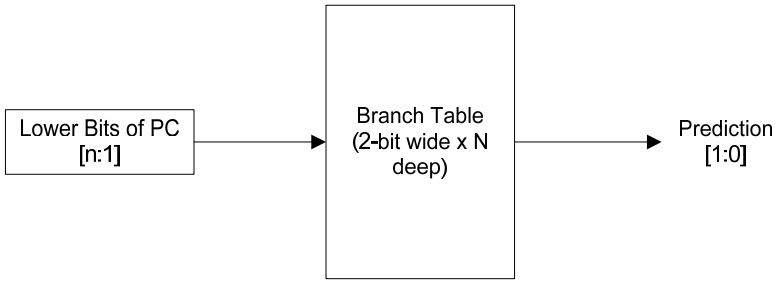

Figure 3. Branch Prediction Mechanism

Note that when using ARM code, the amount of branch table entries is cut by half since the lower 2 bits of PC are 0. A simple state machine is used to reinforce or modify the status of a branch:

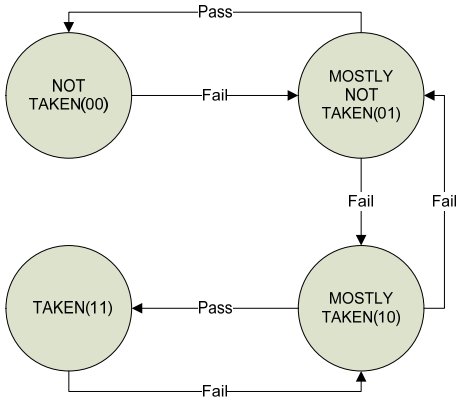

Figure 4. Branch State Machine

The pass and fail signals are generated from the ALU.

1.7 Cache/TLB Overview

ZAP uses direct mapped cache (separate I/D) that is virtual. For high performance, the cache is writeback. To enable writeback, each cache line has a dirty bit. The size of each cache line is fixed at 16 bytes. Since ARMv4 specifies three kinds of paging schemes: section, large and small pages, 3 TLB block RAMs are employed which are also direct mapped. ZAP has independent instruction and data caches/TLBs.

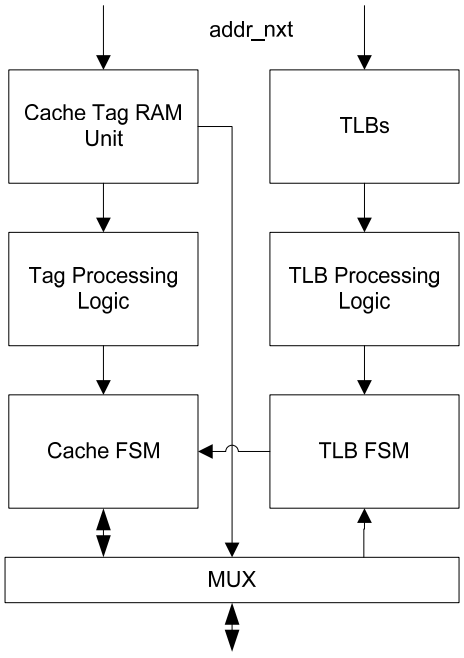

Figure 5. Cache/MMU High Level View

NOTE
The cache should be enabled as soon as possible for good performance because the memory subsystem is efficient for burst transactions.

1.8 Running Simulations

1.8.1 Creating test cases

To create a test case called *N*, create a set of three files `N.s`, `N.c` and `N.ld` (assembly, C file and the linker script) in the directory `$ZAP_HOME/sw/tests/N/`

The assembly file serves as the startup boot assembly code need to launch the C file. The linker is needed to correctly arrange parts of the generated binary.

A couple of test cases are provided by default (*arm_test* and *factorial*).

1.8.2 Simulating test cases

To run simulations using the scripts provided, you will need a Linux machine with ARM development tools (bare-metal), Icarus Verilog 10.0 or above and GTKWave.

Enter `$ZAP_HOME/hw/sim` and run

```
perl run_sim_gui.pl
```

which will launch a GUI where configuration can be done. The form is pre-filled to indicate the syntax needed to fill it. Once filled, an XTerm window is launched in which Icarus Verilog is invoked (This is done by another script `run_sim.pl` in the same directory. To bypass the GUI and run the base script directly, execute `perl run_sim.pl` (without any arguments) for a list of options).

1.9 Deliverables

NOTE
Ensure that the environment variable $ZAP_HOME is set to point to the root directory of the project.

Table 2. ZAP Processor Related Deliverables

Deliverable	Description
Synthesizable Verilog-2001 RTL description of the ZAP processor.	RTL description of all modules needed to build the processor on an FPGA. The RTL is coded in Verilog-2001. All RTL files needed to build the processor can be found in `$ZAP_HOME/hw/rtl/zap/`
Sample code to test the processor.	C and assembly code to test the processor. Test cases may be found in `$ZAP_HOME/sw/tests/<test_name>`
Verilog-2001 testbench. This bench creates a simple environment consisting the processor and some memory.	The testbench instantiates the ZAP core and simulates external memory in order to test the processor. Test bench files can be found in `$ZAP_HOME/hw/tb/zap/`
Simulation scripts. Used to quickly test the processor without needing to simulate the entire SoC.	Perl/C-shell scripts to simplify running the testbench. Scripts can be found in `$ZAP_HOME/hw/sim/`

2 IO Ports and Processor Configuration

2.1 Interface Ports

Table 3. Interface IO Ports

Signal Name	IO	Description	Comments
I_CLK	I	Core clock.	The entire processor is built using synchronous rising edge based design techniques.
I_RESET	I	Core reset.	Active high. Must be synchronous to the rising edge of I_CLK.
O_WB_CYC	O	Wishbone CYC signal.	Outputs are purely synchronous with the rising edge of I_CLK.
O_WB_STB	O	Wishbone STB signal.	
O_WB_ADR[31:0]	O	Wishbone 32-bit address.	
O_WB_SEL[3:0]	O	Wishbone byte lane enables.	
O_WB_WE	O	Wishbone write enable.	
O_WB_DAT[31:0]	O	Wishbone write data.	
O_WB_CTI[2:0]	O	Wishbone cycle type indicator. 000 – Classic. 010 – Incrementing burst. 111 – End of burst. The interface shall only generate one of the above 3 codes.	
I_WB_DAT[31:0]	I	Wishbone read data.	
I_WB_ACK	I	Wishbone acknowledge.	
I_IRQ	I	IRQ Interrupt	Active high level sensitive.
I_FIQ	I	FIQ Interrupt	Active high level sensitive.

2.2 Configuration

ZAP may be configured using top level Verilog parameters. You can override these parameters when you instantiate of the core in your SoC. Ensure that you read the notes listed after the table below carefully else the design may fail to compile.

Table 4. Processor Top Level Parameters

Parameter	Description	Notes
DATA_CACHE_SIZE[31:0]	Data cache size in bytes.	1
CODE_CACHE_SIZE[31:0]	Instruction cache size in bytes.	1
CODE_SECTION_TLB_ENTRIES[31:0]	Section TLB entries (CODE).	2
CODE_SPAGE_TLB_ENTRIES[31:0]	Small page TLB entries (CODE).	2
CODE_LPAGE_TLB_ENTRIES[31:0]	Large page TLB entries (CODE).	2
DATA_SECTION_TLB_ENTRIES[31:0]	Section TLB entries (DATA).	2
DATA_SPAGE_TLB_ENTRIES[31:0]	Small page TLB entries (DATA).	2
DATA_LPAGE_TLB_ENTRIES[31:0]	Large page TLB entries (DATA).	2
FIFO_DEPTH[31:0]	Depth of the fetch buffer in the pipeline.	3

BP_ENTRIES[31:0]	Depth of the branch predictor memory.	3
STORE_BUFFER_DEPTH[31:0]	Set the depth of the store buffer. Do not set it to a value less than 16.	4

NOTE.

1. Should be a power of 2 and greater than 128 bytes.
2. Should be a power of 2 and must be at least 2 entries.
3. Should be a power of 2 and must be greater than 2.
4. Depth must be 16 or more and a power of 2.

3 CP #15 Commands

ZAP features an ARMv4 compatible cache subsystem (cache and MMU). This subsystem may be configured by issuing commands to specific CP #15 registers using coprocessor instructions. A list of supported CP #15 commands/registers are listed in the table below:

WARNING
In particular, cleaning and flushing of specific locations is not supported. The OS should avoid issuing such commands.

Table 5. Supported CP #15 Commands/Registers

Register	Name	Description	Notes
0	ID	[23:16] – Always reads 0x01 to indicate a v4 implementation. Other bits are UNDEFINED.	1
1	CON	[0] – MMU enable. [2] – Data cache enable. [8] – S bit. [9] – R bit. [12] – Instruction cache enable. READ ONLY bits are described in note 2. Bits other than the ones specified here and in note 2 are UNDEFINED.	2
2	TRBASE	Holds 16KB aligned base address of L1 table.	
3	DAC	Domain Access Control Register.	
5	FSR	Fault address register.	4
6	FAR	Fault status register.	4
7	CACHECON	Data written to this register should be zero else UNDEFINED operations can occur. **Table 6. CACHECON Control** **Opcode2** / **CRm** / **Description** **000** / 0111 / Flush all caches. **000** / 0101 / Flush I cache. **000** / 0110 / Flush D cache. **000** / 1011 / Clean all caches. **000** / 1010 / Clean D cache. **000** / 1111 / Clean and flush all caches. **000** / 1110 / Clean and flush D cache.	3
8	TLBCON	Data written to this register should be zero else UNDEFINED operations can occur. **Table 7. TLBCON Control** **Opcode2** / **CRm** / **Description** **000** / 0111 / Flush all TLBs **000** / 0101 / Flush I TLB. **000** / 0110 / Flush D TLB.	3

NOTE:

1. Read only. Writes have NO effect.

2. Processor does not check for address alignment ([1] reads 0), only supports Little Endian access, full 32-bit, write buffer always enabled ([7:4] reads 0b0011), does not support high vectors ([13] reads 0) and always has a predictable cache strategy ([11] reads 1) i.e., direct mapped.
3. Reads are UNPREDICTABLE.
4. Only data MMU can update this. For debug purposes, these are RW registers.