# ZAP

## Data Sheet

Issued: May 2017
Version 0.00.01


**! CAUTION !**

The ZAP project is in an EXPERIMENTAL state.



**Trademarks**
ARM is a registered trademark of ARM Ltd.
Xilinx, Spartan and ISE are a registered trademark of Xilinx Inc.

## Contents

# 0 Running Simulations

## 0.1 Creating test cases

A test case N is a set of three files N.s, N.c and N.ld present 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. By default, a factorial test case is provided.

## 0.2 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.

Ensure $ZAP_HOME Enter `$ZAP_HOME/hw/sim` directory 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 Introduction

ZAP is a synthesizable Verilog soft processor that is capable of executing ARMv4T binaries at both the user and kernel level. The processor features separate instruction and data virtual caches whose size is configurable via Verilog parameters. Each cache controller is also fitted with a memory management unit (MMU) that handles virtual to physical address mapping and enforces access security policies. For fast translation, the MMU is equipped with translation buffers whose sizes may be configured using Verilog parameters. The processor features a 10 stage pipeline that allows most instructions to executed within a single clock cycle. On a Spartan 6 speed grade -3 class FPGA, the processor reaches speeds of up to 70MHz. Memory access is done using a standard 32-bit Wishbone B3 compatible interface.

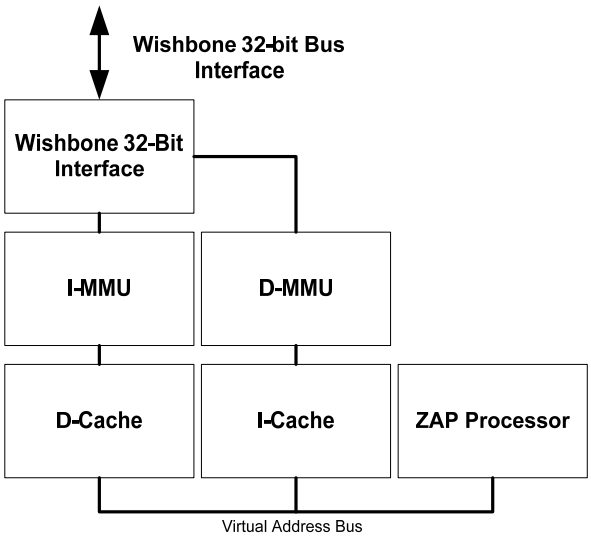


Figure 1. ZAP Block Diagram

**Features**

- High performance RISC processor capable of executing ARMv4T binaries.
- Uses a 9-stage pipelined architecture capable of executing most instructions in a single clock cycle.
- Memory Management Unit (MMU). Supports virtual memory.
- Instruction and data cache for high performance. Caches are direct mapped, virtual and writeback.
- 32-bit Wishbone B3 compatible memory interface.
- Memories exclusively use synchronous reads and writes to allow implementation as standard block RAM.

# 2 Deliverables

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

**Table 1. 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/` |

# 3 Configuration

The processor may be configured using Verilog parameters. You must provide these parameters when instantiating the processor in your SoC.

**Table 2. Processor Configuration 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 |

**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. Should be a power of 2 and must be greater than 2.

# 4 Top Level I/O Ports

**Table 3. Interface Signals**

| Signal Name | Direction | Description | Comments |
|---|---|---|---|
| I_CLK | IN | Core clock. Should be a divide by 2 (frequency) version of I_CLK_MULTIPUMP. | Typically driven from PLL 1x port. |
| I_CLK_MULTIPUMP | IN | Register file clock. Must be the 2x version (frequency) of I_CLK. | Typically driven from PLL 2x port. |
| I_RESET | IN | Core reset. Reset is passed through a reset synchronizer internally. | Must be held high for the device to be reset. Typically driven from the system reset controller. |
| O_WB_CYC | OUT | Wishbone CYC signal. | STB and CYC are always asserted and deasserted at the same time. |
| O_WB_STB | OUT | Wishbone STB signal. | |
| O_WB_ADR[31:0] | OUT | Wishbone 32-bit address. | |
| O_WB_SEL[3:0] | OUT | Wishbone byte lane enables. | |
| O_WB_WE | OUT | Wishbone write enable. | |
| O_WB_DAT[31:0] | OUT | Wishbone write data. | |
| I_WB_DAT[31:0] | IN | Wishbone read data. | |
| I_WB_ACK | IN | Wishbone acknowledge. | |
| O_WB_CTI[2:0] | OUT | Wishbone cycle type indicator.<br>000 – Classic.<br>110 – Incrementing burst.<br>111 – End of burst. | Incrementing bursts are always linear. |

**NOTE**

**The processor expects every Wishbone access to eventually respond with an ACK even if the access failed. The software running must ensure that no illegal accesses are made. This is easy to do if the page tables are set up properly to reflect the state of the system.**

## 4.1 Clock Waveforms

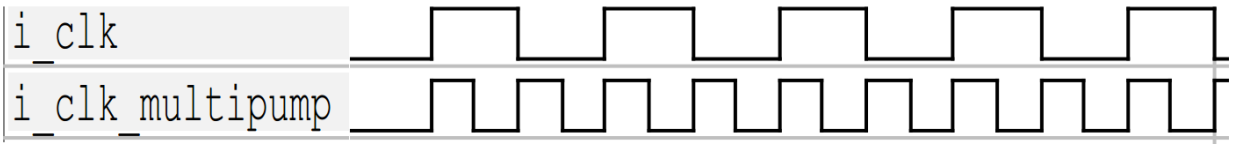


**Figure 2. Clock Waveforms**

A top level diagram of the processor clock connections in an FPGA would appear as shown below:

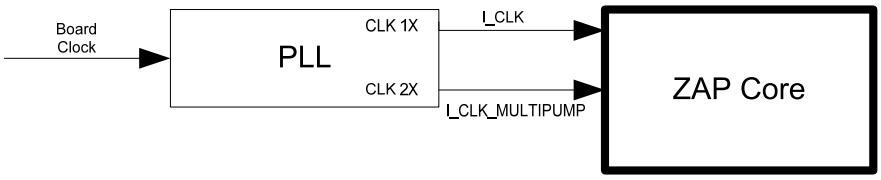


**Figure 3. ZAP Clocking**

# 5 CP15 Registers

ZAP features an ARMv4 compatible memory management unit. According to the specification itself, not all implementations are required to support all of the commands. This chapter lists the CP15 registers supported by the current release of ZAP.

## 5.1 Register List

**Table 4. CP15 Register List**

| Index | Name | Description | Notes |
|---|---|---|---|
| 0 | ID | [23:16] – Always reads 0x01 to indicate a v4 implementation. | 1 |
| 1 | CON | [0] – MMU enable.<br>[2] – Data cache enable.<br>[8] – S bit.<br>[9] – R bit.<br>[12] – Instruction cache enable. | 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.<br><br>**Table 5**<br><br>**Opcode2 / CRm / Description**<br>000 / 0111 / Flush all caches.<br>000 / 0101 / Flush I cache.<br>000 / 0110 / Flush D cache.<br>000 / 1011 / Clean all caches. Same as clean D cache since I cache is read-only.<br>000 / 1010 / Clean D cache.<br>000 / 1111 / Clean and flush all caches.<br>000 / 1110 / Clean and flush D cache. | 3 |
| 8 | TLBCON | Data written to this register should be zero else UNDEFINED operations can occur.<br><br>**Table 6**<br><br>**Opcode2 / CRm / Description**<br>000 / 0111 / Flush all TLBs<br>000 / 0101 / Flush I TLB.<br>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 ([7:4] reads 0b1111), does not support high vectors ([13] reads 0) and always has a predictable cache strategy ([11] reads 1).
3. Reads are UNPREDICTABLE.
4. Only data MMU can update this.

# 6 Pipeline Overview

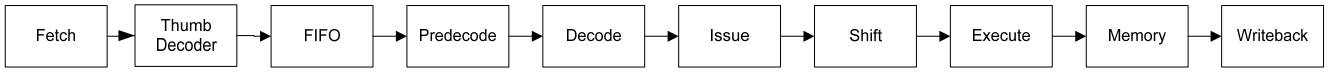


**Figure 4. ZAP Pipeline**

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:

ADD R1, R2, R3
ADD R4, R5, R1 LSL #1

If the second register is not source shifted, a data dependency check may be relaxed and thus the following code takes 2 cycles to execute:

ADD R1, R2, R3
ADD R4, R1, R9 LSL #1

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

| **NOTE** |
| --- |
| The reason for using I_CLK_MULTIPUMP is to allow the register file to have 2 independent write ports and 4 independent read ports. The multi pump clock divides a main processor clock cycle into 2 phases for the register file to operate on. In one phase, write occur and reads in the other. |

**Table 7. Pipeline Description**

| Stage | Detail |
| --- | --- |
| Fetch | Clocks data from I-cache into instruction register. |
| Thumb Decoder | Converts 16-bit instructions to 32-bit ARM instructions. |
| FIFO | Instruction are clocked into a shallow buffer. |
| Predecode | Handles coprocessor instructions, SWAP and LDM/STM. |
| Decode | Decodes ARM instructions. |
| Issue | Operand values are extracted here from the bypass network. |
| Shift | Performs shifts and multiplies. Contains a single level bypass network to optimize away certain dependencies. |
| Execute | Contains the ALU. |
| Memory | Clocks data from the data cache. |

| | |
|---|---|
| Writeback | Writes to register file. |

## 6.1 Branch Prediction Mechanism

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

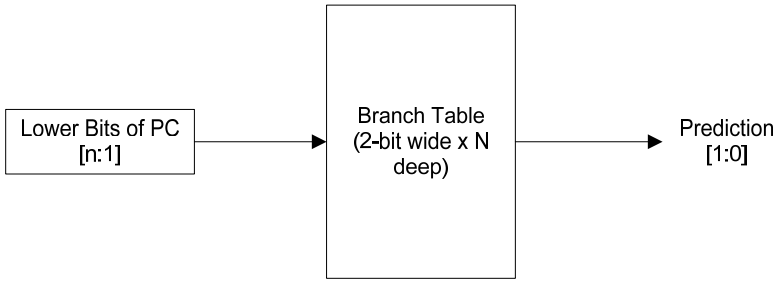


**Figure 5. Branch Prediction Mechanism**

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:

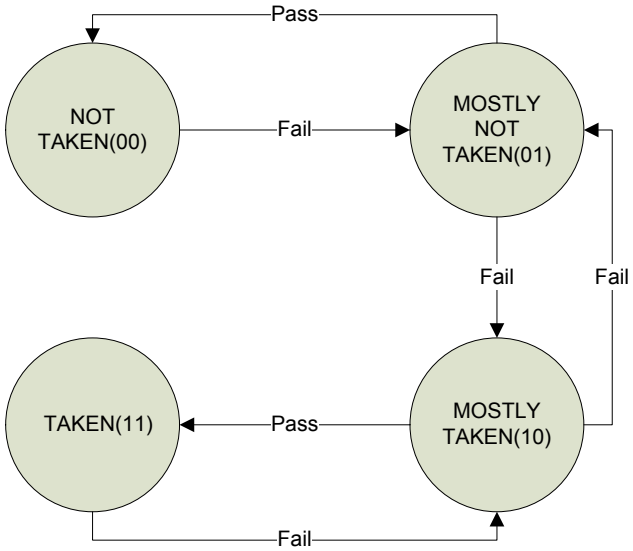


**Figure 6. Branch State Machine**

The pass and fail signals are generated from the ALU.

# 7 Cache Overview

ZAP uses direct mapped cache 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.

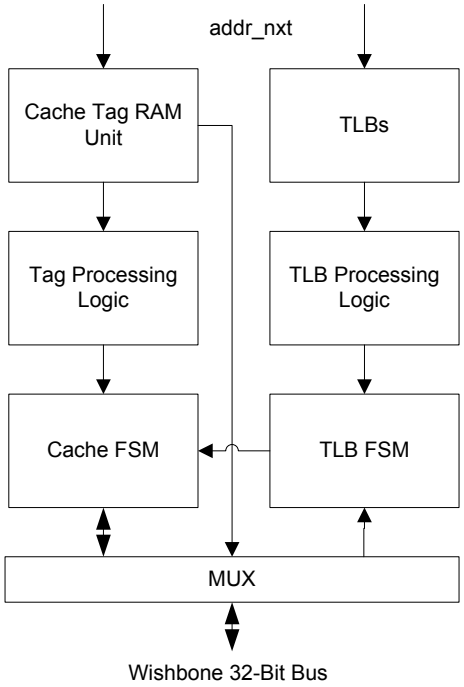


**Figure 7. Cache/MMU High Level View**