The Potato Processor

Technical Reference Manual

Kristian Klomsten Skordal
skordal@opencores.org

Project page:
`http://opencores.org/project,potato`
Report bugs and issues on
`http://opencores.org/project,potato,bugtracker`

Updated May 20, 2015

Contents

1 Introduction

The Potato processor is an implementation of the 32-bit integer subset of the RISC-V instruction set v2.0. It is designed around a standard 5-stage pipeline. All instructions execute in 1 cycle, with the exception of load and store instructions when the processor has to wait for external memory.

The processor has been tested on an Artix 7 (xc7a100tcsg324-1) FPGA from Xilinx, on the Nexys 4 board from Digilent. More details about the test design can be found in chapter 2.

1.1 Features

Here is a highlight of the current features of the Potato processor:

- Implements the complete 32-bit integer subset of the RISC-V ISA v2.0.
- Implements the CSR* instructions from the RISC-V supervisor extensions v1.0.
- Supports using the FROMHOST/TOHOST registers for communicating with a host environment, such as a simulator, or peripherals.
- Supports exception handling, with support for 8 individually maskable IRQ inputs.
- Includes a wishbone B4 compatible interface.

1.2 Planned features

Here is a highlight of the future planned features of the Potato processor:

- Caches.
- Branch prediction.

- Hardware multiplication and division support (the RISC-V M extension).
- Compressed instruction support (the RISC-V C extension).
- Supervisor mode support

2 Quick Start Guide

This chapter contains instructions on getting started with the demo/example design that is included in the Potato source distribution. The example design targets the Nexys 4 board available from Digilent[1].

2.1 Setting up the Vivado Project

Start by creating a new project in Vivado. Import all source files from the `src/` directory, which contains all source files required for using the processor. Then import all source files from the `example/` directory, which contains the toplevel setup for the example SoC design, and from the `soc/` directory, which contains various peripherals for the processor.

2.2 Adding IP Modules

The example design requires two additional IP modules. These are not included in the source distribution and must be added separately.

2.2.1 Clock Generator

Add a clock generator using the Clocking Wizard. Name the component "`clock_generator`" and make sure that the checkboxes for "frequency synthesis" and "safe clock startup" are selected.

Add two output clocks with frequencies of 50 MHz and 10 MHz. Rename the corresponding ports to "`system_clk`" and "`timer_clk`". Rename the input clock signal to "`clk`".

The added module should appear in the hierarchy under the toplevel module as "`clkgen`".

[1]See `http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS4`

2.2.2 Instruction memory

Add a block RAM to use for storing the test application using the Block Memory Generator. Choose “Single-port ROM” as memory type and name the module “`instruction_rom`”. Set port A width to 32 bits and the depth to 2048 words. Initialize the block RAM with your application or use one of the provided benchmarks, such as the SHA256 benchmark, which, when built, produces a `.coe` file that can be used for this purpose.

Note that in order to build a benchmark application, you have to install a RISC-V toolchain. See section 4.1 for instructions on how to accomplish this.

2.3 Running an Example Application

Assuming you initialized the instruction memory with the SHA256 benchmark, synthesize and implement the design, generate a bitfile and load it into the FPGA. Using a serial port application, such as `minicom`, watch as the number of hashes per second are printed to the screen and rejoice because it works!

3 Instantiating

The Potato processor can be used either with or without a wishbone interface. Using the wishbone interface allows the processor to communicate with other wishbone-compatible peripherals. However, if no such peripherals are to be used, the processor can, for instance, be connected directly to block RAM memories for full performance without needing to use caches.

3.1 Customizing the Processor Core

The processor can be customized using generics. The following list details the parameters that can be changed:

`PROCESSOR_ID`: Any 32-bit value used as the processor ID. This value can be read back from the hardware thread ID register, HARTID.

`RESET_ADDRESS`: Any 32-bit value used as the address of the first instruction fetched by the processor after it has been reset.

3.2 Instantiating in a Wishbone System

In order to integrate the Potato processor into a wishbone-based system, the module `pp_potato` is used. It provides signals for the wishbone master interface, prefixed with `wb_`, and inputs for interrupts and the HTIF interface.

The specifics of the wishbone interface is listed in table 3.1. To see an example of the processor used in a Wishbone system, see the example design under the `example/` directory.

Wishbone revision	B4
Interface type	Master
Address port width	32 bits
Data port width	32 bits
Data port granularity	8 bits
Maximum operand size	32 bits
Endianess	Little endian
Sequence of data transfer	Not specified

Table 3.1: Wishbone Interface Specifics

3.3 Instantiating in a Standalone System

The processor can also be used without connecting it to the Wishbone bus. An example of this can be seen in the processor testbench, `tb_processor.vhd`.

3.4 Verifying

The processor provides an automatic testing environment for verifying that the processor correctly executes the instructions of the ISA. The tests have been extracted from the official test suite available at `https://github.com/riscv/riscv-tests` and covers most of the available instructions.

Two testbenches are used to execute the test programmes: `tb_processor.vhd`, in which the processor is directly connected to block-RAM-like memories so the processor never stalls to wait for memory operations to finish (see section 3.3 for more details about this kind of setup), and `tb_soc.vhd`, which models a simple system-on-chip with the processor connected to memories through the wishbone interface (see section 3.2 for more information about this kind of setup).

To run the test suites, run `make run-tests` or `make run-soc-tests`.

Make sure that `xelab` and `xsim` is in your `PATH` or the tests will fail.

4 Programming

The processor implements the RISC-V instruction set, and can be programmed with tools such as GCC.

4.1 Building a RISC-V Toolchain

An “official” toolchain is provided by the RISC-V project. In order to install it, clone the “riscv-tools” Git repository from `https://github.com/riscv/riscv-tools` and follow the instructions provided by the README file.

4.2 Control and Status Registers

The supported control and status registers are shown in table 4.1. The registers can be manipulated using the CSR* family of instructions, listed in 5.1.

Name	**ID**	**Description**
HARTID	0x50b	Hardware thread ID
EVEC	0x508	Exception vector address
EPC	0x502	Return address for exceptions
CAUSE	0x509	Exception cause
SUP0	0x500	Support register 0, for operating system use
SUP1	0x501	Support register 1, for operating system use
BADVADDR	0x503	Bad address, used for invalid address exceptions
STATUS	0x50a	Processor status and control register
TOHOST	0x51e	Register for sending data to a host system
FROMHOST	0x51f	Register where data received from a host system is stored
CYCLE	0xc00	Cycle counter, low 32 bits
CYCLEH	0xc80	Cycle counter, high 32 bits
TIME	0xc01	Timer tick counter, low 32 bits
TIMEH	0xc81	Timer tick counter, high 32 bits
INSTRET	0xc02	Retired instruction counter, low 32 bits
INSTRETH	0xc82	Retired instruction counter, high 32 bits

Table 4.1: List of Control and Status Registers

5 Instruction Set

The Potato processor is designed to support the full 32-bit integer subset of the RISC-V instruction set, version 2.0. The ISA documentation is available from `http://riscv.org`.

5.1 Status and Control Register Instructions

In addition to the base ISA, some additional instructions have been imported from the RISC-V supervisor specification[1] version 1.0.

Mnemonic	**Description**
`scall`	System call
`sbreak`	Breakpoint instruction
`sret`	Exception return
`csrrw rd, rs1, CSR`	Writes rs1 into CSR, place sold value in rd
`csrrs rd, rs1, CSR`	Ors rs1 with CSR, places old value in rd
`csrrc rd, rs1, CSR`	Ands the inverse of rs1 with CSR, places old value in rd
`csrrwi rd, imm, CSR`	Writes imm into CSR, places old value in rd
`csrrsi rd, imm, CSR`	Ors CSR with imm, places old value in rd
`csrrci rd, imm, CSR`	Ands the inverse of imm with CSR, places old value in rd

Table 5.1: List of CSR Instructions

[1]The processor is in the process of being upgraded to the new specification.

A Peripherals

The source distribution of the processor contains several peripheral modules that can be used in system-on-chip designs using the Potato processor (or other processors).

This chapter briefly describes each of the modules.

A.1 GPIO

The GPIO module provides a simple GPIO interface for up to 32 general purpose pins. Each pin can be separately configured to work as either an input or an output pin.

Registers are provided to set the direction of each pin. Additional registers provide the ability to read or write the values of the pins.

A.2 Timer

The timer module provides a timer that fires off an interrupt at a specified interval.

A.3 UART

The UART module provies a fixed-baudrate serial port interface. It features separate FIFOs for buffering input and output data, and interrupts for when the module is ready to send or has received data.

A.4 Memory

The memory module is basically a simple block RAM wrapper with support for byte-writes.