OR1K startup code

A small internal program to handle startup of the OR1K family of RISC processors

Brought to You By ORSoC / OpenCores

Legal Notices and Disclaimers

Table of Contents

Chapter 1 Description

On the OR1K family of processors the reset address is by default 0x100. On some implementations this can be changed to an arbitrary address. Since memory address 0x0 and upwards is RAM this invokes a challenge.

Some systems may have non volatile memory in the form of SPI FLASH. Dependant on controller this type of memory might not be accessible to ordinary read and write operations but rather through an SPI controller.

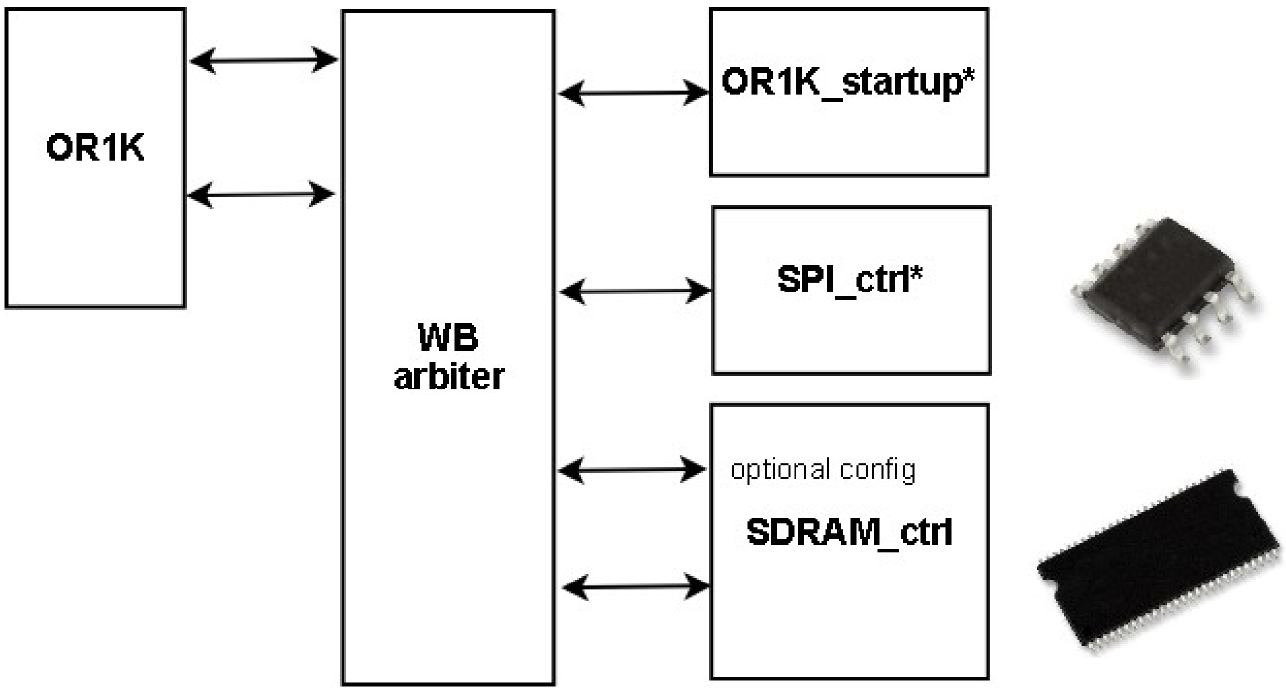
This IP do not take care of the start address, that must be the responsibility of wishbone arbiter or internal OR1K setup. If the system directs the instruction fetch to this module at startup the following will happen

1. required internal registers will be set
2. an optional loop will write configure data to memory addresses defined in SPI FLASH
3. application code and data will be copied from external SPI FLASH to memory location 0x0 and upwards
4. program execution will start at address 0x100

This IP is written in Verilog RTL and can be targeted to the following target technologies:

- Generic target independent design
- ACTEL ProASIC3

Typical setup with IPs included in OR1K_startup marked with *:



Chapter 2 OR1K Startup IP

This IP is wishbone B3 compatible. It is a wishbone slave with 32 bit data bus.

Top level signals

The following signals are present on the top module:

Wishbone signal	Width	Direction
wb_dat_o	[31:0]	Output
wb_dat_i	[31:0]	Input
wb_adr_i	[6:2]	Input
wb_stb_i		Input
wb_cyc_i		Input
wb_ack_o		Output
wb_clk		Input
wb_rst		Input

The Startup IP is functionally an embedded ROM (or pre configured RAM).

Target independent implementation

The target independent design is in file called "OR1K_startup_generic.v".

Implementation is done with flip-flops on output *wb_dat_o*. Memory content is implemented as case statement. Function will be implemented as ordinary logic.

This design can be used for any target technology family.

The base address to the SPI core can be changed with use of define SPI_BASE_MSB. Set the define to the upper 16 bits of the base address.

Example: SPI has base adress 0xB000_0000.

```
+define+SPI_BASE_MSB+B000
```

FPGA	resources
ACTEL PROAsic3	155 slices
ALTERA Cyclone III	64 combinatorial functions, 33 registers

ACTEL ProASIC3 implementation

The ACTEL ProASIC3 family of FPGA has built in 1 kbit FLASH which can be used for the OR1K startup program.

The Verilog file to be used is called "OR1K_startup_ACTEL.v". The design also includes file flash.v which is generated from ACTEL Libero smartgen and can be found in

"\syn\flash\smartgen\flash"

Also included in directory syn is a Libero project to ease future updates of FLASH content. To be able to simulate design the user must compile ACTEL file proasic3.v which included simulation support for internal FLASH.

When configuring ACTEL devices be sure to include file "flash.ufc" to define FLASH content.

FPGA	resources
ACTEL PROAsic3	50 slices, 1kbit FLASH

Chapter 3 Companion modified SPI IP

The SPI controller to be used with the startup program is the "SPI controller core" available from OpenCores.org. This IP can be found on the following URL

http://www.opencores.org/?do=project&who=spi

This controller acts as a wishbone slave. A number of configuration registers is used for the setup of the behavior of the SPI communications. In this case the configuration is known in advanced. It is therefor possible to replace some of these registers with configuration patterns. A modified version of the SPI controller is included in this project to handle this.

The following modifications are done in the SPI controller for this project

Define/parameter	value	comment
Module name	Spi changed to spi_flash	To be able to include original design together with this modified version
Char register set to 32 bit	define	
Reset value of char register	0x03000000	This is the command to initiate burst read at address 0x0
Ctrl register bit LSB replaced with define	undefined	Operation is MSB
Ctrl register bit TX_NEG replaced with define	defined	if defined mosi_pad_o is changed on falling edge of sclk_pad_o
Ctrl register bit RX_NEG replaced with define	undefined	if defined miso_pad_i is changed on falling edge of sclk_pad_o
Ctrl ASS removed		Automatic slave select deselected
SPI clk divider register changed to parameter		This value is wishbone clock frequency dependent and can be changed per instance. SPI clock divider set to length 4
SPI interrupt		function removed
wishbone err_o		function removed

FPGA	resources
ACTEL PROAsic3	393 slices
ALTERA Cyclone III	206 combinatorial functions, 83 registers

Chapter 3 Startup program

The start up programs is a very small application that is supposed to be implemented either as preconfigured RAM, built-in ROM or as combinatorial gates. It relies on a in system SPI controller.

Program flow:

1. Register init
   R0 = 0x0
   R1 = 0x0
   R4 = SPI_BASE
   R5 = SPI_CONTROL_VALUE
2. activate slave select[0]
3. Read first 32 bit word, end of program pointer
   R2 = read()
4. Loop1, peripheral configuration
   inc(R1)
   R8 = read()
   R3 = read()
   (R8) = R3
   branch if not(R8=0x0) Loop1
5. Loop2, application copy
   R3 = read()
   inc(R1)
   (R1) = R3
   branch if not(R1=R2) Loop2
6. inactivate slave select[0]
7. R6 = 0x100
   jump R6

Application program data

Application must contain the following for proper startup

1. address 0x0 must contain last address of application as a 32 bit number, aligned to 32 bit address
   .long end_of_app
2. consecutive addresses starting at 0x4 should contain configuration address + configuration data as two 32 bit words. Must end with configuration address equal to 0x0
   .long config_adr1
   .long config_data1
   .long config_adr2
   .long config_data2
   ...
   .long 0x0
   .long dummy_data

Chapter 4 Optional use as SD FLASH controller

If the SPI controller is defined to have two slave select signals this controller could also be used as a SD FLASH card controller.

In the verilog directory a ready made module instantiation file is included. This file, "OR1K_startup_module_inst.v", includes the following

1. OR1K_startup module instantiation
2. SPI module instantiation
3. input and output multiplexer for SPI signals

In your top level Verilog RTL include the following:

```
`include "OR1K_startup_module_inst.v"
```

Recommended Resources

ORSoC – http://www.orsoc.se

ORSoC is a fabless ASIC design & manufacturing services company, providing RTL to ASIC design services and silicon fabrication service. **ORSoC** are specialists building complex system based on the OpenRISC processor platform.

Open Source IP – http://www.opencores.org

Your number one source for open source IP and other FPGA/ASIC related information.

Appendix A Program code

```
/*          Register usage
            R0 = 0
            R1 = pointer to destination
            R2 = nr of words to be copied
            R3 = data
            R4 = pointer to SPI core
            R5 = spi control word
            R6 = jump adr
            R7 = nr of control words
            r8 = adr pointer                    */

            .equ spi_base,0xb0000000            ;
            .equ spi_ctrl,0x0520                ;
            .equ read_cmd, 0x03000000           ;
            .global start                       ;
            .text

start:      l.movhi r0,0x0                      ;
            l.ori r1,r0,0x0                     ; # R1 = 0x0
            l.movhi r4,hi(spi_base)             ; # R4 = spi base
            l.ori r5,r0,lo(spi_ctrl)            ; # R5 = spi control

            # set slave select active
            l.ori r3,r0,0x1                     ; # R3 = 0x1

            # initiate SPI FLASH READ
            l.jal read                          ;
            l.sw 0x18(r4),r3                    ; # ss[0] active

            l.jal read                          ; # read
            l.sw 0x0(r4),r0                     ; # set TX to zero
            l.or r2,r3,r3                       ; # R2 = nr of words

loop1:      l.jal read                          ; # read adr pointer
            l.addi r1,r1,0x8                    ;
            l.jal read                          ; # read data
            l.or r8,r3,r3                       ; # copy adr pointer to r8
            l.sfeq r8,r0                        ;
            l.bnf loop1                         ;
            l.sw 0x0(r8),r3                     ; # write control data

loop2:      l.jal read                          ;
            l.addi r1,r1,0x4                    ;
            l.sw 0x0(r1),r3                     ; # write to RAM
            l.sfeq r1,r2                        ;
            l.bnf loop2                         ;
            l.ori r6,r0,0x100                   ;
            l.jr r6                             ; # jump to 0x100
            l.sw 0x18(r4),r0                    ; # set slave select inactive

read:       l.sw 0x10(r4),r5                    ; # write spi control
w4busy:     l.lwz r3,0x10(r4)                   ;
            l.sfeqi r3,lo(spi_ctrl)             ;
            l.bf w4busy                         ;
            l.nop                               ;
            l.jr r9                             ;
            l.lwz r3,0x0(r4)                    ; # R3 = RX
```