STORM SoC – System on Chip
by Stephan Nolting

Datasheet and Implementation Guide

WISHBONE COMPATIBLE

Proprietary Notice

The **STORM CORE** Processor System and the **STORM SoC** were created by Stephan Nolting.
Contact: stnolting@googlemail.com, zero_gravity@opencores.org

The most recent versions of them can be found at
STORM Core: http://www.opencores.com/project,storm_core
STORM SoC: http://www.opencores.com/project,storm_soc

Table of content

1. Introduction

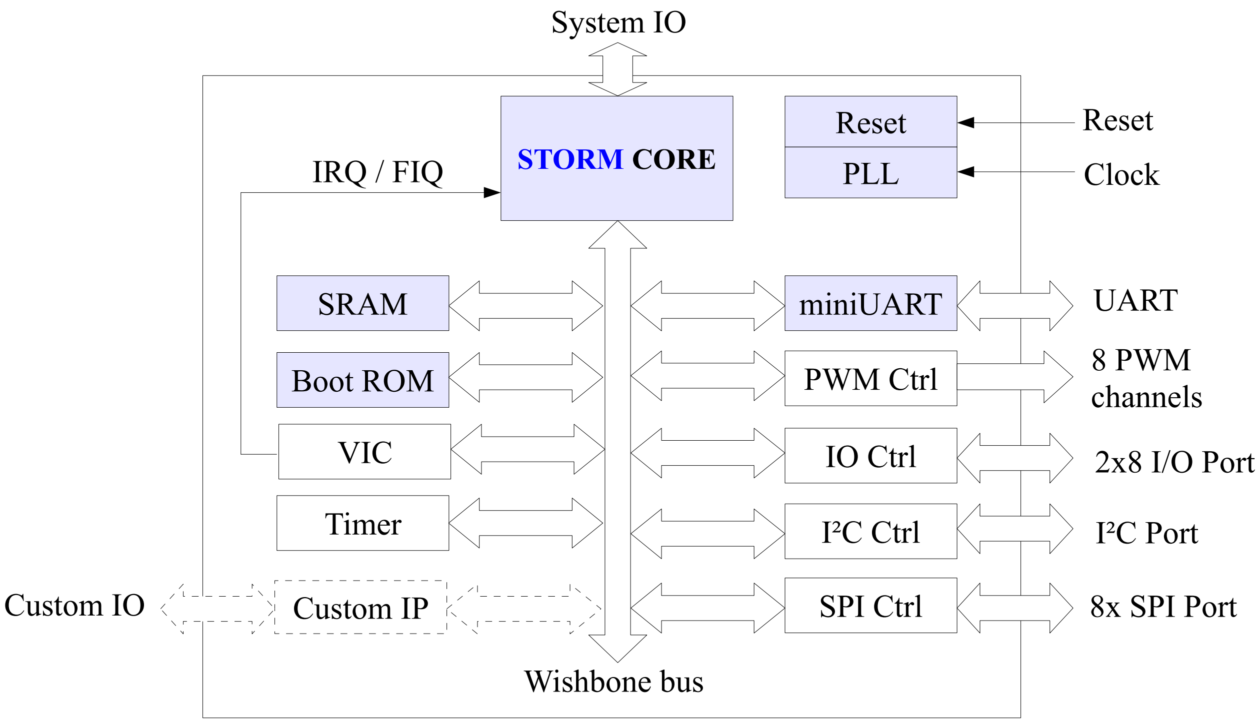
The STORM SoC (System on Chip) is a complete microcontroller system, build around the STORM Core processor. It is completely FPGA and evaluation board – manufacturer independent. Due to it's Wishbone bus system, it can easily be expanded with a large variety of open-source hardware components like memory controller, different communication interfaces and special processing modules.

2. STORM SoC Basic

This project already includes a library of basic system components, which can easily be added or removed from an example system configuration – the STORM_SoC_basic. It is intended to serve as initial platform for a user-specified system designs.

IO driver and WinARM compatible makefiles are included within this project.
A pre-installed console bootloader can be used for easy program downloading and debugging.

Top entity of the basic STORM SoC configuration:
storm_soc\trunk\basic_system\rtl\STORM_SoC_basic.vhd



>> STORM SoC Basic configuration <<

Note: For a minimum system configuration, which is still able to run the pre-installed bootloader, the shaded modules are mandatory.

2.1 Features of the Basic STORM SoC Configuration

- ✔ Based on the STORM Core Processor System (ARM7 native)
- ✔ 1kb D-cache and 1kb I-cache (both are full-associative)
- ✔ 32-bit Wishbone bus system (pipelined)
- ✔ Clock distribution system (PLL)
- ✔ Reset protector
- ✔ WinARM compatible makefiles
- ✔ Pre-defined driver libraries and example programs (C files)
- ✔ Internal 32 kb RAM for program code and data
- ✔ Internal 8 kb ROM with pre-installed bootloader
- ✔ 32-bit system timer
- ✔ Vectorized interrupt controller (LPC compatible)
- ✔ 8 general purpose input pins
- ✔ 8 general purpose output pins
- ✔ Simple mini UART (9600-8-N-1)
- ✔ SPI controller providing 3 ports (3/3/2 chip select lines each)
- ✔ I²C controller (boot from I²C EEPROM supported by processor)
- ✔ 8 independent PWM outputs

2.2 Port Description

This chapter describes the interface of the basic system configuration (top entity).

The type of all signals is std_logic / std_logic_vector. The corresponding signal width is noted in 'bit'.
Signal suffix: I → FPGA input pin/port; O → FPGA output pin/port; IO → FPGA bidirectional pin/port

System interface:

Signal name	Bit	Function
CLK_I	1	System clock input (50MHz suggested), all internal system trigger on the rising edge of this signal, connected to SYSCON_CLK
RST_I	1	System reset input (synchronous, low active), this pin is connected to a reset protection circuit – only activating this pin for at least 1 second will generate a valid internal reset, connected to SYSCON_RST
UART0_RXD_I	1	System console terminal data receiver input, fixed interface properties: 9600-8-N-1, connected to GP_UART_0
UART0_TXD_O	1	System console terminal data transmitter output, fixed interface properties: 9600-8-N-1, connected to GP_UART_0
START_I	1	Start application button (low active!), activating this pin will skip the bootloader console and immediately load and start a boot image from the attached I²C EEPROM (address = 0xA0) mapped to IN(0) of the processor's own system IO port
BOOT_CONFIG_I	4	Power-on boot configuration, “0000” - start bootloader console “0001” - boot from RAM “0010” - boot from I²C EEPROM (always from address 0xA0), mapped to IN(4:1) of the processor's own system IO port
LED_BAR_O	8	System status output (connect to 8xLEDs), mapped to OUT(7:0) of the processors own system IO port

General purpose ports (GP_IO):

Signal name	Bit	Function
GP_INPUT_I	8	General purpose inputs, mapped to IN(7:0) of GP_IO_CONTROLLER_0, IN(31:8) are set to '0'
GP_OUTPUT_O	8	General purpose outputs, mapped to OUT(7:0) of GP_IO_CONTROLLER_0

Inter Integrated-Circuit Bus (I²C):

Signal name	Bit	Function
I2C_SCL_IO	1	I2C_CONTROLLER_0 serial clock input/output
I2C_SDA_IO	1	I2C_CONTROLLER_0 data clock input/output

Serial Peripheral Interface (SPI):

The interface of the SPI controller is split up into three ports allowing to create different bus subsets.
For example when accessing internal chip select 3, SPI port 1 is used and CS(0) of SPI port 1 is active (low).

Signal name	Bit	Port #	Function
SPI_P0_CLK_O	1	Port 0	SPI_CONTROLLER_0 port_0 serial clock output
SPI_P0_MISO_I	1		SPI_CONTROLLER_0 port_0 serial data input, only selected when a CS of this port is active
SPI_P0_MOSI_O	1		SPI_CONTROLLER_0 port_0 serial data output
SPI_P0_CS_O	3		Port_0 chip select lines (low active), mapped to SS(2:0) of SPI_CONTROLLER_0
SPI_P1_CLK_O	1	Port 1	SPI_CONTROLLER_0 port_1 serial clock output
SPI_P1_MISO_I	1		SPI_CONTROLLER_0 port_1 serial data input, only selected when a CS of this port is active
SPI_P1_MOSI_O	1		SPI_CONTROLLER_0 port_1 serial data output
SPI_P1_CS_O	3		Port_1 chip select lines (low active), mapped to SS(5:3) of SPI_CONTROLLER_0
SPI_P2_CLK_O	1	Port 2	SPI_CONTROLLER_0 port_2 serial clock output
SPI_P2_MISO_I	1		SPI_CONTROLLER_0 port_2 serial data input, only selected when a CS of this port is active
SPI_P2_MOSI_O	1		SPI_CONTROLLER_0 port_2 serial data output
SPI_P2_CS_O	2		Port_2 chip select lines (low active), mapped to SS(7:6) of SPI_CONTROLLER_0

Pulse-Width-Modulation (PWM) Port:

Signal name	Bit	Function
PWM0_PORT_O	8	PWM_CONTROLLER_0 pulse-width-modulated output signals

3. System Setup

This chapter explains step-by-step the setup of the STORM System on Chip.
Altera Quartus II © will be used as FPGA design tool.
For other synthesis tools (like Xilinx IDE ©) the basic setup flow is nearly the same.

Terminal v1.9b (copyright by Br@y++) will be used as com port terminal program. Of course, any other terminal program providing a file transfer option can be used as well.

Basic Setup Flow

1) Start Altera Quartus II and create a new project.

2) Configure the device settings corresponding to your target FPGA.

3) Add all core and system module HDL files to the project (a list of all hardware source files can be found in chapter 5.1). Also, add a PLL (configuration described in the PLL chapter) with the Megawizzard tool to the design.

4) Declare the '**STORM_SoC_basic.vhd**' as the design's top entity.

5) Add / remove system components to fit the design to your application.
→ see chapter 3.1

6) Configure the system settings (internal SRAM size, external reset level, clock frequency, ...).
→ see chapter 3.2

7) Start the synthesis of the design.

8) Open the pin assignment editor and assign FPGA pins to the interface ports.

9) Compile your design.

10) Connect your FPGA evaluation board to the computer and download the configuration bit stream.

11) Connect the UART port of the STORM SoC via an RS232 interface to your computer and open the terminal program. Select the correct COM port and set the interface properties (baud rate = 9600, 8 data bits, 1 stop bit, no parity bit).
→ see chapter 3.4

12) Press the configured reset button of the STORM SoC for at least 1 second. After wards, the bootloader console will show up in the terminal window.

13) Press (and send) '1' ("program core RAM with program file"). Then, transmit the application code ('storm_program.bin' file) in byte stream mode.
→ see chapter 3.3

14) The application program will start automatically after the download has finished.

3.1 Adding / Removing Components

You can expand the basic STORM SoC configuration to fit your specific application.
In this chapter I will show you how to add additional components to the design. As an example, the pulse-width-modulation controller will be instantiated.

To remove components from the design, precede with this tutorial – just delete all the new entries and signals, which are created here.

3.1.1 Adding the module to the components list

First of all, you have to make sure the component is already listed within the components list of the STORM SoC top entity (for the case of the basic configuration, the module is already listed). Don't forget to add the module's source file(s) to the project sources as well.

```
-- PWM Controller -------------------------------------------------------------------------
-- -------------------------------------------------------------------------------------------
    component PWM_CTRL
           port (
                  -- Wishbone Bus --
                  WB_CLK_I      : in  STD_LOGIC;                     -- memory master clock
                  WB_RST_I      : in  STD_LOGIC;                     -- high active sync reset
                  WB_CTI_I      : in  STD_LOGIC_VECTOR(02 downto 0); -- cycle identifier
                  WB_TGC_I      : in  STD_LOGIC_VECTOR(06 downto 0); -- cycle tag
                  WB_ADR_I      : in  STD_LOGIC;                     -- address input
                  WB_DATA_I     : in  STD_LOGIC_VECTOR(31 downto 0); -- write data
                  WB_DATA_O     : out STD_LOGIC_VECTOR(31 downto 0); -- read data
                  WB_SEL_I      : in  STD_LOGIC_VECTOR(03 downto 0); -- data quantity
                  WB_WE_I       : in  STD_LOGIC;                     -- write enable
                  WB_STB_I      : in  STD_LOGIC;                     -- valid cycle
                  WB_ACK_O      : out STD_LOGIC;                     -- acknowledge
                  WB_HALT_O     : out STD_LOGIC;                     -- throttle master
                  WB_ERR_O      : out STD_LOGIC;                     -- abnormal termination

                  -- PWM Port --
                  PWM_O         : out STD_LOGIC_VECTOR(07 downto 0)
                 );
    end component;
```

<< PWM Controller component declaration in system component list >>

3.1.2 Adding the interconnection signals, IO ports and module addresses

Declare 5 new signals (type std_logic / std_logic_vector) for the module ↔ system bus interface. Even if most of the bus signals are shared within the system (see interface signals of the STORM Core processor), at least five signals must be unique for each bus-connected module.

Signal name	Bit	Function
PWM_CTRL0_DATA_O	32	Module data output (for read access)
PWM_CTRL0_STB_I	1	Module select input
PWM_CTRL0_ACK_O	1	Module acknowledge output
PWM_CTRL0_HALT_O	1	Module bus halt request
PWM_CTRL0_ERR_O	1	Module bus transaction (abnormal) abort

Note: The 'I' / 'O' suffix refers to the module. For example PWM_CTRL0_DATA_O is a signal driven by the module (PWM controller).

```
-- PWM Controller 0 --
signal PWM_CTRL0_DATA_O   : STD_LOGIC_VECTOR(31 downto 0);
signal PWM_CTRL0_STB_I    : STD_LOGIC;
signal PWM_CTRL0_ACK_O    : STD_LOGIC;
signal PWM_CTRL0_HALT_O   : STD_LOGIC;
signal PWM_CTRL0_ERR_O    : STD_LOGIC;
```

<< Interconnection signals module ↔ bus system declaration >>

Since the component provides IO ports ("real world peripherals" → PWM outputs), you need to define an interface bus within the entity's port list.

```
-- PWM Port 0 --
PWM0_PORT_O   : out   STD_LOGIC_VECTOR(07 downto 0);
```

<< Definition of the module's specific IO interface >>

Furthermore, you need to configure the device address and IO size.

Add two new constants to the address map. The PWM_CTRL0_BASE_C constant declares the base address of the module. PWM_CTRL0_SIZE_C declares the size of the occupied IO space of the module in bytes. The PWM controller provides 2 internal registers (each 32 bit wide), which can be accessed via the bus system. So the needed IO address space is 2*32/8 = 8 byte.

You have to make sure, the base address as well as the occupied IO space is NOT used by any other component. Also, I recommend to use an address within the dedicated IO area.

```
constant PWM_CTRL0_BASE_C  : STD_LOGIC_VECTOR(31 downto 0) := x"FFFF0070";
constant PWM_CTRL0_SIZE_C  : natural := 2*4; -- byte
```

<< Definition of the module's specific IO interface >>

3.1.3 Adding the unique module signals to the Wishbone fabric

The 5 unique module signals from the previous tutorial part must be included in the Wishbone fabric. This fabric consists of five signal terminals – one for each unique signal.

Add a new address-comparator for the "Valid Transfer Signal Terminal":

```
PWM_CTRL0_STB_I <= CORE_WB_STB_O
             when ((CORE_WB_ADR_O >= PWM_CTRL0_BASE_C) and
                   (CORE_WB_ADR_O < Std_logic_Vector(unsigned(PWM_CTRL0_BASE_C)+ PWM_CTRL0_SIZE_C)))
             else '0';
```

<< New entry of STB terminal >>

Insert new entries for the "Read-Back Data Selector Terminal", the "Acknowledge Terminal", the "Abnormal Termination Terminal" and the "Halt Terminal" corresponding to the address-location of your new module (→ canonical):

```
...
I2C0_CTRL_DATA_O    when (I2C0_CTRL_STB_I    = '1') else
PWM_CTRL0_DATA_O    when (PWM_CTRL0_STB_I    = '1') else
VIC_DATA_O          when (VIC_STB_I          = '1') else
...
```

<< Inserting the component's read-data bus into the "Read-Back Data Selector Terminal" >>

```
...
I2C0_CTRL_ACK_O     or
PWM_CTRL0_ACK_O     or
VIC_ACK_O           or
...
```

<< Inserting the component's acknowledge signal into the "Acknowledge Terminal" >>

```
...
I2C0_CTRL_ERR_O     or
PWM_CTRL0_ERR_O     or
VIC_ERR_O           or
...
```

<< Inserting the component's error signal into the "Abnormal Termination Terminal" >>

```
...
I2C0_CTRL_HALT_O    or
PWM_CTRL0_HALT_O    or
VIC_HALT_O          or
...
```

<< Inserting the component's halt signal into the "Halt Terminal" >>

3.1.4 Instantiating the new module

Finally, it is time to instantiate the new component and connect it to the rest of the system.

The PWM controller contains two 32-bit register, which are accessible via the bus system, so it needs a one-bit signal to determine which register is accessed. Since each register is accessible with one specific 32-bit address and both register are 32 bit wide, the address must be on word boundary → connect bit 2 of the system's address bus to the single address input signal.

Even when using components with 8 bit register / data width, you have to map the registers to a complete 32-bit address on word boundary (bis 0 and 1 of the system address bus CORE_WB_ADR_O are always zero). This also implicates the CORE_WB_SEL_O signal (4 bits wide) is always "1111".

```
-- PWM Controller 0 -----------------------------------------------------------------------
-- ----------------------------------------------------------------------------------------------
    PWM_CONTROLLER_0: PWM_CTRL
        port map (
                    -- Wishbone Bus --
                    WB_CLK_I      => MAIN_CLK,          -- memory master clock
                    WB_RST_I      => MAIN_RST,          -- high active sync reset
                    WB_CTI_I      => CORE_WB_CTI_O,     -- cycle identifier
                    WB_TGC_I      => CORE_WB_TGC_O,     -- cycle tag
                    WB_ADR_I      => CORE_WB_ADR_O(2),  -- address input
                    WB_DATA_I     => CORE_WB_DATA_O,    -- write data
                    WB_DATA_O     => PWM_CTRL0_DATA_O,  -- read data
                    WB_SEL_I      => CORE_WB_SEL_O,     -- data quantity
                    WB_WE_I       => CORE_WB_WE_O,      -- write enable
                    WB_STB_I      => PWM_CTRL0_STB_I,   -- valid cycle
                    WB_ACK_O      => PWM_CTRL0_ACK_O,   -- acknowledge
                    WB_HALT_O     => PWM_CTRL0_HALT_O,  -- throttle master
                    WB_ERR_O      => PWM_CTRL0_ERR_O,   -- abnormal termination

                    -- PWM Port --
                    PWM_O         => PWM0_PORT_O
                );
```

<< Instantiation and signal connection for the new PWM controller >>

Even if a new component does not feature a halt request signal or an error signal, bus connection signals should be declared and implemented as well (→ canonical). Assign a '0' to these unconnected signals then.

```
    -- Halt / Error --
    I2C0_CTRL_HALT_O <= '0'; -- no throttle -> full speed
    I2C0_CTRL_ERR_O  <= '0'; -- nothing can go wrong - never ever!
```

<< Signal termination example for unsupported HALT and ERR signals >>

3.1.5 Integrating the new module into software

To use the new system component, it must be declared within the storm_soc_basic.h file.

```
/* PWM Controller 0 */
#define PWM0_BASE       (*(REG32 (0xFFFF0070)))   // base address of module
#define PWM0_SIZE       2*4                       // module's IO size in bytes
#define PWM0_CONF0      (*(REG32 (0xFFFF0070)))   // address of 1st register (CONF0)
#define PWM0_CONF1      (*(REG32 (0xFFFF0074)))   // address of 2nd register (CONF1)
```

<< Register address declaration >>

Now you can access the device within your program:

```
unsigned long temp;

temp = PWM0_CONFIG0;        // load old configuration
temp = temp & 0xFFFFFF00;   // mask → keep configuration for channel 3, 2 and 1
temp = temp | 0x00000008;   // set new configuration for channel 0
PWM0_CONFIG = temp;         // store configuration to controller
```

<< Changing the duty cycle of PWM channel 0 to '8' >>

3.2 Changing Memory Size

If your FPGA does not provide enough dedicated memory elements to create a 32kb SRAM or if you simply do not need this amount of storage capability, you can reduce the size of the internal memory. Of course you can also extend the memory size (for example when using external memory).
Changing the memory size in the top entity's address map (the memory size must be entered in bytes):

```
-- Address Map -------------------------------------------------------------------
-- -------------------------------------------------------------------------------
    constant INT_MEM_BASE_C     : STD_LOGIC_VECTOR(31 downto 0) := x"00000000";
    constant INT_MEM_SIZE_C     : natural := 32*1024; -- byte
    constant BOOT_ROM_BASE_C    : STD_LOGIC_VECTOR(31 downto 0) := x"FFF00000";
    constant BOOT_ROM_SIZE_C    : natural := 8*1024; -- byte
    ...
```

<< Memory size configuration (blue → internal SRAM) >>

Open the STORMcore-ROM.ld linker file in the "build" of your program folder.
Change the RAM length corresponding to the new memory size and set the STACK_SIZE to a value covering your requirements (like ½ memory_size).

```
ENTRY(_boot)
STACK_SIZE = 0x4000;

/* Memory Definitions */
MEMORY
{
  ROM (rx) : ORIGIN = 0xFFF00000, LENGTH = 0x00002000
  RAM (rw) : ORIGIN = 0x00000000, LENGTH = 0x00008000
}
```

<< "STORMcore-RAM.ld" stack and memory size definition >>

! **If you have changed any memory size (RAM or ROM), you have to modify the memory definitions of the bootloader code, too. Afterwards, you need to recompile it and load the new image into the memory initialization area of the BOOT_ROM.vhd component.**

Currently, only a bootloader version using 8*1024 bytes ROM and 32*1024 bytes RAM is impemented. Bootloader images supporting other SRAM sizes will be selectable via the boot ROM initialization string will be implemented in future versions.

3.3 Software Setup

The STORM SoC project features WinARM* compatible makefiles, which allow fast and easy program setup. These files also include a STORM Core specific start-up code, which takes care of a proper system initialization.

*) Download and installation tutorial for WinARM: http://www.winarm.scienceprog.com/comment/2

Include the STORMcore.h (processor internal definitions) and the STORMsoc.h (module address map) within the application program, to get easy access to processor/system functions.

To guarantee a fast and easy program setup, an IO driver file is included. Via this file, programmers can easily use all system modules. Most of the control functions feature a short information text, so no further function explanation should be necessary.

The “io_driver.c” / “io_driver.h” enables easy access to the following hardware components:

- General purpose IO controller 0 (read/write port/pin, toggle pin)
- PWM controller 0 (read/write PWM channel duty cycle)
- GP miniUART 0 (receive/send byte)
- SPI controller 0 (config SPI, read&write SPI, manual (de-)select devide)
- I²C controller 0 (config I²C, read/write to/from I²C device)
- System (access to system coprocessor & CMSR)

Open the system console (execute CMD.exe) and navigate to the software folder of the STORM SoC project folder. Enter and execute “make clean all” to compile your project (main file is “main.c”).

```
...\storm_soc\trunk\basic_system\software\demo_program> make clean all
```

After the assembler has finished, the storm_extractor will create a “storm_program.txt” file for direct memory initialization and also a “storm_program.bin” file, which can be downloaded into the STORM Core's RAM or an attached I²C EEPROM via the pre-installed STORM SoC bootloader.

The “demo_program” folder contain a simple demo program, which blinks LED(0) of the system IO port and echoes any received (UART) character to the terminal program.
See the next chapter to see how you can download the demo program into the STORM SoC's SRAM.

3.4 Using the Bootloader

The boot ROM component of the STORM SoC is pre-loaded with a powerful bootloader software. Connect the serial port of the STORM SoC (RXD and TXD) via an RS232 transceiver to your host PC and launch a terminal program (like Hterm.exe from the tools folder). Configure the terminal program corresponding to the default system configuration (baud rate = 9600, 8 data bits, 1 stop bit, no parity bits). Press the reset button of the STORM SoC. Now, the bootloader menu should appear in the terminal window (see below).

```
+--------------------------------------------------------------+
|    <<< STORM Core Processor System - By Stephan Nolting >>>    |
+--------------------------------------------------------------+
|         Bootloader for STORM SoC  Version: 15.05.2012         |
|               Contact: stnolting@googlemail.com                |
+--------------------------------------------------------------+

 < Welcome to the STORM SoC bootloader console! >
 < Select an operation from the menu below or press >
 < the boot key for immediate application start. >


 0 - boot from core RAM (start application)
 1 - program core RAM via UART_0
 2 - core RAM dump
 3 - boot from I2C EEPROM
 4 - program I2C EEPROM via UART_0
 5 - show content of I2C EEPROM
 a - automatic boot configuration
 h - help
 r - restart system

Select:
```

<< bootloader console output >>

To download a program to the STORM SoC, press (and transmit) '1'. Afterwards, you can send ("*Send File*") the "storm_program.bin" programming file, which was created by the makefile / the storm_extractor, respectively (in byte-stream mode = no handshake, no transmission frames, simple raw data).

You can also download this programming file into an attached I²C EEPROM (like 24FC64 - the device must be accessible with a 16-bit address). After programming, you can configure the STORM SoC via the boot configuration port to boot automatically from the EEPROM after power-up. Press (and transmit) 'a' / 'h' within the bootloader console to get more information.

It is also possible to copy the content of the "storm_program.txt" file to the memory initialization area inside the MEMORY.vhd component. Then it is possible to boot from the internal RAM, directly.

4. System Components

This chapter gives a brief overview of all components, which are already included in the STORM SoC's source files. These components allow you to create a basic system on chip without downloading (and eventually porting) other IP cores.
Of course it is possible to expand the module library with other (Wishbone) compatible components.

Notes

- Most of the components, which are connected to the Wishbone bus, can be byte-accessed, even if this feature is not relevant within the STORM SoC (due to the static 32-bit interface=.
- All modules are Wishbone compatible and use therefore the same signal namings for the bus interface. These bus interface signals are not listed in the component's description, only special (non-Wishbone) signals are mentioned.
- Most components, which where not created by me, feature an additional data sheet. See the component's doc folder.
- IO devices are automatically protected and can only be accessed in privileged modes.
- Due to the fabric-internal address-comparator and -mappings, the register order of the relative memory address map is not always equal to the register order in the system memory map. Use only the addresses / IO locations specified in the system memory map.

4.1 STORM Core Processor

Author: Stephan Nolting
File name of component's top entity: **STORM_TOP.vhd**

The STORM Core Processor is the heart of the STORM SoC. It's sources are not included within the SoC project and must be downloaded separately at: http://www.opencores.com/project,storm_core.

All IO/Memory transactions are controlled by the STORM Core and only by the STORM Core – there is no DMA implemented. It is equipped with separated, full-associative cache units for instructions and data. A Wishbone compatible bus unit connects these units to rest of the system.

The core itself is compatible to ARM's famous RISC controller family (→ ARMv2 Instruction Architecture). The opcodes, the functionality as well as the programmer's model are ARM-native.

! See the STORM Core data sheet in the core's doc folder for more information.

Configuration

Generic	Generic type	Function
I_CACHE_PAGES	natural	Number of pages in I-Cache
I_CACHE_PAGE_SIZE	natural	I-Cache page size (# of 32-bit words)
D_CACHE_PAGES	natural	Number of pages in D-Cache
D_CACHE_PAGE_SIZE	natural	D-Cache page size (# of 32-bit words)
TIME_OUT_VAL	natural	Maximum Wishbone bus cycle length
BOOT_VECTOR	std_logic_vector(31:0)	Boot vector address
IO_UC_BEGIN	std_logic_vector(31:0)	First address of not cache-able IO area
IO_UC_END	std_logic_vector(31:0)	Last address of not cache-able IO are

Interface

Port signal	Signal size	Direction	Function
CORE_CLK_I	1 bit	Input	Core clock signal, triggering on rising edge
RST_I	1 bit	Input	System rest, high-active, sync to rising edge of core clock
IO_PORT_O	16 bit	Output	Direct system output port
IO_PORT_I	16 bit	Input	Direct system input port
WB_ADR_O	32 bit	Output	Wishbone bus address, word-boundary → bits(1:0) = "00"
WB_CTI_O	3 bit	Output	Wishbone bus cycle type "000" → classic cycle "001" → constant address burst "010" → incrementing address burst "111" → burst end
WB_TGD_O	7 bit	Output	Wishbone bus cycle tag WB_TGD_O(6) → '1' for instruction- / '0' for data transfer WB_TGD_O(5) → '1' for IO- / '0' for MEM access WB_TGD_O(4:0) → STORM Core current status mode code
WB_SEL_O	4 bit	Output	Wishbone bus byte select, always "1111"
WB_WE_O	1 bit	Output	Wishbone bus write enable
WB_DATA_O	32 bit	Output	Wishbone bus data output
WB_DATA_I	32 bit	Input	Wishbone bus data input
WB_STB_O	1 bit	Output	Wishbone bus valid transfer
WB_CYC_O	1 bit	Output	Wishbone bus valid cycle
WB_ACK_I	1 bit	Input	Wishbone bus acknowledge signal
WB_ERR_I	1 bit	Input	Wishbone bus abnormal cycle termination
WB_HALT_I	1 bit	Input	Wishbone bus halt
IRQ_I	1 bit	Input	Interrupt request
FIQ_I	1 bit	Input	Fast interrupt request

4.2 Internal SRAM

Author: Stephan Nolting
File name of component's top entity: **MEMORY.vhd**

This memory component is the basic module for the internal data/program memory. Set the OUTPUT_GATE generic "true", if you are using an or-based Wishbone data read-back. Enabling this feature might cause problems for the synthesis tool to map the memory to dedicated memory components.
The memory can be initialized with a program code or data segment when needed. Place the program/data in the "INIT MEMORY IMAGE" labeled signal initialization (this feature should only be used for simulation / debugging).

Interface / Configuration

Generic	Generic type	Function
MEM_SIZE	natural	Memory size in cells (=32 bit entries)
LOG2_MEM_SIZE	natural	Log2 of memory size (= log2(MEM_SIZE))
OUTPUT_GATE	boolean	Use and-gates for data output

Relative address map

Relative address	Area (byte)	R/W	Function
0x00000000 ... MEM_SIZE-4	MEM_SIZE*4	R/W	Free-to-use memory cells, each 4 bytes wide

4.3 Boot ROM

Author: Stephan Nolting
File name of component's top entity: **BOOT_ROM_FILE.vhd**

This memory component is the basic module for the internal boot rom memory. Set the OUTPUT_GATE generic "true", if you are using an or-based Wishbone data read-back. Enabling this feature might cause problems for the synthesis tool to map the memory to dedicated memory components.
The memory can be initialized with a program code or data segment when needed, pre-installed bootloaders for several development boards can be selected via the INIT_IMAGE_ID. Ensure, the STORM Core is booting up from the base address of the boot ROM.

Interface / Configuration

Generic	Generic type	Function
MEM_SIZE	natural	Memory size in cells (=32 bit entries)
LOG2_MEM_SIZE	natural	Log2 of memory size (= log2(MEM_SIZE))
OUTPUT_GATE	boolean	Use and-gates for data output
INIT_IMAGE_ID	string	Name of initialization image

Relative address map

Relative address	R/W	Function
0x00000000 ... MEM_SIZE-4	R	Free-to-use memory cells, each 4 bytes wide

4.4 IO Controller

Author: Stephan Nolting
File name of component's top entity: **GP_IO_CTRL.vhd**

The IO controller provides a simple IO port. 32 bits are used as outputs, another 32 are used as inputs.
It can be used for directly controlling FPGA pins or for internal system control functions.
The IRQ output will go high for one clock cycle when the status of the inputs has changed.

Do not leave any input pin floating. This might cause unintended IRQ generation.

Interface / Configuration

Special signal	Signal type	Function
GP_IO_O	std_logic_vector(31:0)	Parallel output port
GP_IO_I	std_logic_vector(31:0)	Parallel input
IO_IRQ_O	std_logic	Input status change interrupt

Relative address map

Relative address	R/W	Function
0x00000000	R/W	Output port register
0x00000004	R/W	Input port register

4.5 Seven Segment Controller

Author: Stephan Nolting
File name of component's top entity: **SEVEN_SEG_CTRL.vhd**

Up to four (d = 0..3) high or low-active seven segment display can be controlled with this module. To display hexadecimal numbers, the corresponding value can be written to the DATA register (offset = 0). To display other symbols, you can write to the CTRL register (offset = 4) to directly control the display segments. Hex-coded DATA will be decoded and the decoded value is automatically written to the CTRL register.

Display control lines (d is the led-display index):

```
A-segment(d) <= HEX_O(d+0)
B-segment(d) <= HEX_O(d+1)
C-segment(d) <= HEX_O(d+2)
D-segment(d) <= HEX_O(d+3)
E-segment(d) <= HEX_O(d+4)
F-segment(d) <= HEX_O(d+5)
G-segment(d) <= HEX_O(d+6)
```

Display segments:

```
 AAAAA
F     B
F     B
F     B
 GGGGG
E     C
E     C
E     C
 DDDDD
```

```
display(d) <= DATA_REGISTER(d*4+3 downto d*4+0)
display(d) <= CTRL_REGISTER(d*7+6 downto d*7+0)
display(d) <= HEX_O(d*7+6 downto d*7+0)
```

Interface / Configuration

Generic	Generic type	Function
HIGH_ACTIVE_OUTPUT	boolean	Connected LEDs are high-active

Special signal	Signal type	Function
HEX_O	std_logic_vector(27:0)	Control signals for 4 seven segment displays

Relative address map

Relative address	R/W	Function
0x00000000	R/W	Hex DATA register
0x00000004	R/W	Segment CTRL register

4.6 Timer

Author: Stephan Nolting
File name of component's top entity: **TIMER.vhd**

This timer can be used for any timing application. Whenever the counter register value reaches the threshold value, an interrupt (when enabled) is generated. An automatic reset can e applied if bit 1 of the control register is set. A prescaler value different from 0 will scale the frequency of the counter increment. The interrupt output signal will become high for one clock cycle, when the counter register value reached the threshold value (and interrupt enable bit is set).

Interface / Configuration

Special signal	Signal type	Function
INT_O	std_logic	Compare interrupt, one clock cycle high

Relative address map

Relative address	R/W	Function
0x00000000	R/W	Counter register
0x00000004	R/W	Threshold value register
0x00000008	R/W	Configuration register
0x0000000C	R/W	Scratch register

Configuration Register

Bit(s)	R/W	Function
31..16	R/W	Prescaler value
15..3	R/W	*unused*
2	R/W	Interrupt enable
1	R/W	Auto reset after threshold reached
0	R/W	Timer enable

4.7 Vector Interrupt Controller

Author: Stephan Nolting
File name of component's top entity: **VIC.vhd**

The vectorized interrupt controller is mostly compatible to the one used e.g. in LPC ARM controller. Up to 16 interrupt sources can be configured using vectorized interrupt service routine (ISR) addresses. Another 16 interrupt request lines can be mapped to one interrupt service routine. See the data sheet of an LPC ARM controller for more information about the VIC.

Relative address map

Relative address	R/W	Function
`0x00000000`	R	Masked IRQ request status
`0x00000004`	R	Masked FIQ request status
`0x00000008`	R	Unmasked interrupt requests
`0x0000000C`	R/W	Interrupt lines type select, '0' = IRQ, '1' = FIQ
`0x00000010`	R/W	Interrupt request lines enable
`0x00000014`	W	Clear interrupt request line enable
`0x00000018`	W	Trigger interrupt request line by software
`0x0000001C`	W	Clear software interrupt request line
`0x00000020`	R/W	Bit 0: Protected mode enable → access only in privileged modes possible
`0x00000030`	R/W	Interrupt service routine address / interrupt acknowledge
`0x00000034`	R/W	Interrupt service routine address for unvectorized interupts
`0x00000038`	R/W	High level / rising edge ('0') or low level / falling edge trigger
`0x0000003C`	R/W	Level triggered ('0') or edge triggered ('1') interrupt
`0x00000040` ... `0x0000007C`	R/W	Interrupt service routine address for corresponding interrupt channel
`0x00000080` ... `0x000000BC`	R/W	Source select (bits 4:0) and enable (bit 5) for corresponding interrupt channel

4.8 Mini UART

Author: Philippe Carton (opencores)
Modified by: Stephan Nolting
File name of component's top entity: **MINI_UART.vhd**

This is a simple UART, created by Philippe Carton. It is capable of transmitting, receiving data via the RS232 port. The system was modified to support 32-bit Wishbone bus access.

Interface / Configuration

Special signal	Signal type	Function
IntTx_O	std_logic	Waiting for byte interrupt
IntRx_O	std_logic	Byte received interrupt
BR_CLK_I	std_logic	Clock used for baud generator
TxD_PAD_O	std_logic	Transmitter output
RxD_PAD_I	std_logic	Receiver input

Relative address map

Relative address	R/W	Function
0x00000000	R/W	UART data register Bits [7:0]: Received / transmitted character
0x00000004	R/W	UART status register Bit 0: Ready to send when '1' Bit 1: Byte received when '1' Bit 31..16: BAUD rate divisor = clk_freq/(4*baud_rate);

Configuration Register

Bit(s)	R/W	Function
0	R	Ready to send when '1'
1	R	Byte received when '1', auto-cleared when reading the data register
31..16	R/W	Baud rate divisor = clk_freq/(4*baud_rate);

4.9 SPI Controller

Author: Simon Srot (opencores)
File name of component's top entity: **spi_top.v**

This SPI controller provides a high-speed SPI port with 8 low-active slave select signals.

Relative address map

Relative address	R/W	Function
`0x00000000`	`R/W`	Data receive / transmit register 0
`0x00000004`	`R/W`	Data receive / transmit register 1
`0x00000008`	`R/W`	Data receive / transmit register 2
`0x0000000C`	`R/W`	Data receive / transmit register 3
`0x00000010`	`R/W`	Control and status register
`0x00000014`	`R/W`	Clock divider register
`0x00000018`	`R/W`	Slave select register

! See the data sheet in the component's doc folder for more information.

4.10 I²C Controller

Author: Richard Herveille (opencores)
File name of component's top entity: **i2c_master_top.vhd**

The I2C controller implements an interface for on-board communication via I²C.
This device was originally created for an eight bit bus system, so only the lowest 8 bits of the data bus are relevant, all other bits are read as zero. Writing data on bits 31 downto 8 does not perform any operation.

Relative address map

Relative address	R/W	Function
0x00000000	R/W	Clock divider register, low byte
0x00000004	R/W	Clock divider register, high byte
0x00000008	R/W	Control register
0x0000000C	R/W	Receive / transmit data register
0x00000010	R/W	Command / status register

! **See the data sheet in the component's doc folder for more information.**

4.11 PS-2 Keyboard Controller

Author: Daniel Quinter (opencores)
File name of component's top entity: **ps2_wb.vhd**

Via this controller, a ps-2 compatible keyboard can be connected to the STORM SoC.

Relative address map

Relative address	R/W	Function
`0x00000000`	`R/W`	Receive / transmit data register
`0x00000004`	`R/W`	Status / control register

4.12 External Memory Controller

>> Currently not implemented <<

**No external memory controller is
currently supported by the STORM SoC.
Maybe you can implement one ;)**

4.13 Reset Protector

Author: Stephan Nolting
File name of component's top entity: **RST_PROTECT.vhd**

The reset protector is not connected to the Wishbone bus system. It is responsible for generating a valid system reset from an external reset request. Only when a valid external reset is applied to the `EXT_RST_I` input for at least `TRIGGER_VAL` clock cycles, the `SYS_RST_O` pin is taken high for `TRIGGER_VAL`/10000 cycles.

Interface / Configuration

Generic	Generic type	Function
TRIGGER_VAL	natural	Trigger value in clock ticks
LOW_ACT_RST	booelan	EXT_RST_I is low active when set to “true”

Special signal	Signal type	Function
MAIN_CLK_I	std_logic	System clock
EXT_RST_I	std_logic	External reset request
SYS_RST_O	std_logic	System reset, high active

4.14 System PLL

Author: Altera Quartus II - Megawizzard
File name of component's top entity: **SYSTEM_PLL.vhd**

This component was generated by the Altera Megawizzard tool. The phase locked loop is used to create the internal and external clock signals, as well as a system reset (only active during warm-up of PLL).

Interface

Special signal	Signal type	Function
inclk0	std_logic	External clock input
c0	std_logic	System clock
c1	std_logic	Memory clock = system clock / 2
c2	std_logic	Memory clock, -3ns phase shifted
locked	std_logic	Clock outputs are stable when '1'

4.15 PWM Controller

Author: Stephan Nolting
File name of component's top entity: **PWM_CTRL.vhd**

This controller provides 8 output ports, which can be configured with a specific on/off pulse-width.
The controller can be used to add 'analog' outputs to the design.

The operating frequency of a PWM channel is WB_CLK/64.
The duty cycle of a single port can be set by an 8-bit value (duty_cycle = value/255).
Setting the duty_cycle bits of a port to 0xFF (duty_cycle = 1) will permanently activate the port (high).
Setting the duty_cycle bits of a port to 0x00 (duty_cycle = 0) will permanently deactivate the port (low).

Interface

Special signal	Signal type	Function
`PWM_O`	`std_logic_vector(7:0)`	8 independent PWM channels

Relative address map

Relative address	R/W	Function
`0x00000000`	`R/W`	PWM configuration register 0 (for channel 0,1,2,3)
`0x00000004`	`R/W`	PWM configuration register 1 (for channel 4,5,6,7)

PWM configuration register 0/1

Bits	R/W	Function
`31..24`	`R/W`	Channel 0/4 duty_cylce
`23..16`	`R/W`	Channel 1/5 duty_cylce
`15..08`	`R/W`	Channel 2/6 duty_cylce
`07..00`	`R/W`	Channel 3/7 duty_cylce

5. Source Files

5.1 Hardware Source Files

All hardware sources can be found in the component's rtl directory.
The top entity of the corresponding component is listed first and is highlighted.

Note: The source files of the STORM Core processor itself are not included in the STORM SoC project. They must be downloaded separately at http://www.opencores.com/project,storm_core

STORM Core Processor:
- storm_core/trunk/rtl/**STORM_TOP.vhd**
- storm_core/trunk/rtl/BUS_UNIT.vhd
- storm_core/trunk/rtl/CACHE.vhd
- storm_core/trunk/rtl/CORE_PKG.vhd
- storm_core/trunk/rtl/CORE.vhd
- storm_core/trunk/rtl/OPCODE_DECODER.vhd
- storm_core/trunk/rtl/FLOW_CTRL.vhd
- storm_core/trunk/rtl/MC_SYS.vhd
- storm_core/trunk/rtl/REG_FILE.vhd
- storm_core/trunk/rtl/OPERAND_UNIT.vhd
- storm_core/trunk/rtl/MS_UNIT.vhd
- storm_core/trunk/rtl/MULTIPLY_UNIT.vhd
- storm_core/trunk/rtl/BARREL_SHIFTER.vhd
- storm_core/trunk/rtl/ALU.vhd
- storm_core/trunk/rtl/LOAD_STORE_UNIT.vhd
- storm_core/trunk/rtl/WB_UNIT.vhd

STORM SoC Basic configuration:
- storm_soc/trunk/basic_system/**STORM_SoC_basic.vhd**

Boot ROM:
- storm_soc/trunk/components/boot_rom/rtl/**BOOT_ROM_FILE.vhd**

I²C controller:
- storm_soc/trunk/components/i2c_controller/rtl/vhdl/**i2c_master_top.vhd**
- storm_soc/trunk/components/i2c_controller/rtl/vhdl/i2c_master_byte_ctrl.vhd
- storm_soc/trunk/components/i2c_controller/rtl/vhdl/i2c_master_bit_ctrl.vhd

IO controller:
- storm_soc/trunk/components/io_controller/rtl/**GP_IO_CTRL.vhd**

miniUART:
- storm_soc/trunk/components/miniuart/rtl/vhdl/**MINI_UART.vhd**
- storm_soc/trunk/components/miniuart/rtl/vhdl/rxunit.vhd
- storm_soc/trunk/components/miniuart/rtl/vhdl/txunit.vhd
- storm_soc/trunk/components/miniuart/rtl/vhdl/utils.vhd

PS-2 controller:
- storm_soc/trunk/components/ps2core/rtl/vhdl/**ps2_wb.vhd**
- storm_soc/trunk/components/ps2core/rtl/vhdl/ps2.vhd

Reset protector:
- storm_soc/trunk/components/reset_protector/rtl/**RST_PROTECT.vhd**

Seven segment controller:
- storm_soc/trunk/components/seven_segment_controller/rtl/**SEVEN_SEG_CTRL.vhd**

SPI controller:
- storm_soc/trunk/components/spi_controller/rtl/verilog/**spi_top.v**
- storm_soc/trunk/components/spi_controller/rtl/verilog/spi_clgen.v
- storm_soc/trunk/components/spi_controller/rtl/verilog/spi_defines.v
- storm_soc/trunk/components/spi_controller/rtl/verilog/spi_shift.v
- storm_soc/trunk/components/spi_controller/rtl/verilog/timescale.v

SPI controller:
- storm_soc/trunk/components/sram_memory/rtl/**MEMORY.vhd**

System timer:
- storm_soc/trunk/components/timer/rtl/**TIMER.vhd**

Vectorized interrupt controller:
- storm_soc/trunk/components/vector_interrupt_controller/rtl/**VIC.vhd**

Pulse-Width-Modulation controller:
- storm_soc/trunk/components/pwm_controller/rtl/**PWM_CTRL.vhd**

5.2 Software Source Files

All software sources can be found in the "basic_system/software/lib" directory.

Processor / SoC definitions:
- storm_soc/trunk/basic_system/software/lib/**storm_core.h**
- storm_soc/trunk/basic_system/software/lib/**storm_soc_basic.h**

Basic IO driver:
- storm_soc/trunk/basic_system/software/lib/**io_driver.c**
- storm_soc/trunk/basic_system/software/lib/**io_driver.h**

UART control functions:
- storm_soc/trunk/basic_system/software/lib/**uart.c**
- storm_soc/trunk/basic_system/software/lib/**uart.h**

Several auxiliary functions:
- storm_soc/trunk/basic_system/software/lib/**utilities.c**
- storm_soc/trunk/basic_system/software/lib/**utilities.h**

6. System Address Map

Memory Address Map (→ cache-access only)

Address (hex)	Name	R/W	Module	Register
0x00000000 ... 0x00008000	IRAM_BASE + offset	R/W	32 kb internal SRAM	32-bit memory cell
0xFFF00000 ... 0xFFF00800	ROM_BASE + offset	R	8 kb internal boot ROM	32-bit memory cell

IO Address Map (→ dedicated IO-access only)

Address (hex)	Name	R/W	Module	Register
0xFFFF0000	GPIO0_OUT	R/W	General purpose IO controller 0	32 bit output port
0xFFFF0004	GPIO0_IN	R		32 bit input port
0xFFFF0018	UART0_DATA	R/W	miniUART 0	RX/TX data register
0xFFFF001C	UART0_SREG	R/W		Status register
0xFFFF0020	STME0_CNT	R/W	System timer 0	Counter register
0xFFFF0024	STME0_VAL	R/W		Threshold value
0xFFFF0028	STME0_CONF	R/W		Configuration register
0xFFFF002C	STME0_SCRT	R/W		Scratch register
0xFFFF0030	SPI0_CONF	R/W	SPI controller 0	Configuration register
0xFFFF0034	SPI0_PRSC	R/W		Prescaler value
0xFFFF0038	SPI0_SCSR	R/W		Slave select register
0xFFFF0040	SPI0_DAT0	R/W		FIFO data register 0
0xFFFF0044	SPI0_DAT1	R/W		FIFO data register 0
0xFFFF0048	SPI0_DAT2	R/W		FIFO data register 0
0xFFFF004C	SPI0_DAT3	R/W		FIFO data register 0
0xFFFF0050	I2C0_CMD/STAT	R/W	I²C controller 0	Command / status register
0xFFFF0060	I2C0_PRLO	R/W		Prescaler, low byte
0xFFFF0064	I2C0_PFHI	R/W		Prescaler, high byte
0xFFFF0068	I2C0_CTRL	R/W		Control register
0xFFFF006C	I2C0_DATA	R/W		RX/TX data register

Address (hex)	Name	R/W	Module	Register
0xFFFF0070	PWM0_CONFIG_0	R/W	PWM Controller 0	PWM channel 0,1,2,3 configuration
0xFFFF0074	PWM0_CONFIG_1	R/W		PWM channel 4,5,6,7 configuration
<< add additional modules here >>				
0xFFFFF000	VICIRQStatus	R	Vector interrupt controller	IRQ status (masked)
0xFFFFF004	VICFIQStatus	R		FIQ status (masked)
0xFFFFF008	VICRawIntr	R		Unmasked interrupt request status
0xFFFFF00C	VICIntSelect	R/W		Interrupt type, '1' FIQ, '0' IRQ
0xFFFFF010	VICIntEnable	R/W		INT request lines enable
0xFFFFF014	VICIntEnClear	W		Clear INT request line enable bit
0xFFFFF018	VICSoftInt	W		Trigger INT line by software
0xFFFFF01C	VICSoftintClear	W		Clear SW INT request enable bit
0xFFFFF020	VICProtection	R/W		Protected mode (only priv. acc.)
0xFFFFF030	VICVectAddr	R/W		ISR address / INT acknowledge
0xFFFFF034	VICDefVectAddr	R/W		ISR address for non-vectorized INTs
0xFFFFF038	VICTrigLevel	R/W		Hi/Lo / rising/falling edge detect
0xFFFFF03C	VICTrigMode	R/W		Level/edge INT detector
0xFFFFF040	VICVectAddr0	R/W		ISR address for VEC_INT 0
0xFFFFF044	VICVectAddr1	R/W		ISR address for VEC_INT 1
0xFFFFF048	VICVectAddr2	R/W		ISR address for VEC_INT 2
0xFFFFF04C	VICVectAddr3	R/W		ISR address for VEC_INT 3
0xFFFFF050	VICVectAddr4	R/W		ISR address for VEC_INT 4
0xFFFFF054	VICVectAddr5	R/W		ISR address for VEC_INT 5
0xFFFFF058	VICVectAddr6	R/W		ISR address for VEC_INT 6
0xFFFFF05C	VICVectAddr7	R/W		ISR address for VEC_INT 7
0xFFFFF060	VICVectAddr8	R/W		ISR address for VEC_INT 8
0xFFFFF064	VICVectAddr9	R/W		ISR address for VEC_INT 9
0xFFFFF068	VICVectAddr10	R/W		ISR address for VEC_INT 10
0xFFFFF06C	VICVectAddr11	R/W		ISR address for VEC_INT 11
0xFFFFF070	VICVectAddr12	R/W		ISR address for VEC_INT 12
0xFFFFF074	VICVectAddr13	R/W		ISR address for VEC_INT 13
0xFFFFF078	VICVectAddr14	R/W		ISR address for VEC_INT 14
0xFFFFF07C	VICVectAddr15	R/W		ISR address for VEC_INT 15
0xFFFFF080	VICVectCntl0	R/W		Source select / Enable VEC_INT 0

Address (hex)	Name	R/W	Module	Register
0xFFFFF084	VICVectCntl1	R/W	Vector interrupt controller	Source select / Enable VEC_INT 1
0xFFFFF088	VICVectCntl2	R/W		Source select / Enable VEC_INT 2
0xFFFFF08C	VICVectCntl3	R/W		Source select / Enable VEC_INT 3
0xFFFFF090	VICVectCntl4	R/W		Source select / Enable VEC_INT 4
0xFFFFF094	VICVectCntl5	R/W		Source select / Enable VEC_INT 5
0xFFFFF098	VICVectCntl6	R/W		Source select / Enable VEC_INT 6
0xFFFFF09C	VICVectCntl7	R/W		Source select / Enable VEC_INT 7
0xFFFFF0A0	VICVectCntl8	R/W		Source select / Enable VEC_INT 8
0xFFFFF0A4	VICVectCntl9	R/W		Source select / Enable VEC_INT 9
0xFFFFF0A8	VICVectCntl10	R/W		Source select / Enable VEC_INT 10
0xFFFFF0AC	VICVectCntl11	R/W		Source select / Enable VEC_INT 11
0xFFFFF0B0	VICVectCntl12	R/W		Source select / Enable VEC_INT 12
0xFFFFF0B4	VICVectCntl13	R/W		Source select / Enable VEC_INT 13
0xFFFFF0B8	VICVectCntl14	R/W		Source select / Enable VEC_INT 14
0xFFFFF0BC	VICVectCntl15	R/W		Source select / Enable VEC_INT 15

7. Interrupt Channels

The vector interrupt controller (VIC) of the STORM SoC supports up to 16 vectorized and another 16 un-vectorized interrupt sources. Only vectorized are used for the basic system configuration.

Channel	Device / Function
0	System timer 0 threshold reached
1	GP IO controller 0 input state change
2	miniUART 0 TX done
3	miniUART 0 RX done
4	SPI controller 0 IRQ
5	I²C controller 0 IRQ
6..31	Unused - enabled