PSS Open Source Project

Technical Reference Manual

(ver. 0.9)

April, 2016

Alexander Antonov

153287@niuitmo.ru

Table of contents

1. Introduction

1.1 Overview

PSS (Programmable Supervisor for Systems-on-chip) is a soft IP core that targets to provide the basic means for conducting service operations independently from the main processing logic within system-on-chip (SoC) design. The system offers data exchange capabilities, directed via UART interface (host interface), optional central processing unit and a set of tcl commands to control the system from the host computer.

The motivation for the project is to provide simple, but yet functional starting point in system-on-chip or system-on-FPGA design with data transfer and onboard processing capabilities, but when the solution should not be resource-hungry to make design cycles as short as possible (primarily if you use FPGA) or leave maximum resources for custom data processing units. On modern machines, PSS alone typically takes minutes to progress through the whole compilation flow. PSS can be utilized for system initialization, debug, or programmed (either statically or dynamically) to support complex host interface transactions, that may include integrity verification, decompression, etc. Besides being used as a core unit of an initial SoC design prototype or a controller for accelerator-intensive designs, it can be used as auxiliary monitoring and debug processor in addition to the main and more powerful application processor.

At the moment, the core is not intended to run complex software stacks (i.e. operating system). Generally, the system targets the gap between the designs where the CPU core is not needed and those that require significant CPU horsepower.

Current implementation of PSS uses FreeBSD-licensed ZPU core as a central processing unit. PSS itself is licensed under FreeBSD as well.

1.2 Block diagram

The core provides some basic facilities to run C programs and interact with the SoC units using expansion bus unit with the simple RAM-like interface. Also, the project includes the special unit that utilizes UART interface to initiate transactions to the core. The block diagram of PSS system is shown in Fig. 1.

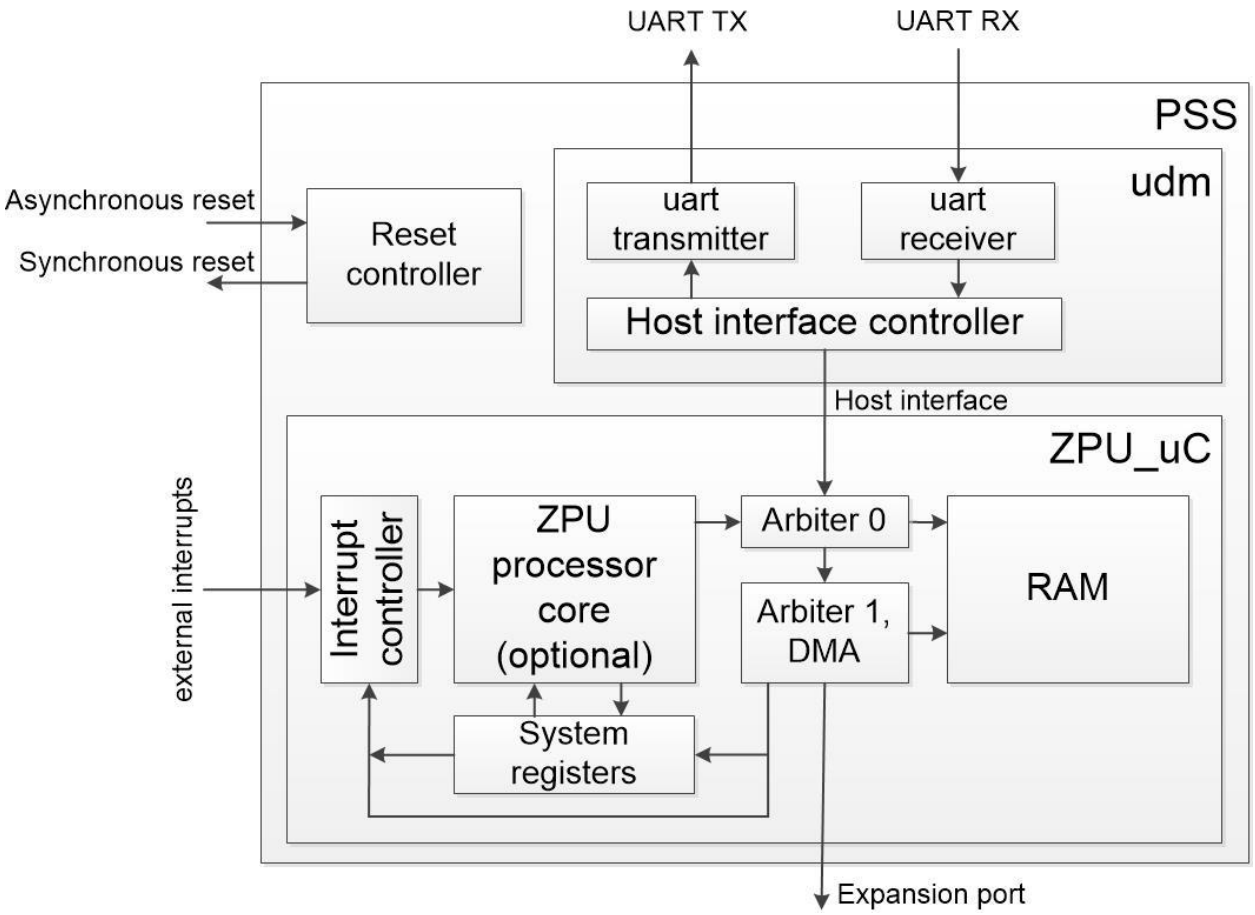


Fig. 1. PSS block diagram

Reset controller converts external asynchronous reset to internal synchronous reset that is used by the units of PSS.

UART debug module (udm) processes commands from external environment via UART interface. It uses host interface to connect to the main processing unit. See Section 2.8 for more details.

ZPU_uC is a module that contains central processing unit (see Section 2.4), RAM unit, system registers that allow to control and monitor the PSS operation (see Section 2.3), interrupt controller (see Section 2.5) and DMA (see Section 2.6).

RAM unit serves as a data buffer for udm-CPU and udm-DMA communication and as a main memory unit for CPU.

2. Programming view

2.1 Introduction

This chapter describes the system facilities that are visible from the programmer's viewpoint.

2.2 Address map

Address space of PSS system is divided between embedded RAM memory (eRAM), system registers and expansion port. Address map is shown in Table 1

Table 1. Address map overview

Address range			Unit
From	To	Size	
0x 0000 0000	0x 0003 FFFF	1 GB	eRAM
0x 0004 0000	0x 0004 FFFF	64 KB	System registers
0x 0005 0000	0x 7FFF FFFF	~ 1 GB	*Reserved*
0x 8000 0000	0x FFFF FFFF	2 GB	Expansion port

eRAM unit is exclusive for PSS system. The amount of memory is configurable, making the actual main memory address space unique for certain configuration. However, it cannot exceed 1 GB. Processor stack is automatically assigned to the greatest addresses of eRAM and grows down to the lower addresses. The actual stack size is not restricted in hardware, which may cause data corruption. However, PSS has a trap that can trigger interrupts once the certain address has been requested (see Section 2.7). The stack border can be programmed into the trap address register, and stack overflow situations can be processed in interrupt service routine.

For description of system registers addresses, please refer to Section 4.1.

Expansion port address space is directly mapped onto expansion bus. However, since expansion port space is restricted by 2 GB value, address MSB has been made separately accessible. So, the whole 4 GB external address space can be potentially accessible by both CPU core and host agent. CPU and host have separate external address MSB bits in system registers unit. See Section 2.3 for more details.

2.3 System control registers

The core includes some basic facilities that will be essential for majority of projects: interrupt controller, DMA and host control interface.

Addresses of system control registers are summarized in Table 2.

Table 2. System control registers overview

Address	Register	R/W	State on reset	Functional description
0x 0004 0000	CPU_CONTROL	R/W	[0] – 1 [1] – 0 [31] – depends on configuration	[0] – reset [1] – cpu in break state [31] – CPU present (read)
0x 0004 0004	CPU_PC	R	0x 0000 0000	CPU program counter
0x 0004 0008	ExBUS_CPU_A31	R/W	[0] - 1	address MSB for CPU external transactions
0x 0004 000C	ExBUS_DBG_A31	R/W	[0] - 1	address MSB for host external transactions
0x 0004 0010	INTC_CONTROL	R/W	0x 0000 0000	interrupt controller state
0x 0004 0014	INTC_MASK	R/W	0x 0000 0000	[7:0] – interrupt mask [31:8] – *reserved*
0x 0004 0018	INTC_REQ	R/W	0x 0000 0000	number of active interrupt request
0x 0004 001C	MEM_SIZE_KB	R	*memory size*	embedded memory size in KB
0x 0004 0020	DMA_CONTROL	R/W	0x 0000 0000	read [1:0] - DMA status: 00 – ready; 01 – ready, error occurred; 10 – DMA read in progress; 11 – DMA store in progress write [1] – DMA command: 0 - DMA load, 1 - DMA store
0x 0004 0024	DMA_SOURCEADDR	R/W	0x 0000 0000	DMA source address
0x 0004 0028	DMA_DESTADDR	R/W	0x 0000 0000	DMA destination address
0x 0004 002C	DMA_SIZE	R/W	0x 0000 0000	DMA transfer size (in bytes)
0x 0004 0030	SGI_REQ	W	*undefined*	0 - SGI request
0x 0004 0034 - *reserved*				
0x 0004 0038	BUS_ERROR_ADDR	R	0x 0000 0000	invalid address being requested
0x 0004 003C	BUS_ERROR_PC	R	0x 0000 0000	PC value when invalid request occurred
0x 0004 0040	TRAP_CONTROL	R/W	0x 0000 0000	[0] – trap enable; [31:1] – *reserved*
0x 0004 0044	TRAP_ADDR	R/W	0x 0000 0000	trap address
0x 0004 0048 - 0x 0004 FFFF - *reserved*				

IMPORTANT NOTICE: Current implementation of PSS DMA supports only 4-byte aligned addresses and transfer sizes with the factor of 4. Any other requests cause undefined behavior.

2.4 Processor core

Current implementation of PSS utilizes the processor core and the corresponding C toolchain from ZPU open source project. For the detailed description of ZPU core, please refer to:
`http://opencores.org/project,zpu`

2.5 Interrupts

PSS system provides four external and four internal interrupts. External interrupts are triggered using interrupt inputs. Internal interrupts have a fixed function and are sourced from the internal units of PSS.

Interrupts have a fixed priority: lower number means higher priority.

Interrupts in PSS are summarized in Table 3.

Table 3. Interrupts overview

Number	Name	Functional description
0	BUS_ERROR	Incorrect address requested from CPU. The actual address is stored in BUS_ERROR_ADDR register, PC value is stored in BUS_ERROR_PC register.
1	TRAP	Trap fired.
2	SGI	SGI fired.
3	DMAI	DMA has finished a transaction.
4-7	EXTI0-EXTI3	External interrupt fired.

2.6 DMA

PSS contains DMA unit that allows the data packets to be transferred from external units to main memory and vice versa almost at the clock speed. At the moment DMA does NOT support the main-memory-to-main-memory and external-units-to-external-units transactions. However, the latter can be achieved by allocating the DMA buffer in memory and transferring data firstly from source to main memory and then from main memory to destination.

DMA uses the separate port to the main memory unit that allows it to operate independently from the processor. For I/O-intensive applications, DMA transfers of the next data package can be initiated prior to having the previous package processed.

DMA is directed by the system control registers (see Tab. 2).

2.7 Trap

Traps are interrupts that are triggered once CPU has accessed the predefined address. Current implementation of PSS has a single trap. Trap can be programmed to trigger by any address within the whole 4 GB address space.

DMA transfers do not cause the trap triggering.

To set the trap, the following actions are required:

1) set the triggering address TRAP_ADDR register;
2) enable the trap by asserting 1 in TRAP_CONTROL[0].

The trap can be useful either for debug purposes or stack border monitoring (to prevent the stack from damaging other main memory content).

2.8 UART debug module (udm)

UART debug module (udm) serves as a host interface controller, directed via UART. Udm has full access to PSS system memory, auxiliary system facilities and expansion bus. Therefore, visibility of both internal and external resources for host port is generally the same as for the processor core. Udm can reset and

start the processor, reprogram it, use DMA for expansion port transfers, communicate with the processor using shared RAM and SGI.

Udm commands are summarized in Table 4.

Table 4. Udm commands overview

Command	Name	Functional description
0x00	PSS_CHECK	Check udm accessibility (udm will return 0x55)
0x80	PSS_HRESET	Reset PSS using the dedicated pin.
0x81 <4-byte address> <4-byte length> <data words>	PSS_WR_INC	Write data with address autoincrement.
0x81 <4-byte address> <4-byte length>	PSS_RD_INC	Read data with address autoincrement.
0x82 <4-byte address> <4-byte length> <data words>	PSS_WR_NOINC	Write data without address autoincrement.
0x83 <4-byte address> <4-byte length>	PSS_RD_NOINC	Read data without address autoincrement.

Udm uses the special synchronization byte (0x55) to signal the beginning of the transaction. Another special byte (0x5a) byte is used as escape character (including the cases when the escape character should be escaped itself).

Udm has baud rate autodetection feature. It uses synchronization byte to detect the current baud rate of UART channel. If you want to change baud rate, reset the device and reconfigure the serial channel on the host.

Udm does not have internal buffering and flow control features – it operates synchronously to host port interface. Therefore, it is not recommended to initiate intensive write transfers directly to expansion port address space due to possible data loss situations. Instead, buffer the data in embedded RAM and use DMA to exchange the data between embedded RAM and expansion port.

3. System integration

3.1 Introduction

This chapter describes how to integrate PSS into your SoC design and configure it correctly.

3.2 Hardware interfaces

The following describes the hardware interfaces that have to be connected to your design or I/O pins of your FPGA board.

Clocking and resets

PSS resides in a single clock domain, and he clocking has to be attached to `clk_i` port. Since PSS adapts to the UART baud rate automatically (see Section 2.8), the user does not have to care about the precise ratio between clock speed and UART baud rate. However, the frequency being used should be sufficient to manage UART transfers. It is recommended to use the clock frequency which is at least an order of magnitude greater than the expected UART baud rates.

PSS uses asynchronous reset input (`arst_i`), which goes to embedded reset controller. Reset controller generates a new reset signal that goes to internal units of PSS. Reset controller asserts reset asynchronously, but deasserts it synchronously several clock periods after it has been deasserted on the input port. The synchronous reset is forwarded to `srst_o` output port, so that it can also be used by the other units outside PSS.

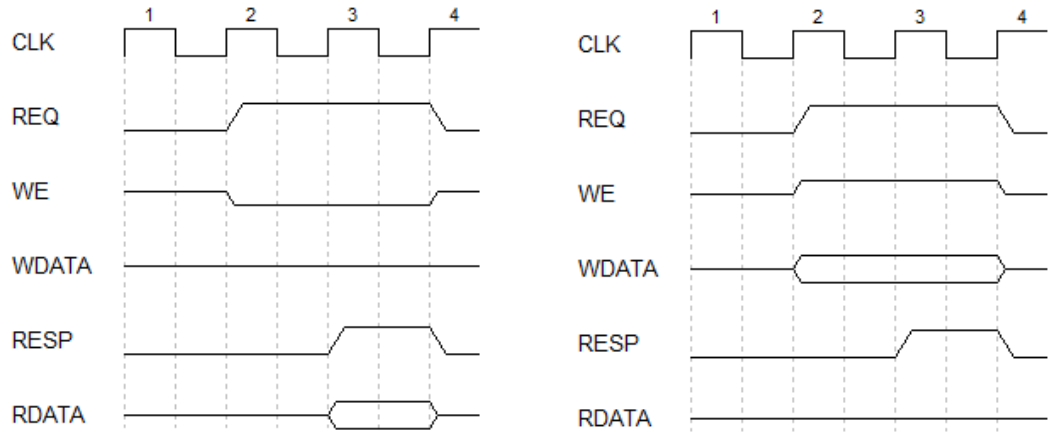
UART interface

UART interface comprises receiver input port (`rx_i`) and transmitter output port (`tx_o`). Receiver input is buffered inside PSS before processing. No special considerations should be taken except locking them on corresponding FPGA pins.

Host port interface and CPU bus

Host port interface and CPU bus are located inside PSS. However, understanding these interfaces can be useful for simulation, debug, and in case you are going to use only specific units of PSS. If you treat PSS as a black-box, you can skip this section.

Host and CPU interfaces are identical. Waveforms for read and write transactions are shown in Fig. 2.



a b

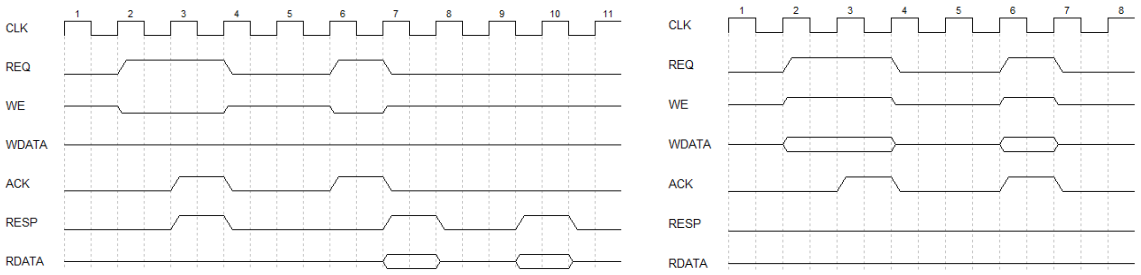
Fig. 2. Read (a) and write (b) transactions for host and CPU interfaces

The interface in synchronous to clock signal. Master unit holds `REQ` signal throughout the entire transaction. `WE` signal identifies if the transaction is read or write. The transaction finishes when the slave device asserts `RESP` signal. `WDATA` is the data that needs to be written, `RDATA` is data that is read from the slave device. Waveforms for read and write transactions are shown in Fig. 3.

Expansion port

Expansion port is used for communication between PSS and other units within the remaining parts of SoC design.

Organization of expansion port differs from host port and CPU interfaces. Read responses are split from requests to utilize pipeline capabilities of modern interconnects. However, strong ordering should be preserved.



a b

Fig. 3. Read (a) and write (b) transactions for host and CPU interfaces

The interface in synchronous to clock signal. Master device holds `REQ` signal until the slave device asserts `ACK`. `WE` signal identifies if the transaction is read or write. If the transaction is write, `WDATA` is the data that needs to be written and the transaction finishes once `ACK` signal is asserted. If the transaction is read, master device waits for `RESP` signals that identifies that the data is ready. `RDATA` is data that is read from the slave device.

Interrupt inputs

External interrupts are triggered by the dedicated input ports (`INT_i`). External interrupts are edge triggered. Interrupt controller contains edge detection and metastability protection logic, therefore, the source of interrupts can reside in separate clock domain. However, interrupt controller does NOT contain debouncing logic, so, in case, for instance, you are going to source the interrupts from switches on your FPGA board, consider using a debouncer.

3.3 RTL configuration

PSS RTL core has to be properly configured depending on your functional requirements. The parameters are summarized in Table 4.

Table 4. PSS parameters overview

Name	Default value	Functional description
`CPU_PRESENT`	1	Presence of CPU in PSS system: 0 – CPU not present; 1 – CPU present.
`CPU_RESET_DEFAULT`	1	CPU reset condition after power-up: 0 – reset deasserted; 1 – reset asserted.
`A31_DEFAULT`	1	A31 bit definitions for CPU and udm units after power-up.
`MEM_SIZE_KB`	1	Size of embedded RAM unit in kilobytes.
`MEM_DATA`	-	File name for RAM initialization.

4. Host-side software

PSS provides the script that defines the set of commands for interaction with the hardware. The script is written in tcl and allows execution from generally any tcl environment, including, for instance, tcl console in Xilinx ISE software. Dedicated tcl interpreters are also allowed.

To load the procedures, type in the tcl console:

`source <path_to_PSS>/SW/terminal/pss.tcl`

The available commands are summarized in Table 5.

Table 5. Host commands overview

Command	Description
`pss_connect com_num baud_rate` [or `pss_con com_num baud_rate`]	connect to serial interface (e.g. *com4*)
`pss_disconnect` [or `pss_discon`]	disconnect from the serial interface
`pss_check`	check response from udm unit
`pss_cc`	connect to serial interface (e.g. *com4*) and check response from udm unit
`pss_sendword dataword`	send 4-byte data word with escape characters
`pss_wr address list_of_datawords`	write 4-byte datawords to address
`pss_rd address count`	read 4-byte datawords from address (total number of bytes is specified by *count)*
`pss_wrfile_le address filename`	write binary file content to address using little-endian
`pss_wrfile_be address filename`	write binary file content to address using big-endian
`pss_reset`	assert reset to CPU
`pss_start`	deassert reset from CPU
`pss_restart`	restart CPU (assert and deassert reset)
`pss_loadbin filename`	assert reset to CPU, load binary image and deassert reset
`pss_dma_wr address list_of_datawords`	write 4-byte datawords to address using DMA
`pss_sgi`	call SGI

IMPORTANT NOTICE: Current implementation of PSS supports only 4-byte aligned addresses and transfer sizes with the factor of 4. Any other requests cause undefined behavior.

5. Demo project description

The project has been tested on Digilent Nexys-2 Spartan-3AN FPGA board. PSS unit is wrapped by pss_soc_top unit that connects PSS to the register, mapped onto address `0x8A000000`. Read request to this address returns the values selected onto switches, while write requests drive the LEDs. The file pss_soc_top is wrapped by either board-specific HDL file (located in project directory) or HDL testbench (*pss/tb/PSS_SoC_tb.v*). Some of the host commands from Section 4 are duplicated in HDL testbench.

Also, the project provides a set of programs for embedded PSS processor that can be loaded to PSS unit to test some of its features. The programs are located in *pss/SW/onboard*.

To test the programs, take the following steps:

1) compile and load the firmware to FPGA board;
2) connect instrumentation PC to FPGA board using UART cable;
3) type "`source <path_to_PSS>/SW/terminal/pss.tcl`" to load PSS tcl procedures;
4) type "`pss_cc <com number> <baud rate>`" to connect to PSS and check the communication channel;
5) type "`pss_loadbin <name of PSS software binary image>`" to load the binary image to PSS.

The following describes the programs available in PSS distribution.

Heartbeat

This program iteratively increments the values on LEDs with delays to make it visible.

SWLED

This program iteratively reads the values of the switches and writes this value to LEDs.

SWnLED

This program iteratively reads the values of the switches and writes inverted value to LEDs.

Interrupts

This program outputs the 0x81 value to LEDs and puts the processor into infinite loop. SGI and pushbutton external interrupt cause the processor to increment this value.

Trap

This program sets the trap to monitor stack overflow and implements recursive function that causes the stack to overflow. Interrupt service routine outputs the 0xAA value to LEDs and stops the processor by putting it into infinite loop.