AXI4 Lite Shift Register Controller

Stefan Kull, Roy Seitz, Marco Zollinger

December 2, 2016

1 Overview

This Document describes functionality and usage of the AXI4 Lite 74xx595 Shift Register Controller. It is supplied as a complete, ready to use IP block.

The controller allows easy access to 74xx595 compliant shift registers. It contains an internal state machine to generate all needed signals, listed in table 1.

The length of the Shift register is a generic and can be up to 32 bits, i.e. up to 4 shift registers in dasy chain. *If the last shift register in daisy chain is not fully used, the unused output pins must be left floating, as their output is not defined.*

The data clock is derived from the AXI bus clock s_axi_aclk, which is divided by D_DIVIDE. See section 2 for detailed information.

Table 1: Output signals of shift register controller

SH output signal	Reset / Idle value	Description
rstn	1	Low active shift register master reset signal
clk	1	Shift register data clock Data is shifted in on rising edge
str	1	Shift register strobe signal Data is load into output register on rising edge
data	0	Shift register serial data in
oen	1	Optional low active output enable If this signal is not used, it is set to constant 0

2 Parameter description

The IP core can be configured at compile time by several parameters, listed in table 2.

Table 2: Parameters for the AXI 74xx595 controller

Parameter	Default	Range	Type	Description
C_SH_DATA_WIDTH	8	1...32	integer	Length of the Shift register
C_USE_OE_N	true	{true, false}	boolean	Use output enable signal
C_CLOCK_DECIMATION	100		integer	Decimation factor for data clock
C_MSB_FIRST	true	{true, false}	boolean	Shift Data out MSB first

3 Register description

Table 3 describes the available registers and their functionality. These registers can be accessed directly by their address or more conveniently by the software driver described in section 4.

Table 3: Register Space

Name	Address offset	Mode	Description
Data	0x0	R/W	Contains LSB aligned data to write to the shift register
Start/Ready	0x4	R/W	Write '1' start programming of shift register Read returns '1' if controller is ready to accept new data
Status	0x8	R/W	Bit 0: Enable controller Bit 1: Low active output enable (if oen is used)

4 Driver

The IP core comes along with a driver to allow easy access by software. There are two categories of functions, high and low level. The usage of high level functions is recommended.

4.1 High level functions

A typical use case with high level functions is as follows:

- call AXI_SH_595_init(baseaddr, mask) to initialize the controller.
  This sets the content of the status register to mask.
  Typically: AXI_SH_595_init(AXI_SH_595_BASEADDR, 0x1);
- Programm shift register by calling AXI_SH_595_programm_sh(baseaddr, data)
  This writes data to the data register and starts the programming sequence.
- If consecutive calls of AXI_SH_595_programm_sh() are used, make sure the controller is ready to accept new commands by calling AXI_SH_595_ready(baseaddr).
  This function returns 1 if the controller is ready and 0 if not.

4.2 Low level functions

The core can also be used by addressing the registers directly. For this purpose two low-level functions are available. These functions are implemented as macros and should only be used if really necessary. The definitions from table 4 can be used to easily access these Registers.

- AXI_SH_595_mWriteReg(BaseAddress, RegOffset, Data)
  Writes data to the specified Register.
- AXI_SH_595_mReadReg(BaseAddress, RegOffset)
  Read the content of the specified register.

4.3 Defined constants

Table 4: Software defined constants

Name	Value
AXI_SH_595_DATA_OFFSET	0
AXI_SH_595_START_READY_OFFSET	4
AXI_SH_595_STATUS_OFFSET	8

5 Test and verification

The core can easily be simulated by using the AXI Trafic generator IP from Xilinx. Therefore no additional simulation files are available. Software verification can be done by writing and reading data to the data register, using low level function.

6 Compatibility and Licence

The core is tested under Vivado version 2016.2 and on Artix7 and Zynq7 FPGA. Since the core does not use any hardware specific resources, it should run on every other FPGA as well. However, this is not tested and may require changes to the core hdl or driver files. The core is supplied under no licence or copyright but is the intellectual property of the authors.