Parallel to Serial Bridge

Functional Specification

July 17th, 2017

1 Introduction **3**

2 I/O Ports and Parameters **3**

2.1 Ports2.1 Ports 3

3 Operation **4**

3.1 Clock and Reset 4

3.2 Input protocol 4

3.3 Output protocol 5

4 CRC Implementation **5**

1 Introduction

A parallel-to-serial block(P2S) converts one or more parallel inputs into a serial data output, transmitted one bit at a time. For data synchronization, P2S uses one clock signal.

All the input data must be read at the same time from the parallel interface and buffered until serial transmission ends. The parallel input data is used to calculate the “end pattern” which is a cyclic redundancy check. Although the output is one bit wide, the parallel data is ordered according to parallel to serial conversion rules.

On the serial interface a starting pattern will be transmitted first to identify the beginning of the packet.

2 I/O Ports and Parameters

This section describes the ports and parameters of P2S block.

2.1 Ports2.1 Ports

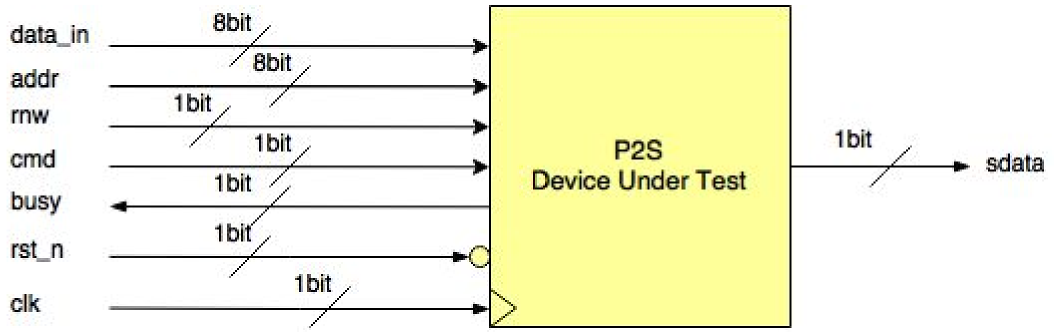

Figure 1. P2S - Device Under Test

In the table below you may find a description of each of the ports.

Port Name	Width (bits)	Direction	Reset value	Description
data_in	8	in	-	The value on this port is sampled by P2s when *cmd* is ’1’ and *rnw* is ‘0’
addr	8	in	-	The value on this port is sampled by P2s when *cmd* is ’1’.
rnw	1	in	-	The value on this port is sampled by P2s when *cmd* is ’1’. It indicates a read or write command.
cmd	1	in	-	P2S samples this port every cycle as long as *busy* is not asserted(i.e. is ‘0’). This port triggers the serialization of

				data_in, *addr* and *rnw* values.
busy	1	out	0	P2S indicates through this port if it can accept a command. As long as *busy* is asserted it will not accept another command (i.e. ignores *cmd* value).
sdata	1	out	0	The data sampled on input parallel interface is transmitted serially on this port.
rst_n	1	in	-	Reset signal active on '0'.
clk	1	in	-	Clock signal of 100MHz.

Table 1. Interface description

3 Operation

3.1 Clock and Reset

P2S uses positive edge of the *clk* to sample the input data, advance the internal finite state machine and drive the output data.

Reset has a negative logic: when its value is 0 it will reset all the outputs and internal logic. If the reset is activated while *busy* is enabled, the transmission will be stopped, but some data might have already been transmitted.

3.2 Input protocol

All the input ports are sampled on the positive *clk* edge, only if *cmd* is active. The value of *data_in* will be sampled only if there is a write operation(i.e. *rnw* = '0'); for a read command constant ***0x5A*** is used instead of *data_in*.

The *busy* signal is asserted if P2S can not accept a new command. P2S supports buffering of one command until the current one ends. This allows the device that controls P2S (e.g. CPU) to send data over the serial link at maximum speed. The value of *cmd* is ignored as long as the *busy* signal is active: the CPU must assert *cmd* until 1 clock cycle after *busy* is released to make sure the command was scheduled for serialization. Figure 2 shows the protocol for read and write commands.

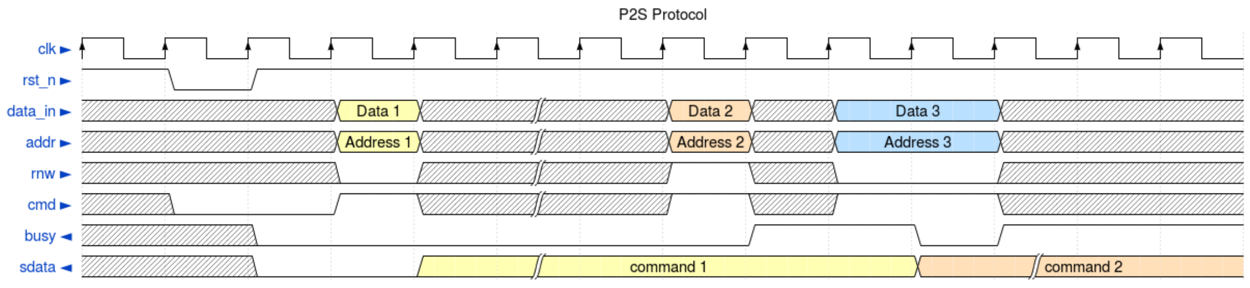

Figure 2. Write command followed by a Read command

All the input data, the “start pattern” and CRC are retained in one of the two shift registers (i.e. a ping-pong buffer as shown in Figure 3). P2S can save one command in a shift register while it serializes the data in the other shift registers.

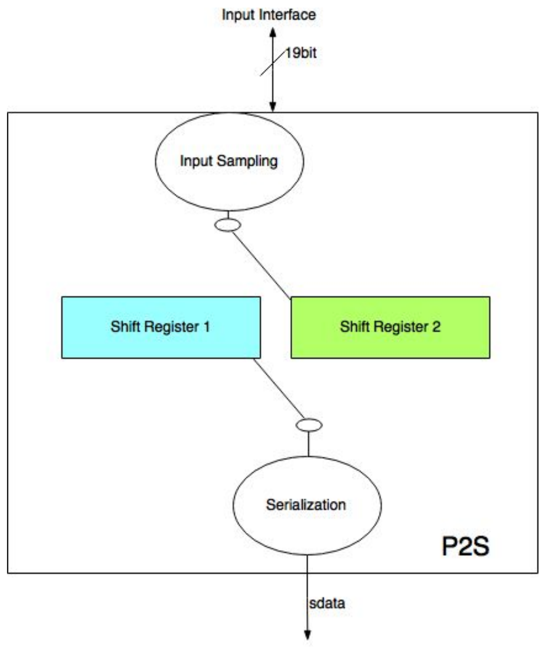

Figure 3. Ping-pong buffer

3.3 Output protocol

The data in the buffer is transmitted sequentially, from left to right, as shown in the image below:

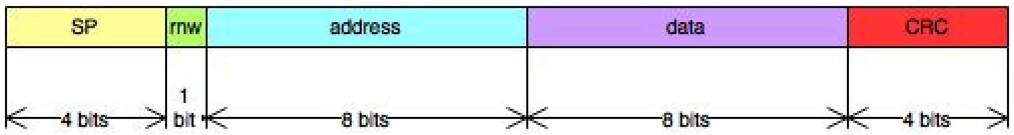

Figure 4. Serial Data Order

The transmission of serial data starts in the cycle following the command OR as soon as the current serialization finished if the command was buffered. Once the last bit of CRC was sent out the current buffer is released and can take in another command.

The “start pattern” indicates the start of transmission on the serial line(i.e. *sdata* signal). The CRC is calculated using polynomial is $1+x+x^4$ for the 17-bits of user data (start pattern is not included).

4 CRC Implementation

Below is a possible implementation of the CRC function for polynomial x^4 + x^1 + 1 and 17-bit data width. The *buf_data* signal holds the command data and *initial_crc* is the seed value for the CRC.

initial_crc = 4'hF;

crc[0] = buf_data[15] ^ buf_data[11] ^ buf_data[10] ^ buf_data[9] ^ buf_data[8] ^
buf_data[6] ^ buf_data[4] ^ buf_data[3] ^ buf_data[0] ^ initial_crc[2];
crc[1] = buf_data[16] ^ buf_data[15] ^ buf_data[12] ^ buf_data[8] ^ buf_data[7] ^
buf_data[6] ^ buf_data[5] ^ buf_data[3] ^ buf_data[1] ^ buf_data[0] ^ initial_crc[2] ^
initial_crc[3];
crc[2] = buf_data[16] ^ buf_data[13] ^ buf_data[9] ^ buf_data[8] ^ buf_data[7] ^ buf_data[6]
^ buf_data[4] ^ buf_data[2] ^ buf_data[1] ^ initial_crc[0] ^ initial_crc[3];
crc[3] = buf_data[14] ^ buf_data[10] ^ buf_data[9] ^ buf_data[8] ^ buf_data[7] ^ buf_data[5]
^ buf_data[3] ^ buf_data[2] ^ initial_crc[1];