VBCP to Wishbone bridge

October 29, 2013

Theodor-Adrian Stana (CERN/BE-CO-HT)

Revision history

Date	Version	Change
26-06-2013	0.01	First draft
14-08-2013	0.02	Second draft
22-10-2013	0.03	Added *Access commands* section, updated document according to changes in protocol
29-10-2013	0.04	Changed PDF link colors

Contents

List of Figures

List of Tables

List of Abbreviations

FSM	Finite-State Machine
VBCP	Inter-Integrated Circuit (bus)
SysMon	ELMA crate System Monitor board
VME	VERSAmodule Eurocard

1 Introduction

This document describes the *vbcp_wb* module, a VME Board Control Protocol (VBCP) to Wishbone bridge HDL core for VME64x crates from ELMA. These crates offer the possibility of accessing boards in VME slots via either VME, or VBCP. Boards not using the VME lines on a slot can implement the *vbcp_wb* module on an FPGA; implements an VBCP slave and translates VBCP accesses into Wishbone [1] accesses to a Wishbone slave device.

A typical system where the *vbcp_wb* module is employed is shown in Figure 1. ELMA VME crates contain a SysMon (system monitor) board [2], that is mainly used for monitoring VME voltages and controlling the fans of the VME crate. The SysMon can be connected to via either a serial connection or Telnet. Then, sending specific commands (see Section 3) via one of the two are translated by the SysMon into VBCP accesses following the protocol described in Section 4.

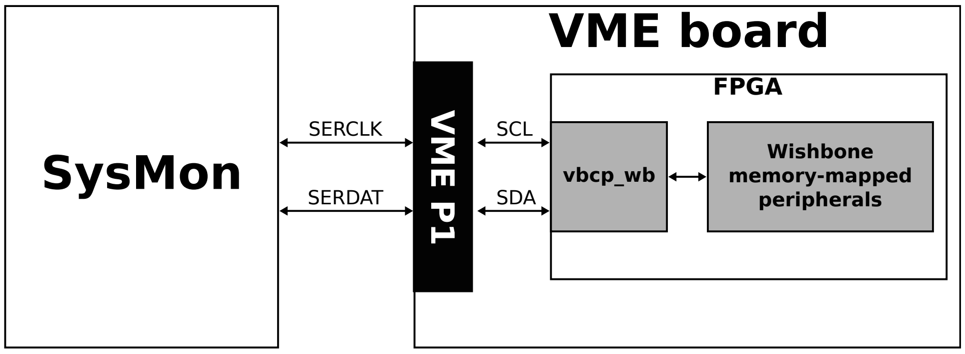

Figure 1: Typical system for the *vbcp_wb* module

2 Instantiation

The ports of the *vbcp_wb* module are shown in Table 1. The I^{2}C signals should be connected to tri-state ports, as shown in Figure 2; Wishbone slaves should be connected to the Wishbone master interface ports, prefixed with *wbm*.

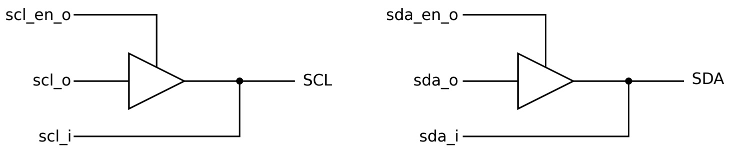

Figure 2: VBCP port external connections

Table 1: Ports of *vbcp_wb* module

Port	Size	Description
clk_i	1	Clock input
rst_n_i	1	Active-low reset input
sda_en_o	1	SDA line output tri-state enable
sda_i	1	SDA line input
sda_o	1	SDA line output
scl_en_o	1	SCL line tri-state enable
scl_i	1	SCL line input
scl_o	1	SCL line output
i2c_addr_i	7	VBCP slave address on ELMA VBCP bus
tip_o	1	Transfer In Progress '1' – I^{2}C address sent by SysMon matches that of the VBCP slave '0' – after transfer has completed and VBCP slave is idle
err_o	1	Error bit, high for one *clk_i* cycle when the Wishbone address the SysMon tries to access is invalid
wbm_stb_o	1	Wishbone data strobe output
wbm_cyc_o	1	Wishbone valid cycle output
wbm_sel_o	4	Wishbone byte select output
wbm_we_o	1	Wishbone write enable output
wbm_dat_i	32	Wishbone data input (to master)
wbm_dat_o	32	Wishbone data output (from master)
wbm_adr_o	32	Wishbone address output
wbm_ack_i	1	Wishbone acknowledge signal input
wbm_rty_i	1	Wishbone retry signal input
wbm_err_i	1	Wishbone error signal input

3 Testing the *vbcp_wb* module

After proper synthesis and download to the FPGA, a Telnet or serial connection should be made to the SysMon board. Commands can then be sent to the boards via the SysMon. The two commands relevant for this basic test are *readreg* and *writereg*. These and other commands relevant for accessing board registers are outlined in Section 4.2.

The example below shows how to connect to an ELMA crate at IP address 1.2.3.4, obtaining the value of a register at address 0x10 in a board in VME slot 2, writing the hex value 0x1234 to the same register and reading it back to check for proper modification.

```
$ telnet 1.2.3.4
Trying 1.2.3.4...
Connected to 1.2.3.4.
Escape character is '^]'.
login:user
password:**********
%>readreg 2 10
 Read Data: 00ABCDEF
%>writereg 2 10 1234
 Done!
%>readreg 2 10
 Read Data: 00001234
```

4 The VME Board Control Protocol

4.1 Protocol details

The VME backplane provides two serial lines (*SERCLK* and *SERDAT*) on the P1 connector. These lines can be used to access boards placed in a VME slots to control them, in cases where the VME interface is not implemented.

The VME Board Control Protocol (VBCP) [3] has been defined for such purposes. Using I^2C as a low-level protocol, the bytes of a register can be read from or written to a VME board.

Figure 3 shows a write operation from the SysMon to a VME board. The process starts with the control byte, containing the board's I^2C slave address and the read/write bit cleared, indicating an I^2C write. After the slave's ACK, the following two bytes send the 12-bit register address in little-endian order (most significant byte first). After the address has been acknowledged, the following four I^2C transfers are used to transmit the 32-bit data to be written to the board register. Data transmission occurs in big-endian order (least significant byte first).

A read transfer (Figure 4) from a VME board is similar to the write transfer. The differences lie in the retransmission of the control byte after the register address, this time with the read/write bit set, to indicate an I^2C read. Following the ACK from the slave, the transfer direction changes and the SysMon will read the four data bytes sent by the VME board. As with the write transfer, the data bytes are sent by the VME board in big-endian order.

4.2 Access commands

In order to send data to a VME board using VBCP, a user connects to the SysMon via Telnet and sends commands which the SysMon translates into

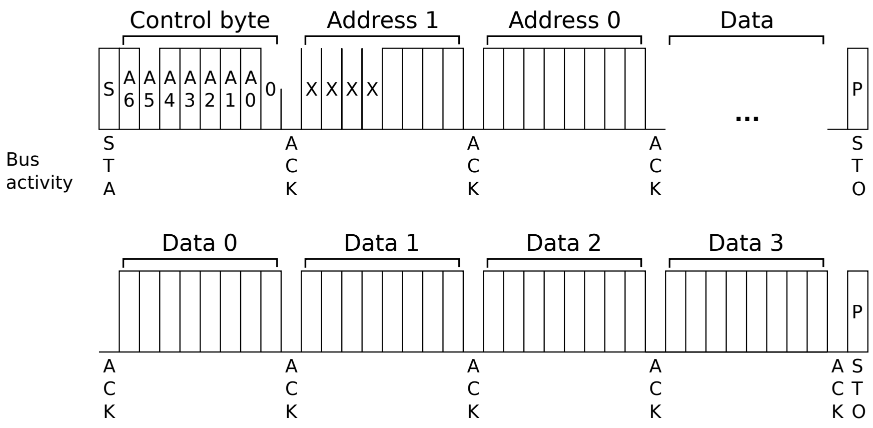

Figure 3: SysMon write operation

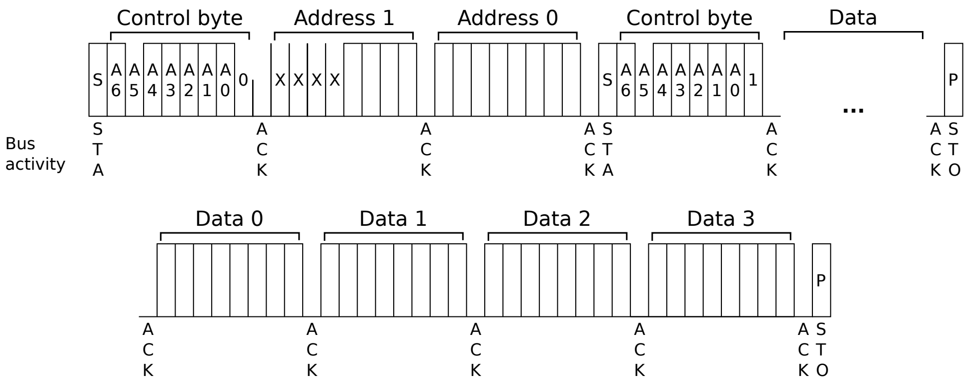

Figure 4: SysMon read operation

I^{2}C accesses as outlined in the previous section. The commands supported by the *vbcp_wb* module are shown in Table 2.

One noteworthy subject here is the *writemregs* command. This command allows writing more up to eight words to the same Wishbone register. It is useful when one wants to use a Wishbone register as a proxy for accessing an on-board peripheral where large amounts of data are to be written. An external memory is a good example of such a peripheral.

In principle, the *writemregs* is a *writereg* with multiple data words packed together, as outlined in Figure 5. In this figure, each data word is split in

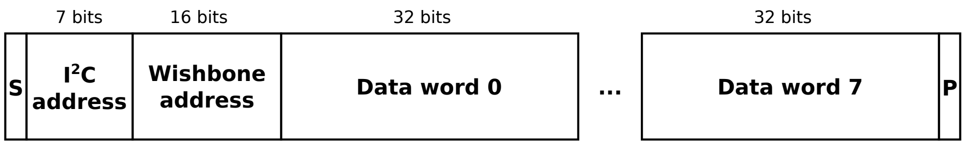

Figure 5: SysMon write using *writemregs*

Table 2: The *readreg* and *writereg* commands

Command	Description
writereg *slot addr val*	Writes the *hex* value *val* to hex address *addr* of board in slot number *slot*
writemregs *slot addr v1 .. v8*	This command is similar to the *writereg* command, but it allows writing up to eight different values to the same Wishbone register. The values are given in hexadecimal format and are separate by spaces
readreg *slot addr*	Returns the value of register at hex address *addr* of board in slot number *slot*

four bytes as outlined in Figure 3, with an ACK sent by the VME board after every byte.

As Figure 5 shows, the data words are sent in little-endian order, word 0 is sent first, followed by word 1 and so forth, until word 7.

5 Implementation

In order to perform low-level I^{2}C transfers, the *i2c_slave* module is instantiated and used within the *vbcp_wb* module. The outputs of the *i2c_slave* module are used as controls for an eight-state finite state machine (FSM), a simplified version of which is shown in Figure 6. Table 3 also lists the states of the state machine.

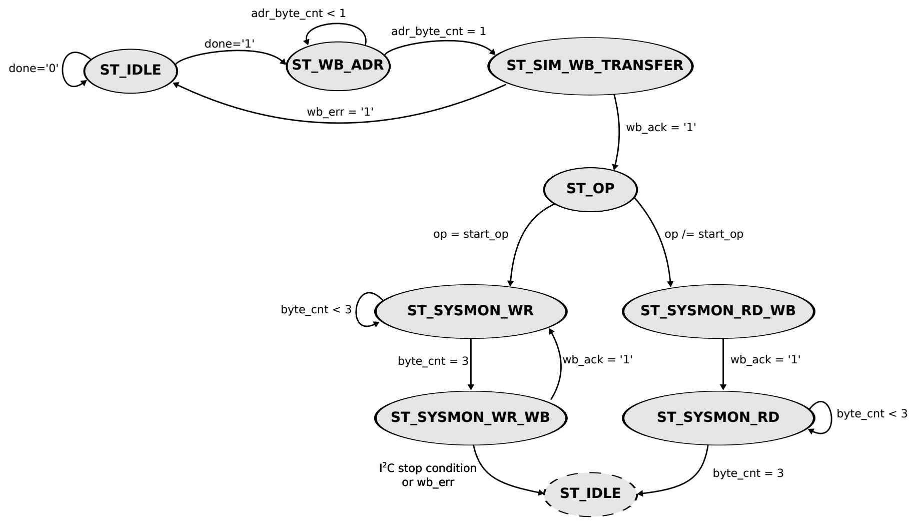

Figure 6: Main FSM of *vbcp_wb* module

Table 3: States of *vbcp_wb* FSM

State	Description
IDLE	Wait for the *i2c_slave* module to receive the VBCP address and go to *WB_ADR*. The starting value at the *op_o* output of the *i2c_slave* module is stored for checking in *OP*
WB_ADR	Shift in the two address bytes sent via VBCP and go to *SIM_WB_TRANSF*
SIM_WB_TRANSF	Start a Wishbone read transfer from address received in previous state and go to *OP* if Wishbone address exists (Wishbone *ack* received), or *IDLE* otherwise (Wishbone *err* received)
OP	Check the *op_o* output of the *i2c_slave* module. If different from the value at the start, go to *SYSMON_RD_WB* state (SysMon is reading from *vbcp_wb*), otherwise continue shifting in bytes (SysMon writing to *vbcp_wb*)
SYSMON_WR	Continue reading up to four bytes sent by the SysMon and go to *SYSMON_WR_WB*
SYSMON_WR_WB	Perform a Wishbone write transfer to the register with the address obtained in *WB_ADR*
SYSMON_RD_WB	Perform a Wishbone read transfer from the address obtained in *WB_ADR* and go to *SYSMON_RD*
SYSMON_RD	Shift out the four bytes of the Wishbone register when the *i2c_slave* module successfully finishes a write

When the *i2c_slave* module finishes a transfer (signaled by a *done_p_o* pulse), the status is checked and if it is as expected (e.g., *address good* while in the *IDLE* state), the FSM advances to the next state. Where the SysMon appears in the state names, it indicates what the SysMon action is. For example, if the state of the FSM is *SYSMON_WR*, this means the SysMon is writing and the *vbcp_wb* is reading.

To better understand how the FSM operates, Figures 7 and 8 can be consulted, where the state of the FSM is shown during reads and writes from the SysMon.

When the SysMon writes (Figure 7), the *vbcp_wb* module waits in the *IDLE* state until the I^{2}C address is received, then, while in the *WB_ADR* state, it shifts in the Wishbone address. A Wishbone transfer is then simulated with the received the address and if this address exists (a Wishbone *ack* is received), the first byte is shifted in while in the *OP* state, followed by the next three bytes while in the *SYSMON_WR* state. Finally, the register is written to in the *SYSMON_WR_WB* state.

When the SysMon reads (Figure 8), the first few steps are the same as for a read. The address is shifted in and checked in the Wishbone transfer simulation state. In the case of a SysMon reading from a board, however, the I^{2}C transfer is restarted and the order is reversed (SysMon starts reading). Thus, while in *OP*, the FSM detects a different value of *op_o* and goes into the *SYSMON_RD_WB* state. The value of the register is read while in this state, and sent via VBCP in the *SYSMON_RD* state.

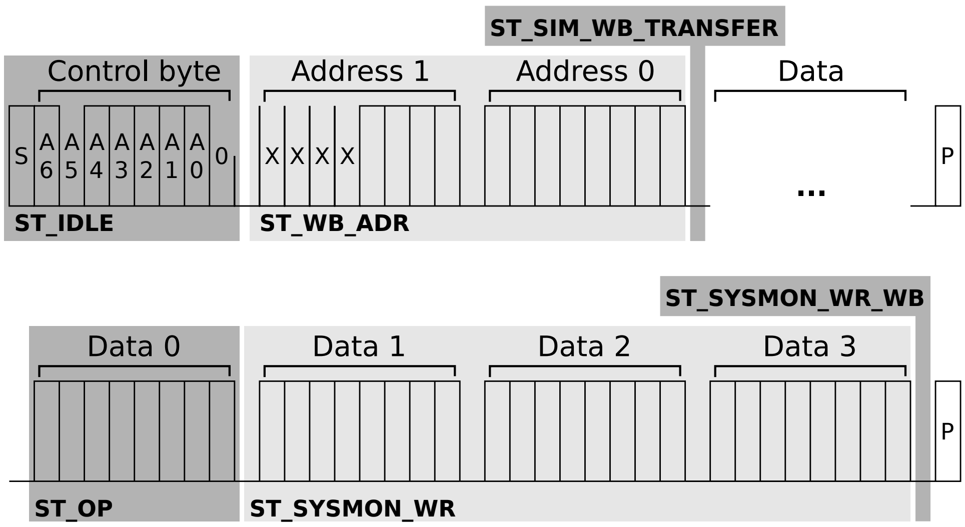

Figure 7: FSM states when the SysMon writes to the *vbcp_wb*

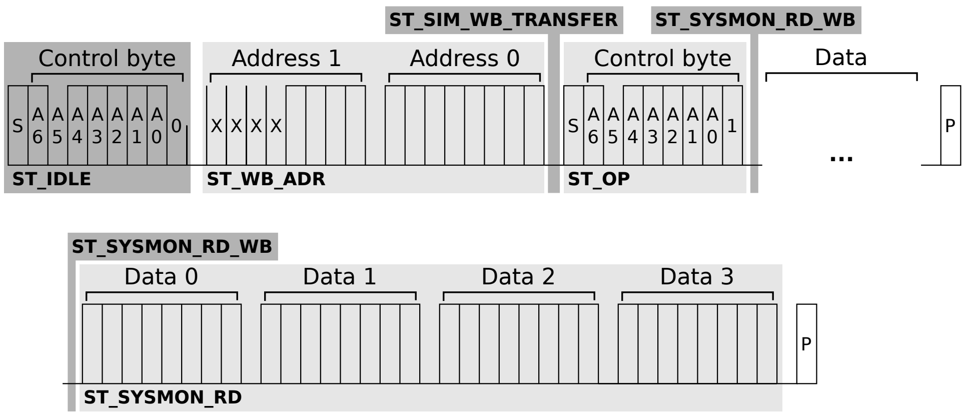

Figure 8: FSM states when the SysMon reads from the *vbcp_wb*

6 Synthesis results

The synthesis results for the *vbcp_wb* design using *xst* on the Spartan-6 XC6SLX45T are shown in Table 4.

Table 4: Synthesis results

Resource	**Used**	**Available**	**%**
Slices	76	6822	1.1
Slice registers	172	54576	0.3
LUTs	151	27288	0.6

References

[1] OpenCores, "Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores." http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[2] ELMA, "New SysMon User Manual Rev. 1.11." http://www.ohwr.org/documents/226.

[3] ELMA, "Access to board data using SNMP and I2C." http://www.ohwr.org/documents/227.