Communication between digital circuits in SoCs – a new master lecture

Lukas Mennicke, Heinz Endres
University of Applied Sciences Würzburg-Schweinfurt
D-97421 Schweinfurt, Ignaz-Schön-Str. 11, Germany
lukas.mennicke@student.fhws.de, heinz.endres@fhws.de

***Abstract* – Today we have lots of chip-systems in our lives. E.g. the processor in a modern smartphone consists of multiple CPU cores and peripherals on the same die. These complex chips are called System-on-Chips (SoCs). For digital circuits, which aim for clock frequencies of a couple of 100 MHz, the distance between the electrical circuits is very important. If there are different systems on the same chip, the communication between them is even more challenging. The paper will outline a method for the data transmission between the independently parts of a SoC.**

In order to prepare students for the upcoming and current trend towards SoCs, a new master lecture is planned based on a Xilinx SoC with a corresponding development kit called "Zedboard". It has lots of peripherals, e.g. a HDMI-Interface and an Oled-Display. In the center of the board stand the Zynq-7000 SoC, which contains two CPU cores and a huge field-programmable-gate-array (FPGA).

I. INTRODUCTION

The Advanced Microcontroller Bus Architecture (AMBA) was developed by the British company ARM Limited. It is an open standard for the communication between independent parts of SoCs. The Advanced extensible Interface (AXI) is the most widely used interface of the AMBA. [1]

The AXI interface is based on the master and slave communication principle and works in the full-duplex-mode. At the same time, data can be sent and received. The AXI interface has two kind of types: AXI-full and AXI-lite. The lite is a simplified protocol version. It only utilizes the necessary control signals, it is easier to understand and saves area on the chip. The disadvantage is the missing burst mode. A burst mode is a data transmission that comprises several data words without initializing them separately. E.g. only the initial address of the first data word is transmitted, and for all following data words the memory address is incremented automatically.

The following section explains the communication between the CPU as master and FPGA as slave using an AXI-lite interface.

II. ILLUSTRATIVE OPERATION OF THE AXI INTERFACE

The AXI interface is responsible for data transmission between two function blocks. These blocks can be two separate circuits on a FPGA, or a CPU and a FPGA. Figure 1 schematically illustrates data transfer between the master (CPU) and the slave (FPGA) on a Zynq-7000 SoC.

CPU and FPGA parts are optically separated by a dashed line. Since the CPU acts as master, all data transfer requests are made by the CPU. As shown in Figure 1, the CPU memory space consists of three parts: RAM, PL (Programmable Logic), and IO peripherals, each having different base addresses. For the initialization of a transmission, only the PL (MAXI_GP0) register is read or written from the CPU point of view. MAXI_GP0 stands for Master AXI general purpose 0 register. However, in this area, there are no memory cells, such as a RAM memory. Address and data are directly passed to a module called "Master Interconnect for Slave Peripherals". This module serves as a "bridge station" between CPU and FPGA. It is responsible for the necessary handshakes and control signals.

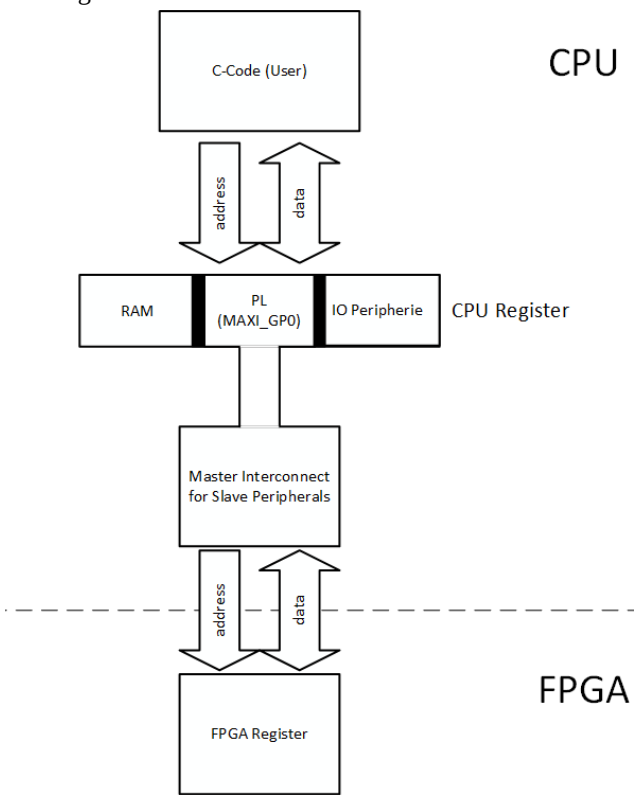

Figure 1. Schematic representation of the data transfer using the AXI interface between CPU and FPGA

The registers in the address area of the PL (MAXI_GP0) are physically located on the FPGA, but virtually on the CPU. From the viewpoint of the CPU, no difference is noticed - whether data is transferred between RAM or FPGA registers.

III. Structure of the AXI-Lite Interface with All Required Control Signals

The internal construction of the CPU understands if the address for the next memory operation is in the address range of the FPGA and will automatically contact the Master Interconnect for Slave Peripherals.

TABLE I.
SIGNALS OF THE AXI-LITE-INTERFACE [2]

global	**read address**	**read data**
S_ACLK	S_ARVALD	rvalid
S_ARESETN	arready	S_READY
	S_ARADDR	rdata
		rresp
write address	**write data**	**write validation**
S_AWVALID	S_WVALID	bvalid
awready	wready	S_BREADY
S_AWADDR	S_WDATA	bresp
	S_WSTRB	

This block is responsible for the handshake signals of the AXI interface. Table 1 lists all the signals needed for a data exchange. All signals in lower-case are generated by the slave (FPGA), all others preceded by a "S_" by the master (CPU).

The signals "valid" and "ready" always serve as a handshake to transfer the data in the same category without errors. In total, there are six categories: In order to write data to a FPGA register, "write address", "write data" and "write validation" are required. To read an FPGA register, "read address" and "read data" are used. The global signals are necessary for both actions.

The global signal S_ACLK is the clock signal. A 100MHz clock is standardly used on the Zedboard. However, since the AXI interface protocol has no frequency limit, other frequencies are possible. The maximum transmission rate depends on the hardware specification. S_ARESETN is used to reset the bus. The remaining signals and their meanings will be explained in the next chapter.

IV. CPU writes to the FPGA

First, the CPU needs to initialize a write operation. In the following example the hexadecimal value 0x64 will be assigned to the address 0x43C00000. The C code looks like this:

*(u32 *)0x43C00000 = 0x64;*

The CPU sends the operation command to the "Master Interconnect for Slave Peripherals". It is responsible for the generation of the control signals and the correct transmission sequence. For writing to a PL-register "write address", "write data", and "write validation" categories from Table 1 are necessary.

The bubbles in Figure 2 show the four states of a transmission. The control signals at the arrows are the conditions to reach the next state. They are all one bit wide. As shown in Figure 3, all signals are set to ‘0’ at the beginning of a transmission (state "bus inactive"). To initiate a writing operation, we need four signals: awready, S_AWVALID, wready and S_WVALID. The signals awready and wready are generated by the slave (written in lower-case letters). Awready "low" signals the master that the slave is ready for address transmission. Wready "low" indicates that the slave is ready to transmit the data.

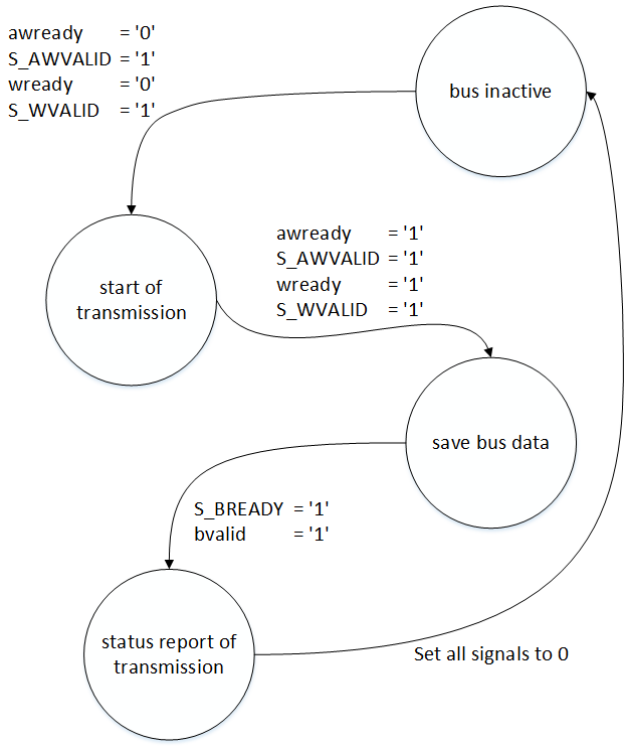

Figure 2. The Moore machine describes the control signals, if the CPU writes data to the FPGA

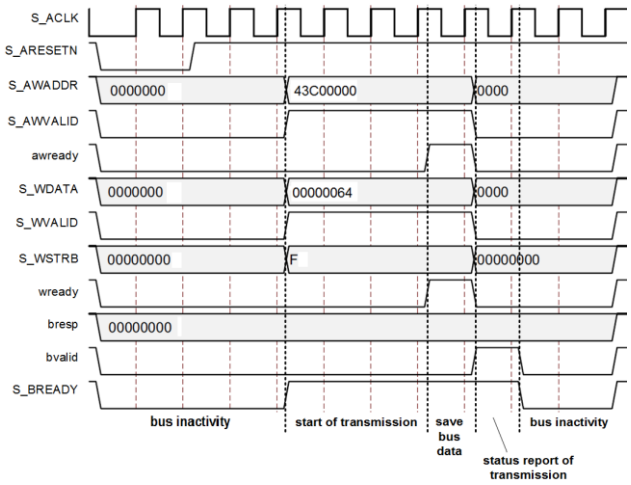

Figure 3. Timing chart for a single data write operation from the CPU to the FPGA [3]

The slave is automatically ready if the bus is inactive and waits for commands from the master. The master applies the address (0x43C00000) to the bus line S_AWADDR. This 32 bit wide signal is responsible for address transmission during a write operation. At the same time, the signal S_AWVALID is set to "high", which validates the address. The data is handled in a very similar way. The master applies the data (0x00000064) to the bus line S_WDATA and displays the validity for the slave using the S_WVALID signal. S_WSTRB is set in the same time step. This signal will be discussed in more

detail later. Finally, the master state machine switches to "start of transmission" (see Figure 2). It sets the signal S_BREADY to "high", and waits for a response from the slave.

Next, the slave needs to read the address and the data from the bus. The signal S_WSTRB is 0xF, so the complete data (0x00000064) is stored in the register with the address 0x43C00000.

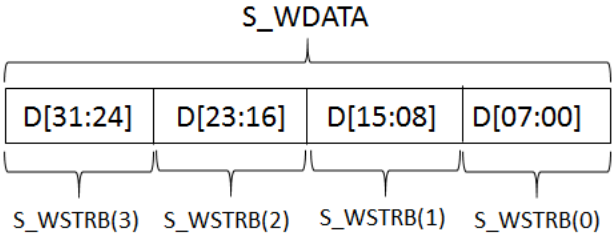

Figure 4. Structure of the S_WSTRB signal

Figure 4 shows the structure of the S_WSTRB signal, which is used for the memory operation. For a 32 bit wide data bus, it is 4 bit wide and responsible for validating the data. With the help of the S_WSTRB signal, single bytes can be transferred to a register. If the signal is 0xF, all four bytes are stored in the FPGA register. For a value of 0x9, the data bits of D[31:24] and D[07:00] are transferred to the register, and D[23:08] remains unchanged. Each bit of signal S_WSTRB tells the slave whether the corresponding byte of the data is to be stored or discarded.

The slave has now transferred the value 0x00000064 to its own register 0x43C00000. It signals with awready or wready, that the address or data was received, respectively. Now the FSM reached the state "save bus data" in Figure 2. Finally, the status of the transmission is transmitted to the master before the state of the bus is reseted.

TABLE II.
CODING OF THE BRESP AND RRESP SIGNAL [4]

rresp [1:0] bresp [1:0]	**status**
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

The slave recognized by the S_BREADY signal, that the master is ready for a status transfer. Table 2 lists the different possibilities of a transfer. When a FPGA register is written, bresp signals the status of the transfer, the rresp signal is responsible for a read operation.

There are four options: OKAY (0b00) stands for a perfect transfer with no errors. EXOKAY (0b01) is only used for exclusive transmissions. These are not integrated in the AXI-lite interface and are mentioned here only for the sake of completeness. SLVERR (0b10) stands for "Slave Error". This error is thrown when the master has sent a data transfer request to the slave, but an error has occurred with it. DECERR (0b11) is the abbreviation for "Decode error". This error occurs when the master wants to interfere an address that has not been assigned to a slave. [4]

The bvalid signal indicates to the master that a valid bresp signal is present. In Figure 3, the timing chart, the bresp signal is 0b00, which indicates a successful transfer. This corresponds to the FSM state "status report of transmission" in Figure 2. Finally, all signals are set to "low" and the bus reached the inactive state again. The slave waits for a new request from the master.

V. CPU READS FROM THE FPGA

The read operation must be initialized by the CPU (master). In the previous chapter the value 0x64 was transferred to the register 0x43C00000. Now the register will be read from the CPU. In the C code, the read operation would look as follows:

*u32 registerContent = (u32 *)0x43C00000;*

The content of the register 0x43C00000 is assigned to the variable *registerContent*. The CPU sends the operation command to the "Master Interconnect for Slave Peripherals". It is responsible for the generation of the control signals and the sequence of the transmission. For writing to a PL-register, "read address" and "read data" categories from Table 1 are necessary.

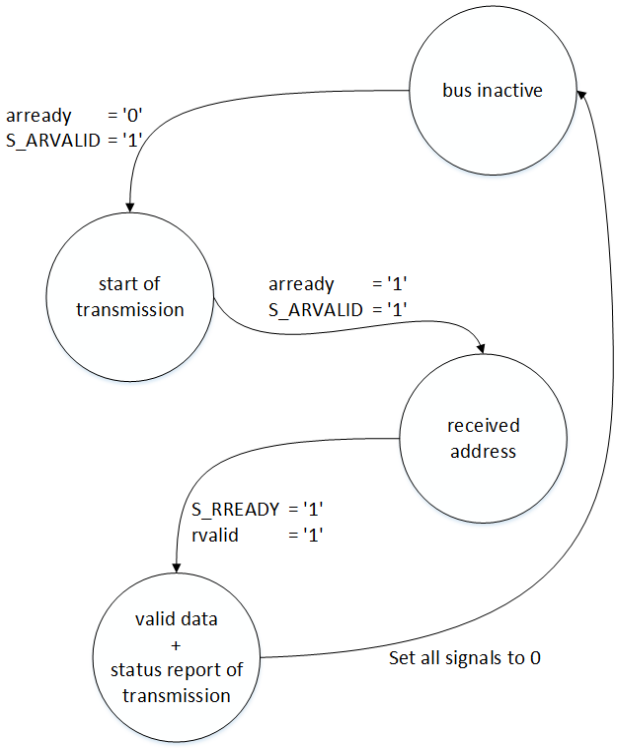

Figure 5. The Moore machine describes the control signals, if the CPU reads data from the FPGA

Figure 5 shows the sequence of a read operation of the FPGA register. First, the master is in the state of bus inactivity. All channels are "low". As in the previous example, all signals produced by the slave are lowercase. With arready "low" the slave signals that it is ready for an address transfer. While the bus is inactive, the slave waits for instructions of the master. The timing chart in Figure 6 shows the exact sequence of a read operation. To initialize, the master specifies the address of the register to be read to the bus S_ARADDR. Simultaneously, the signal S_ARVALID is set to "high". It indicates the slave, that the address is valid. With the S_RREADY signal the master indicates that it is ready for receiving the data and

the status of the transmission. Now, the state "start of transmission" in Figure 5 is reached. The master puts all necessary data on the bus and waits for the slave to respond.

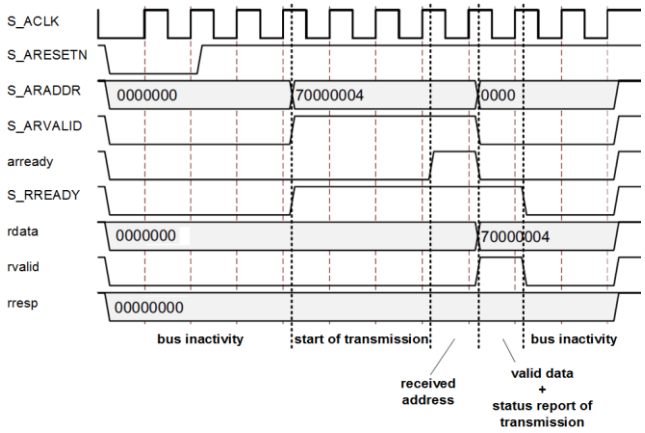

Figure 6. Timing chart for a single data read operation [5]

In the first step, the slave receives the address of the register to be read. To receive no further addresses, aready set to "high". In Figure 5, the instantaneous state is "received address".

As shown in the timing chart, the data of the register (0x43C00000) will be put on the bus in the next clock period The data channel rdata is assigned the value 0x00000064. This corresponds to the value that was written in the previous chapter in the register. At the same time, the status of data transmission is passed to the master with the signal rresp. This signal is equivalent to the signal bresp of the write operation and is described in detail in the previous chapter. There are no changes at the signal rresp in the timing chart, because 0b00 stand for "OKAY" - a successful transfer. Rvalid confirms the master that the two signals rdata and rresp are valid. Now the state "valid data + status report of transmission" is reached (see Figure 5).

The "Master interconnect for Slave Peripherals" receives the value of the register and outputs the data to the CPU. In the final step, all channels are set to "low". The bus returns to the idle-state and the slave waits until a new read request is send from the master.

VI. Conclusion

In modern systems with high performance demands a single processor core alone is usually insufficient. For example, complex graphical calculations require a separate Graphics Processing Unit (GPU). This paper outlines the ARM AXI interface, which allows a high-speed data transmission between a CPU and independent digital circuits. These high-speed communication systems are the key for modern System-on-Chip (SoC) architectures.

References

[1] AMBA Specifications, http://www.arm.com/products/system-ip/amba-specifications.php, accessing 2016-04-26

[2] AMBA AXI and ACE Protocol Specification, S122 (2011)

[3] Figure based on LogiCORE IP AXI4-Lite IPIF v2.0, S24 (2013)

[4] AMBA AXI and ACE Protocol Specification, S54 (2011)

[5] Figure based on LogiCORE IP AXI4-Lite IPIF v2.0, S23 (2013)