Internship Report

Design of a Serial Peripheral Interface (SPI) Master for the ANANAS FPGA

Simon Rosenkranz

18.10.2017 - 29.06.2018

Abstract

Throughout this internship, the basics of *Field Programmable Gate Array* (FPGA) based hardware design have been acquired by designing, verifying and implementing a *Serial Peripheral Interface* (SPI) master. The SPI should be configurable in terms of timing constraints, data transmission modes as well as the number of connected slave devices and the operating frequency. It has been implemented in a Spartan-6 FPGA on the *Analog Network Attached Sampling unit* (ANANAS) [1] and verified by simulation and using oscilloscope measurements.

Contents

1 Learning the Basics: Clock Divider

1.1 Motivation

In order to get to know the concepts and tools for field-programmable gate array (FPGA) based hardware design a simple clock divider was to be designed and simulated. Communication between hardware components on the *Analog Network Attached Sampling unit* (ANANAS) [1] needs to be driven by clocks with various frequencies specified by the manufacturer. However, it is not possible to control every single component on the board with an individual global FPGA clock, due to limited resources (the *Spartan-6* FPGA used in the *Flyspi* FPGA system provides 16 global clocks in total [2]). It is rather favoured to provide a $f_{in} \approx 40\,\mathrm{MHz}$ global clock from the *Open Core Protocol bus* (OCP) of the FPGA which is then scaled down by a factor of $\frac{1}{n}$ to the individually required frequencies f_{out} of the components:

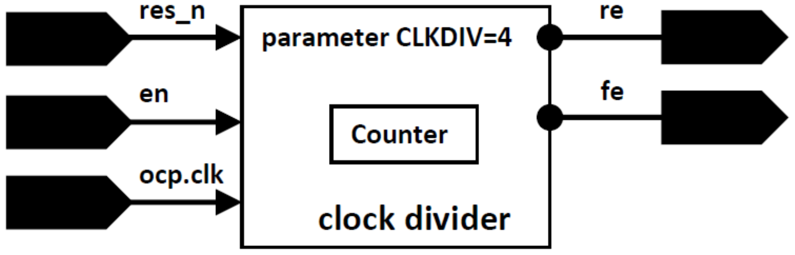
$$f_{out} = \frac{f_{in}}{n} \tag{1}$$

For this task, a clock divider can be used. However, one has to be careful with this term because it refers to the generation of a new, separate slower clock. In our case the module to be designed should only mimic the behaviour of a traditional clock divider by exploiting the clock enable of flip-flops in higher level modules. Further, only even multiples of the global OCP clock period should be configurable.

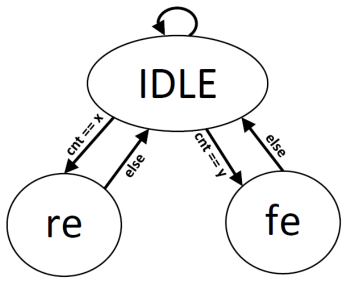
1.2 Specification, Verification and Implementation

The specification of the module to be designed marks the first step of every design flow. After the basic features of the module have been formulated, a rough specification of the design by an abstract visualization (in a high level language) was carried out. This was realized by modelling the behaviour of the module in a block diagram and a state machine. The basic concept of the module `clock divider` has been illustrated in such a way and can be seen in figure 1. The state machine process begins at the initial state `IDLE` (cf. figure 1b). The most simple way to mimic the behaviour of the operating clock is to flag the rising and falling edges of the clock signal with two separate states `re` and `fe` that are the output. A counter, based on a register of width depending on the division factor `CLKDIV`, indicates the transitions between `IDLE` and `re` or `fe`. The counter value `cnt` is increased every OCP clock cycle and is reset when `cnt==y`. Whenever `cnt` equals one of the two specifiable values x or y, a state transition to the rising or falling edge states `re` or `fe` occurs and a pulse is generated on the corresponding outputs (cf. figure 1a) with a duration of one OCP clock cycle. For `y=2x`, this infers a downgrade of the duty cycle by 50%. The pulses on these outputs can then be used as clock enables of flip-flops within the modules describing the interfaces of data flow between different hardware components. Furthermore, only even division factors should be valid in order to guarantee a synchronous communication specification. Therefore, an exclusion of odd division factors has been implemented.

For the description of a system such as the clock divider a hardware description language (HDL) is needed. Throughout this internship System-Verilog has been used for the description, verification and implementation of all designs that have been and will be discussed in the following. For verification purposes it is necessary to simulate the behaviour of the design to be implemented in hardware. This task has been carried out with the multi-language HDL simulation environment *ModelSim*. This tool provides a graphical user interface (GUI) for the observation of all in- and output signals of individual modules in the design. It allows for verification of the temporal behaviour of the module given ideal operating conditions, thus no timing constraints are considered. In simulation, the module proved to correctly trigger the rising and falling edge signals. Also, the illegal assignment of odd division factors resulting in a break of the simulation could be observed. Finally, the implementation of the verified design, meaning the procedure placing and interconnecting logic elements on the grid of the FPGA, has been done with the software *Xilinx ISE*.



(a) Block diagram



(b) State machine

Figure 1: State machine for clock divider module. Whenever the counter reaches the values x or y (typically $y = 2x$), a pulse is generated on `re` or `fe` with a length of one OCP clock cycle. The pulses represent a rising or a falling edge to be used as clock enables, respectively.

2 Implementation of a Serial Peripheral Interface Master

2.1 Motivation

A *Serial Peripheral Interface* (SPI) is a synchronous serial communication interface specification for short distance communication, primarily used in embedded systems developed by *Motorola* [3]. Next to I^{2}C, it has become a *de facto* standard for interconnection of digital circuits using a master-slave architecture. A rough overview over such a system is provided in figure 2. In principle, a single master can be connected to multiple slave devices. The master is able to initialize communication with a single slave via a so called *chip-* or *slave-select* ($\overline{\text{CS}}$). The $\overline{\text{CS}}$ is active low so when it gets pulled down, the selected slave is informed that the master has initialized the communication protocol. Then, two signals are transmitted from the master to the slave: The *Serial Clock* (SCLK) and the data output from the master to the slave, which is called *Master Output Slave Input* (MOSI). In parallel, the slave delivers an output of his data to the master on the so called *Master Input Slave Output* (MISO) line. If the slave is not selected, the MISO pin is set on high impedance. For each clock cycle of the SCLK one bit of the master's data is shifted onto the MOSI line to the slave and vice versa (on the MISO line). This means that independent of a master `READ` or `WRITE` request, both devices simultaneously clock in and transfer data on the bus from their respective buffers to the other participant.

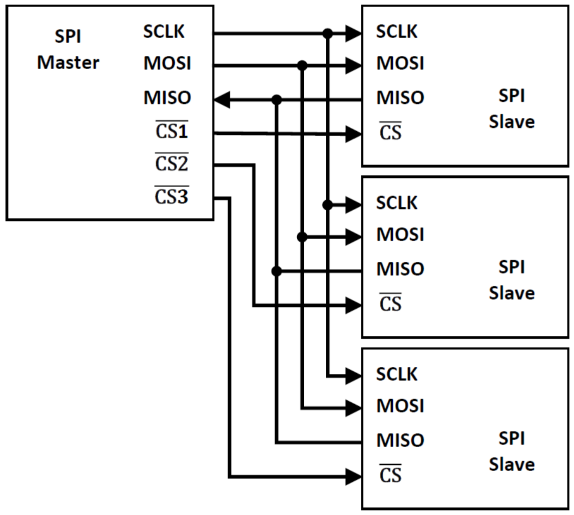


Figure 2: SPI master-slave architecture with an example set of three slaves connected to a single master. There are in general four signals necessary for a working SPI master-slave communication: The serial clock (SCLK) guarantees synchronous data transmission on the master output slave input (MOSI) and master input slave output (MISO) lines. In order to initialize the communication, the chip-select ($\overline{\text{CS}}$) of a certain slave device has to be pulled down. However, if there is only a single slave connected to the master, $\overline{\text{CS}}$ is not needed. In general an arbitrary number of slaves can be connected to a single master. [3]

There are in general four modes for the communication between master and slave. They depend on the phase and the polarity of the SCLK with respect to the active edge relevant for MOSI and MISO. Figure 3 and table 1 provide an overview over the properties of the four different modes.

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	Active Edge
0	0	0	positive
1	0	1	negative
2	1	0	negative
3	1	1	positive

Table 1: Characteristics of SPI modes

In the previous implementation of the SPI module only modes 0 and 2 have been supported since clock inversion has been considered only. Since a slave on the ANANAS board uses mode 3, a quick unflexible adaption of the given module was previously written in order to make the communication work. This has been the case for a chip serving as an SPI-to-I^{2}C-converter (a *SC18IS600* chip from *NXP Semiconductors*) which only supports SPI mode 3 [4]. In addition, the waiting time between the assertion of the $\overline{\text{CS}}$ and the generation of the SCLK could not be adjusted. The SPI-to-I^{2}C-chip requires specific timing constraints for the initialization of a SPI based communication [4]. Therefore, the goal of this internship was to write, verify and implement a full-duplex capable SPI master module, configurable in terms of the number of connected slave devices, the operating frequency and the $\overline{\text{CS}}$ to SCLK pause with support of all four possible SPI modes.

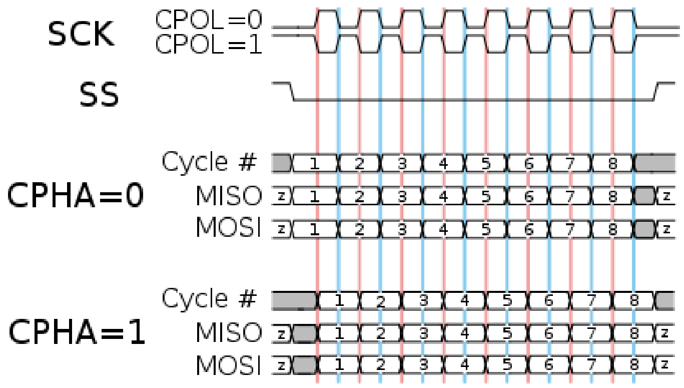


Figure 3: Timing diagram of data transmission for different SPI modes. The communication starts as soon as the slave-/chip-select (SS) is pulled down. Data has to be held long enough on the MOSI and MISO lines in order to be valid, meaning that depending on the selected SPI mode the bits have to be stable either at the rising or the falling edge of the serial clock (SCK). The points of time at which the data is stable can be shifted with respect to SCK by half a cycle which defines the clock phase (CPHA). Further, the polarity of the clock (CPOL) can be changed between active high and active low, leading to four different modes at which the SPI can operate. [3]

2.2 Specification

The FPGA uses the OCP for the contribution of all important global signals such as the global reset and the global clock as well as the distribution of data and event and command indicating flags for slow control interfaces. For a neat and simple handling of this information with the SPI however it is preferred to make use of interfaces connecting it to the OCP. The common `axi4-stream` interface has been chosen for this task. A schematic of the data flow for a transmission between a host and a slave device via SPI using an `axi4-stream` adapter can be seen in figure 4.

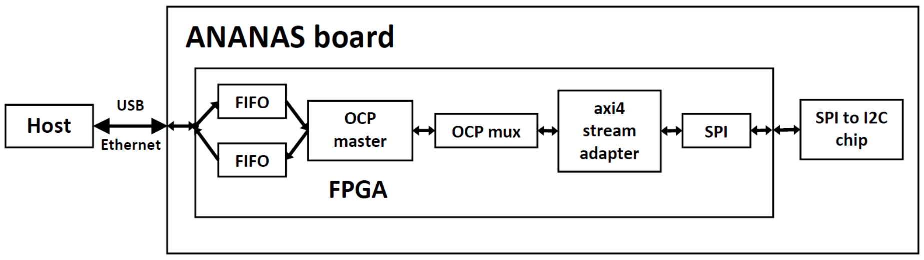


Figure 4: Schematic of the data flow from the host to a chip on the ANANAS board. Data from the host is written to and read from the FPGA via USB or Ethernet connection. In both directions respectively a *First In - First Out* buffer (FIFO) is used to temporarily store the incoming and outgoing data from the FPGA. The OCP bus is then connected to the SPI, which communicates with the SPI-to-I^{2}C-chip, via the `axi4-stream` adapter.

The SPI related in- and outputs of the `axi4-stream` adapter can roughly be subdivided into an OCP output (corresponding to MOSI) and an OCP input (corresponding to MISO) direction as described in the sketch of the interface in the upper half of figure 7. The flags and signals are chosen to be consistent name-wise in both directions of data transmission, to and from the SPI module. Eight bit wide parallel data packages are transmitted on the `tdata` lines to one of the N slaves tied to the SPI. The slave is specified with the `tuser` lines. The width $(x+1)$ of `tuser` depends on N, where

$$x = \min \{k \in \mathbb{Z} \,|\, k \geq \log_2 N\}\,. \tag{2}$$

The extra bit added on top indicates the command initialized by the host (0 for `READ` and 1 for `WRITE`). Figure 5 shows a simplified timing diagram of a SPI data transmission with the `axi4-stream` adapter.

The state machine chosen for the SPI is now depicted in figure 6. It can be subdivided into six different states:

- `IDLE`: Initial state for reset and whenever there is currently no data to be processed. On the next packet flagged with `tfirst` from the `axi4-stream` adapter the following state will be the configuration state `MODECFG` of the SPI mode. This is however only possible, if the output register to the OCP bus of the SPI master is empty (`ocpin_tvalid` is low). If the packet is not the first, the state machine will directly move to the data transmission state `DATA`.
- `MODECFG`: Configuration state for the SPI mode. It is the first step for each master-slave communication between the host and a newly selected slave. Here, one of the four available SPI modes needed by the selected slave device (cf. table 1) is configured and not changed until a new slave is selected. It is important that CPOL is configured before the $\overline{\text{CS}}$ is pulled down so that no wrong edges are detected.
- `CSASS`: In this state, the $\overline{\text{CS}}$ of the desired slave is pulled down, informing the slave to listen to the master.
- `CSWAITPRE`: State that can be used to set the number of clock cycles to the SCLK first edge which may be requested due to timing constraints. Is selected once at the start of every communication between the host and the SPI slave.

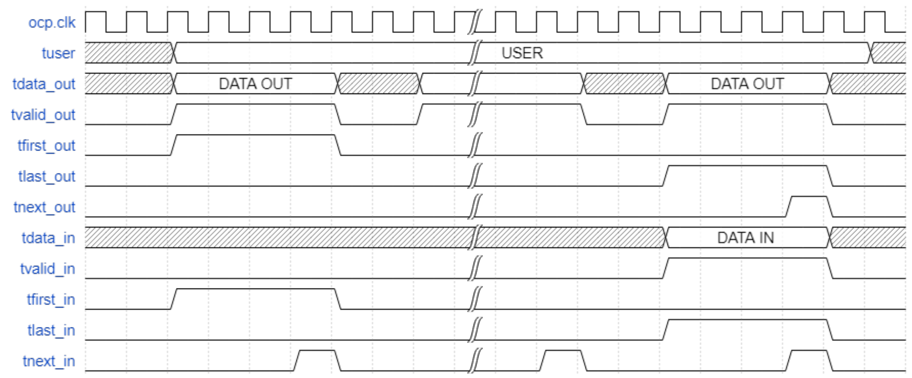


Figure 5: Timing diagram of SPI data transmission with the `axi4-stream` adapter. The signals describing the data on MOSI and MISO are labelled with `_out` and `_in`, respectively. `tfirst` and `tlast` indicate the first and the last data packet of the transmission with a specific SPI slave and are mirrored for consistency in both directions. In output direction, the flag `tvalid_out` is triggered whenever valid data to be written is available for the transmission on MOSI. It gets pulled down again when the transmission has been successful. This means that the flag is always active when an event occurs on the OCP. For the OCP input direction however, `tvalid_in` is only triggered when valid data to be read is available from the slave. This flag also allows slave blocking (cf. figure 6). In both directions, `tnext` is activated every time a data transaction has been successfully performed on MOSI or MISO, forcing the corresponding `tvalid` to go down again. It indicates that the SPI is ready to perform the transmission of the next data packet.

- `DATA`: Data transmission state. Depending on the width of the data packets to be transmitted, the state machine stays in this state for the corresponding number of clock cycles. Here, the SCLK is given out from the master to the slave. Also, the data from the master and the slave is clocked in on the MOSI and MISO lines, respectively. When there is currently no valid data from the master and the recently transmitted packet was not flagged as last, the state machine changes to the resting state `IDLE`. If the previous packet was indicated as the last one of the communication, the system changes to the state `CSWAIT`.

- `CSWAITPOST`: Similar to `CSWAITPRE` the duration of this state is specified in order to match timing constraints. Is selected once after the transmission of the last data packet.

- `CSDEASS`: Final state for each master-slave interaction. Here, the $\overline{\text{CS}}$ of the previously selected slave is released, indicating the communication to be finished.

This control path of the SPI module is strictly separated from the data path processing the output of the serial data, the serial clock as well as the chip select. For the discussion of the implementation of the SPI, let us consider the block diagram for the SPI represented in figure 7. We have not only made use of an `axi4-stream` interface but also of an existing interface for the necessary in- and outputs of the SPI. It provides easy access to all important signals (`sclk`, `mosi`, `cs` and `miso`).

The parameter `CSNUM` specifies the number N of slave devices tied to the SPI. Since every individual device needs a separate line for the $\overline{\text{CS}}$, so called *1-in-n-encoding* has been used to convert the selected chip from `tuser` to N interconnections between the master and the slaves. The chip select `cs` in figure 7 of the desired slave is not active low but active high since it gets inverted by the interface `spi_if`.

The conversion from eight bit wide data packets received by the SPI from the `axi4-stream` adapter to serial data clocked in on the MOSI line is realized using a shift register. For every cycle of the SCLK, the current *Most Significant Bit* (MSB) is shifted out on the MOSI line while the content left in the register moves up by one position. In parallel, serial data from the MISO line is clocked into another

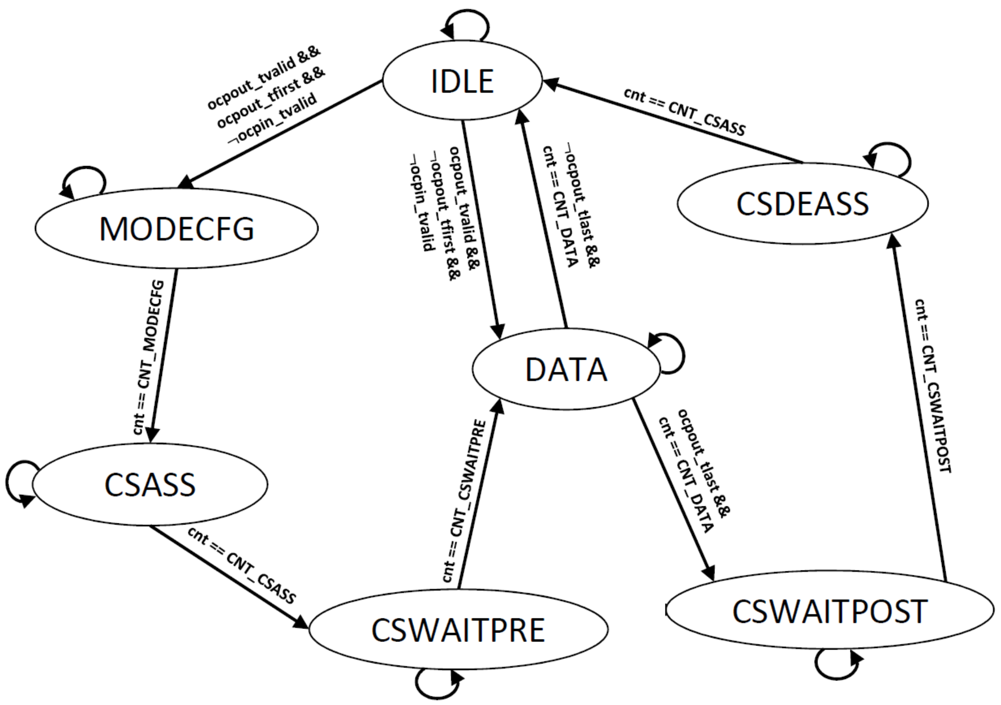


Figure 6: SPI state machine. The prenames `ocpout` and `ocpin` denote the direction of the signals with respect to MOSI and MISO (corresponding to the sketch of the `axi4-stream` in the upper half of figure 7). In general, each state (with the exception of `IDLE`) is provided with a maximum counter value that is mostly the only condition for a state transition. Only for the transition from `IDLE` and `DATA` there have been defined more input dependent transition conditions.

shift register, leading to an eight bit wide packet of parallel data after a full transmission step which can then be fed to the `axi4-stream` adapter again.

In order to provide an operating frequency that can be easily adjusted to the needs of the SPI slaves, the previously discussed `clock divider` has been implemented. The whole SPI protocol is driven by the 40 MHz OCP clock. The frequency of the state machine and SCLK is then varied by using the clock enable pin of the flip-flops by triggering on the outputs `re` and `fe` of `clock divider`. It is given by equation (1), where n corresponds to the parameter `CLKDIV`. Further, the user is able to declare the number of downscaled clock cycles that the module should rest in the various states defined in the state machine in figure 6 with the parameters containing the prefix `CNT_`. An implemented counter is able to control the state machine process. This especially important for the definition of the waiting periods in `CSWAIT` used to buffer the timing constraints resulting from the hardware.

The different SPI modes shown in table 1 are generated by manipulating the downscaled SCLK given out by the master. In order to vary the CPOL (defined by the parameter `cpol`), the SCLK can be inverted. The phase shift to the other clock edge for `cpha=1'b1` can be achieved by an additional procedural assignment of the SCLK, leading to a phase shift of half a clock cycle of the downscaled clock.

2.3 Verification on Hardware

In order to test the basic functionalities of the SPI module, an USB connection has been established between the ANANAS board and a Raspberry Pi used to perform the test transmissions of user defined data packets to the SPI. The MOSI line as well as the SCLK and the $\overline{\text{CS}}$ provided from the master

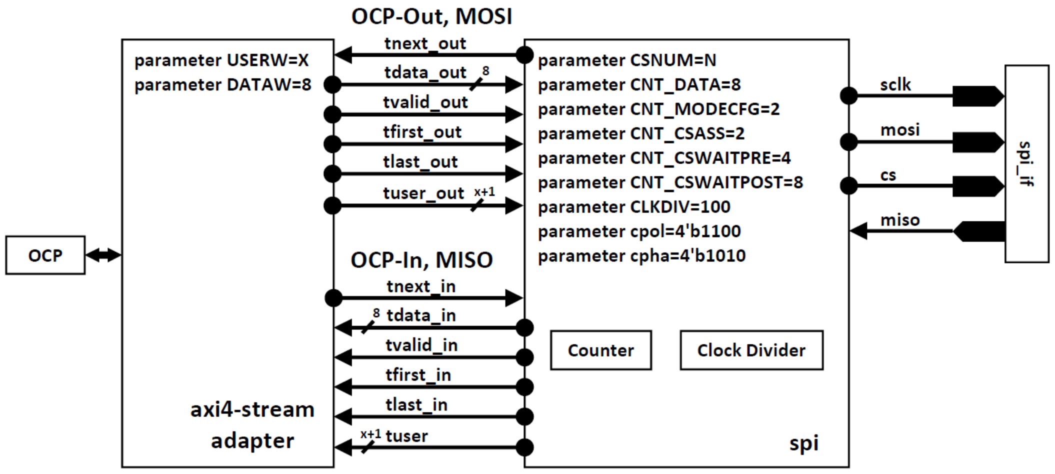


Figure 7: Schematic of the `axi4-stream` interface for the SPI (above) and block diagram of the SPI module (below). In- and outputs of the `axi4-stream` adapter are only displayed relating to the SPI. Their arrangement is chosen such that the flags describing the MOSI and the MISO lines are grouped together. Regarding the block diagram of the SPI, the parameters for the clock polarity and phase (`cpol` and `cpha`) are defined to be packed arrays. The length of each state within the state machine as well as the number of connected slaves and the clock dividing factor can be defined with integers.

have been recorded using an oscilloscope (*LeCroy WaveRunner 44Xi*). At this stage, there is no slave device connected. Therefore, we simply short MOSI and MISO for a loop-back configuration.

In figure 8, we can see the signals received by the oscilloscope for SPI mode 0 and different dividing factors `CLKDIV`. The SPI test consisted of two USB data packets containing eight bit wide data (`ab` for the first and `cd` for the second packet, corresponding to `10101011` and `11001101` in binary notation). Tests are carried out with `CSNUM=4` where each slave is driven with a different of the four available SPI modes. Only the $\overline{\text{CS}}$ driven by SPI mode 0 has been recorded by the oscilloscope due to the limited available pins of the FPGA. Other than that, the following parameters are set:

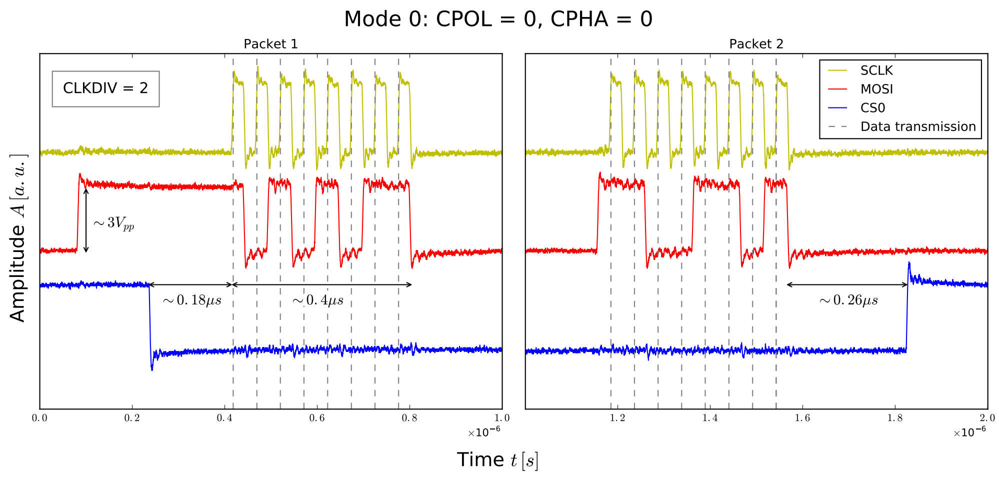
- `CNT_MODECFG=2`
- `CNT_CSASS=2`
- `CNT_CSWAITPRE=2` (for `CLKDIV=2`) and `CNT_CSWAITPRE=8` (for `CLKDIV=6`)
- `CNT_DATA=8`
- `CNT_CSWAITPOST=4` (for `CLKDIV=2`) and `CNT_CSWAITPOST=16` (for `CLKDIV=6`)
- `CNT_CSDEASS=2`

The yellow signal displays the SCLK, the red one corresponds to the MOSI line and the blue signal is the dummy $\overline{\text{CS}}$ corresponding to SPI mode 0. The active clock edges have been indicated with a grey dashed line. The first important thing one can notice from the plots in figure 8 is that the data on the MOSI line is held stable at the positive edge of SCLK for the duration of data transmission as desired for SPI mode 0. This means that the timing of SCLK and stable data on the MOSI line can be guaranteed. Additionally, the dummy $\overline{\text{CS}}$ for SPI mode 0 is active, thus it is pulled down correctly. Considering the duration of the different states of the state machine displayed in figure 6, a measurement of the approximate time needed for the completion of three steps has been included: The data transmission, the interval between pulling down the $\overline{\text{CS}}$ and the first SCLK edge as well as the one between the last transmitted bit and the release of the $\overline{\text{CS}}$. By using equation (1) with a frequency of the OCP clock of $f_{OCP} \approx 40\,\text{MHz}$ and the corresponding division factors, the following triggering frequencies are to be expected:

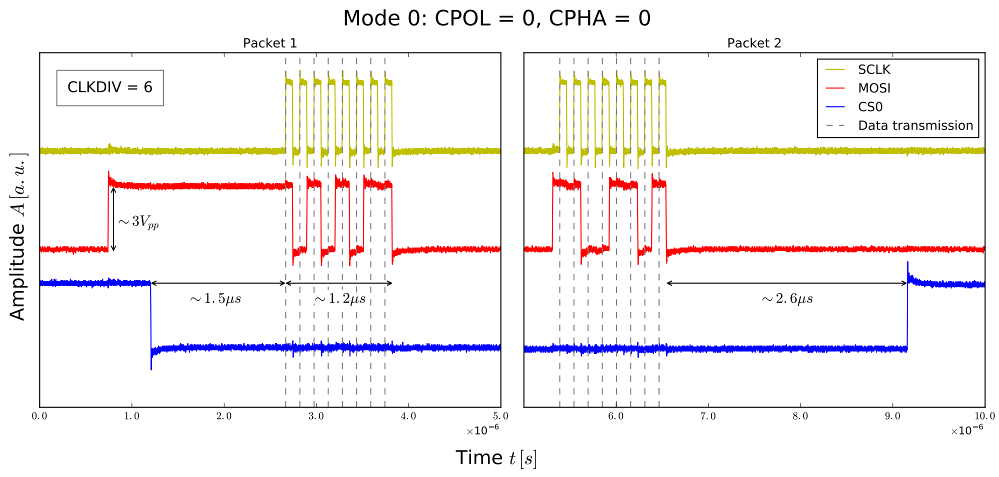
- $f_{\text{SPI}}^{\texttt{CLKDIV=2}} = 20\,\text{MHz}$
- $f_{\text{SPI}}^{\texttt{CLKDIV=6}} \approx 6.7\,\text{MHz}$

The expected time τ needed for N cycles of the downscaled clock with a frequency of f_{SPI} can be calculated with

$$\tau = \frac{N}{f_{\text{SPI}}}. \tag{3}$$



(a) Oscilloscope output for `ClKDIV=2`.



(b) Oscilloscope output for `CLKDIV=6`.

Figure 8: SPI test for mode 0. The output of two data packets `ab` and `cd` on the MOSI line as well as the $\overline{\text{CS}}$ of the dummy slave have been recorded with the oscilloscope. Grey dashed lines indicate the points of time where data from the MOSI line should be clocked in by the slave. Horizontal arrows have been used to mark the duration of three significant intervals within the state machine cycle.

With a counter value of eight for the state `DATA` as well as roughly four counter cycles between pulling/releasing the $\overline{\text{CS}}$ and the beginning/ending of the data transmission (corresponding to an expected duration of $\tau_{data} = 0.4\,\mu$s and $\tau_{pre,post} = 0.2\,\mu$s, respectively after equation (3)), the observed time intervals are satisfactory for the `CLKDIV=2` case (cf. figure 8a). Good results could be received for a `CLKDIV=6` and ten/sixteen counter cycles between pulling down/releasing the $\overline{\text{CS}}$ and the data transmission (expected time needed with equation (3): $\tau_{pre} = 1.5\,\mu$s and $\tau_{post} = 2.4\,\mu$s) additionally (cf. figure 8b). Note that the number of cycles within the state `CSASS` have been counted towards the pre-data waiting period together with the cycles of `CSWAITPRE`.

The signals received for the SPI modes 1, 2 and 3 can be seen in figure 9. For each row containing four plots of the oscilloscope output respectively, the same SPI mode has been used meaning that every row also corresponds to one of the remaining three different dummy chip selects. The SPI state machine has been triggered with a division factor `CLKDIV` of 2 for all plots within the left two columns and with one of 6 for all plots within the right two columns. Comparing the oscilloscope outputs for the SPI modes 1, 2 and 3 in figure 9 with the expected timing diagram in figure 3 one recognizes that the observed timing is also correct in terms of SCLK edge and CPOL. The $\overline{\text{CS}}$ of these slaves could not be recorded due to the limited amount of available FPGA pins.

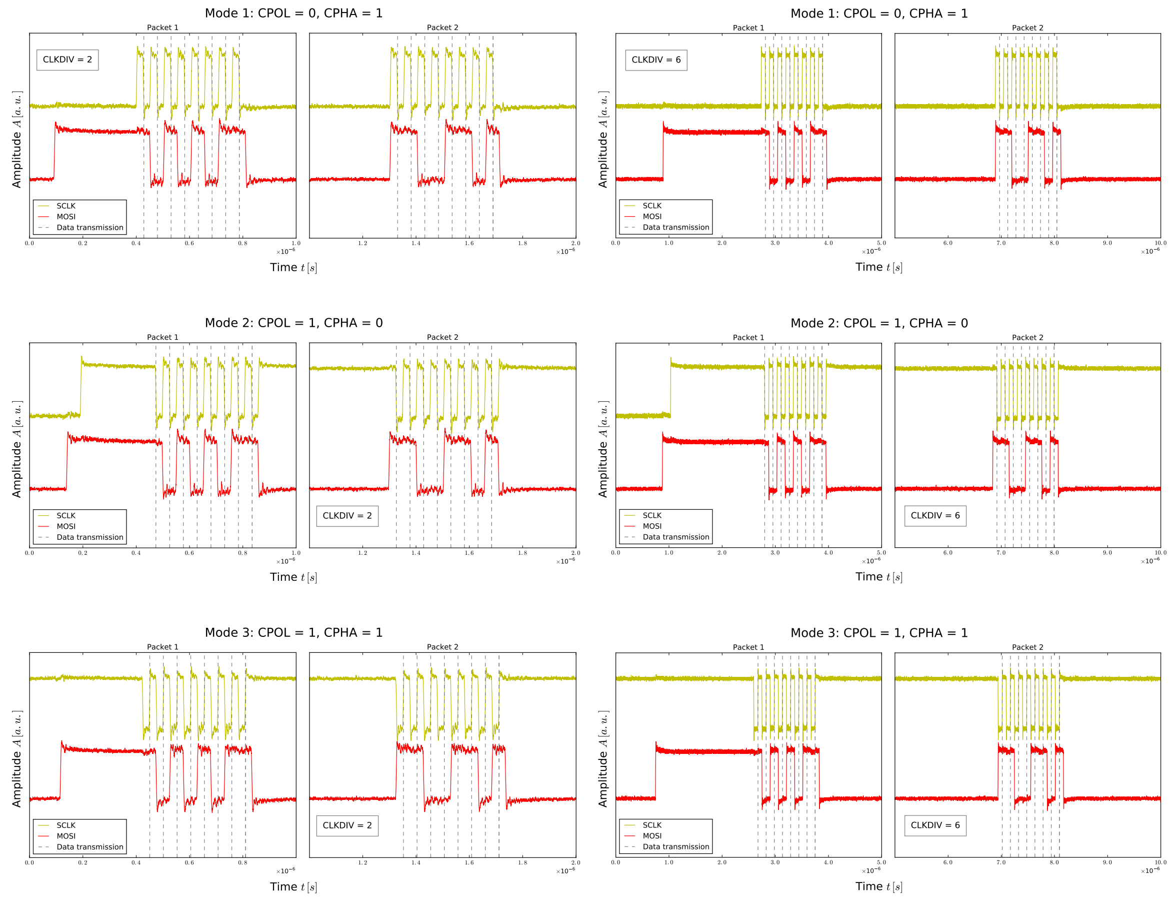


Figure 9: SPI test for modes 1, 2 and 3. For the plots on the left side, a clock dividing factor of 2 has been used whereas we have applied a factor of 6 to receive the output displayed by the ones on the right side.

2.4 Implementation: Timing Corrections with OCP-IO

When performing a simple `READ` transaction with the SPI connected to the SPI-to-I^{2}C-chip on the ANANAS board, timing constraints prohibit valid data to be read from the buffer of the chip as soon as the `READ` request from the host is passed to the slave. Figure 10 demonstrates this issue.

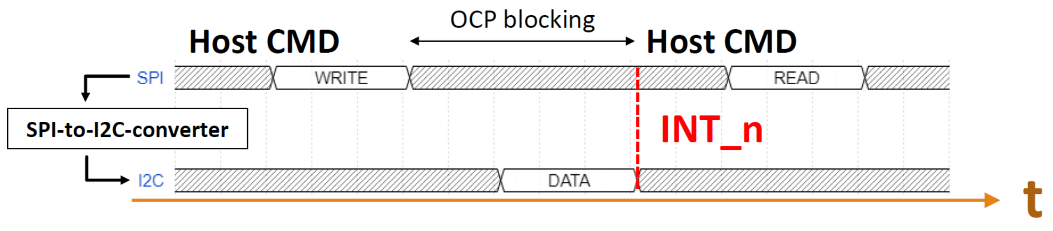


Figure 10: Timing diagram of a single `READ` transaction between the host and some slave device of the SPI-to-I^{2}C-chip. A time line has been included in order to roughly sketch the sequence in which the commands from the host are transmitted and the data from the slave is returned. First, the host writes the command to request a `READ` from the slave device via the OCP. The SPI begins the communication with the SPI-to-I^{2}C-chip. Because the data from the I^{2}C slave device is not directly returned and stored in the buffer of the SPI-to-I^{2}C-chip, an immediately issued `READ` request of the OCP would not lead to a readout of valid data. Therefore, one needs the OCP to block the commands of the host until it is verified that valid data is to be read from the buffer of the SPI-to-I^{2}C-chip. The $\overline{\text{INT}}$ of the chip can be used as a trigger for this verification.

The SPI-to-I^{2}C-chip however possesses an active low *Interrupt* ($\overline{\text{INT}}$) output. The $\overline{\text{INT}}$ is always pulled down when a data transmission between the SPI and the I^{2}C succeeded/failed or in case of a time-out. It is deactivated as soon as the internal status register of the SPI-to-I^{2}C-chip is read or as a result of a master reset. This means that under the assumption of an error free communication between the host and the SPI-to-I^{2}C-chip and its slave device respectively (every packet sent by the host would get accepted), valid data in the chip's buffer could be guaranteed as soon as the $\overline{\text{INT}}$ gets pulled down. The communication between the host and the slave device now has to be blocked after the last packet to be written has been sent until the $\overline{\text{INT}}$ goes down. The easiest way to achieve this, is to completely stop transactions on the OCP bus for this duration. In order to do so, one can use a feature of the OCP instances within other modules which is the *Variable Response Slave* (VS) and a module called `ocp_io`. The reason for this is that a VS is able to perform slave blocking, thus can prevent the OCP from performing further transactions. Additionally, the `ocp_io` is a register access for the OCP bus originally used to connect *Analog-Front-End* (AFE) and *Physical Layer* (PHY) resets on the ANANAS board. By instantiating a VS connection of the module to the OCP, one is in principle able to trigger a blocking behaviour by simply sending a dummy packet to `ocp_io`. Furthermore, it can also be connected to the $\overline{\text{INT}}$ of the SPI-to-I^{2}C-chip (via one of the pins of the FPGA) that can now be used as a trigger for disabling the VS blocking.

A block diagram of the `ocp_io` can be seen in figure 11. The input `in` is connected to the $\overline{\text{INT}}$ of the SPI-to-I^{2}C-chip. The output `out` is available for the AFE-PHY connection but is so far of no interest for our needs. There has an additional enable been included for the blocking behaviour (`blk_enable`). It becomes active when the lower twelve bits of the `ocp_io` address are set to `12'h111`, otherwise the OCP will not be put into a blocking state. Given a dummy packet written to the `ocp_io` which enables blocking, the OCP bus will only be released whenever the $\overline{\text{INT}}$ on `in` becomes active or when the time-out given by the maximum OCP clock cycles of the parameter `CNT_BLOCK` is reached. This is achieved by generating an accept. Here, 700000 clock cycles have been chosen as a default value considering a 40 MHz OCP clock and a 0.1 MHz SPI-to-I^{2}C-chip with a maximum transmit buffer of 96 bytes (taken from the datasheet of the *SC18IS600* [4]). This equals a counter width of 20 and is an acceptable usage of the available FPGA resources.

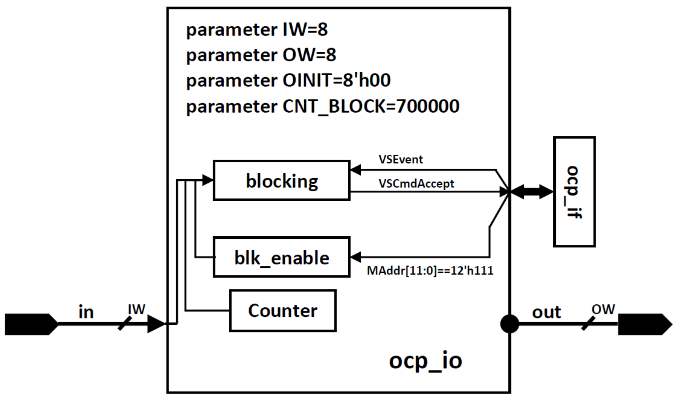


Figure 11: Block diagram of the `ocp_io` module. Input and output widths are controlled with the parameters `IW` and `OW`, respectively. The parameter `OINIT` sets the default value for the output in case of a reset. A time-out counter has been implemented to release the OCP bus after `CNT_BLOCK` cycles of the OCP clock. `blocking` can be enabled with the lower twelve bits of the `ocp_io` address and is activated whenever a `VSEvent` occurs on the OCP. When blocking is finished, a `VSCmdAccept` is generated.

The correct blocking behaviour could be verified on hardware by sending data via an Ethernet connection to the SPI-to-I^{2}C-chip. This allows for maximum capacity utilisation of the receiving FIFO of the FPGA thus reinforcing timing dependent errors when reading data received. Without blocking enabled, a request to read the board-ID of the ANANAS could not be processed correctly because the buffer of the SPI-to-I^{2}C-chip was read out too early. With blocking enabled, this issue could be fixed leading to a correct readout of the ID.

3 Conclusion and Outlook

Throughout this internship, the basics of FPGA based hardware design could be acquired by designing and verifying a SPI master module. The SPI was specified to allow for user defined $\overline{\text{CS}}$ to SCLK times, support of all four available modes concerning the active SCLK edge and flexible triggering frequencies for the adaption to individual requirements of various slave devices. The module could be verified with the analysis of the control and data signals using an oscilloscope. For the implementation on hardware, OCP blocking was introduced by making use of VS blocking within `ocp_io`, a module used for the connection of AFE and PHY resets on the ANANAS. With the use of an Ethernet connection, a high-speed test showed the correct treatment of the packets sent by the host.

The SPI master can now be used for the initial setup of the ANANAS board after a reset has been performed. Here, various power-on sequences and a read-out of the board- and the wafer-ID have to be performed via an I^{2}C connection. Currently, this is done with several milliseconds of waiting time after each data transmission between the host and different slave devices on the board in order to guarantee for valid data in the buffer of the SPI-to-I^{2}C-chip. Also, the frequency of the data transmission itself must be scaled down dramatically to fit the $\overline{\text{CS}}$ to SCLK specifications of the chip. With the VS blocking and the variable pre- and post-data waiting times of the SPI master, these issues can now be fixed easily.

Considering other issues to be fixed, all slave devices are provided with the same SCLK at the moment. The module can be improved further by allowing for adjustable SCLK frequencies for multiple slaves. Also, currently there is no configurable waiting time between two consecutive data packets. The data transmission state is always selected as soon as valid data is available from the host in case the output register of the SPI master to the OCP is empty. An additional user defined waiting time for this transition would allow for more flexibility of the module.

References

[1] Joscha Ilmberger. *Development of a digitizer for the BrainScaleS neuromorphic hardware system, master thesis by Joscha Ilmberger, Heidelberg Univerity, 23.06.2017*

[2] Xilinx. *Spartan-6 FPGA Clocking Resources User Guide (UG382), www.xilinx.com/support/documentation/user_guides/ug382.pdf, last access: 30.06.2018*

[3] Wikipedia. *Serial Peripheral Interface Bus, en.wikipedia.org/wiki/Serial_Peripheral_Interface, last access: 23.06.2018*

[4] NXP Semiconductors. *SC18IS600 data sheet, www.nxp.com/docs/en/data-sheet/SC18IS600.pdf, last access: 28.06.2018*