Points	Grade

Philipp Lehninger 1327039
Benedikt Morgenbesser 1027440

Digital Integrated Circuits Lab (LDIS)

384.088, Summer Term 2019

Supervisors:
Christian Krieg, David Radakovits, Axel Jantsch

Task 3: Documentation
Amba Thermometer Nexys 4 DDR

1 Introduction

The *Amba Thermometer Nexys 4 DDR* core reads in data from the temperature sensor on the Nexys 4 DDR board, processes this data in a moving average algorithm and ouputs the temperature on the LED display. The components of the core communicate via the AMBA ABP bus.
This documentation gives an overview of the system. All subcores are described in their respective chapters, where valuable information for the usage is given and potential problems are highlighted.

2 System Overview

A blockdiagram of the system is shown in figure 1. The core includes general AMBA ABP bus master and slave modules. The sub-components *SensorSlave*, *InputSlave*, *DSPSlave* and *OutputSlave* include an instance of the slave module. The top level design *thermometer* controls the bus via the master module.

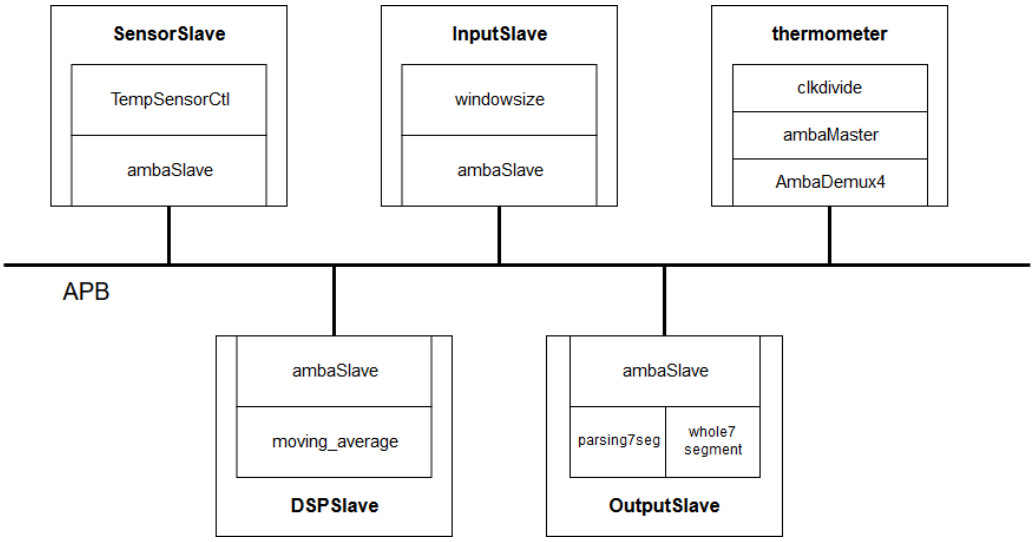

Figure 1: Block diagram of the thermometer cores with APB Bus communication

3 AMBA APB Bus

The communication via the bus is implemented according to the AMBA 3 ABP Protocol v1.0 Specification[1]. Please read this specification for further information.

3.1 ambaMaster

The *ambaMaster* module operates through the four following states.

1. **IDLE** Default state of the module. When a transfer is required the master moves to the SETUP state.
2. **SETUP** The master asserts the PSELx signal to select one slave. This state remains for only one clock cycle and is followed by the ACCESS state.
3. **ACCESS** The PENABLE signal is asserted and the read/write access is granted. A transition from this state is controlled by the PREADY signal with following possibilities:

   After a read access the next state is PROVIDE

   Else, if another transfer is required, the next state is SETUP

   If nothing of the above applies the next state is IDLE
4. **PROVIDE** For a read access the buffered data is provided. Depending on whether there is another transfer requested the follwing state is either IDLE or again SETUP.

[1] https://static.docs.arm.com/ihi0024/b/AMBA3apb.pdf

The interface of the *ambaMaster* module is shown in figure 2. The interface to the APB bus is according to the AMBA 3 ABP Protocol v1.0 Specification.[2] For information about the signals please read the descriptions in this specification. The interface for the communication with the top level design includes the following signals:

1. **data** Is a bi-directional 32 bit vector for the communication between top level design und bus master. Please notice: If any changes affecting the data signal are applied, make sure to keep consistent with the tristate logic.
2. **slaveaddr** A slave is addressed by asserting the respective place in the slave address vector.
3. **regaddr** Is euqivalent to PADDR and not used in this core. It would be only necessary to address registers, if the data exceeds the 32 bit width of the ABP bus.
4. **transferRequest** Indicates if a transfer is requested.
5. **write** Indicates an write access if high and a read access if low.
6. **ready** Indicates if the master module is ready to start a new access if high.

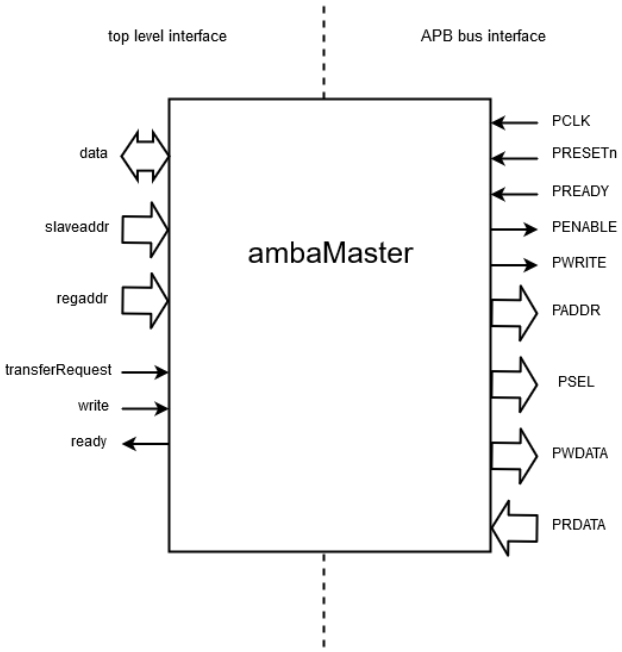

Figure 2: Interface of the ambaMaster module.

3.2 ambaSlave

The ambaSlave module generates its current state from the received signals. The interface of the *ambaSlave* module is shown in figure 3.The interfaceto the APB bus is again according to the specification. The interface to the slave modules includes the following signals.

1. **data_in** Is a 32 bit vector for the data to be read by the master.
2. **data_out** Is a 32 bit vector for the data written by the master. Please notice: In contrast to the master module the communication for read/write between the slave module and their sub-components is seperated, i.e. an uni-directional bit vector is used for each operation.
3. **addr** Is euqivalent to PADDR and not used in this core. It would be only necessary to address registers, if the data exceeds the 32 bit width of the ABP bus.
4. **readreg** Indicates a read access if high.
5. **writereg** Indicates a write access if high.
6. **done** Indicates if the slave finished its operations and a new transfer can be requested.

[2] https://static.docs.arm.com/ihi0024/b/AMBA3apb.pdf

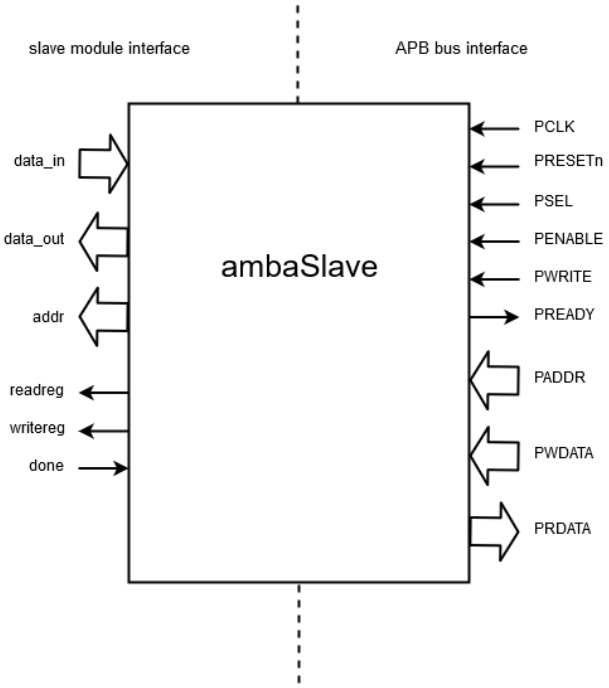

Figure 3: Interface of the ambaSlave module.

4 thermometer

Thermometer is the top level entity and controls the bus master via a state machine with the 6 following sequential states:

1. **IDLE:** This state is just used as an initial state for the system after power-up or resets.
2. **INPUT:** The user input for the size of the averaging window is read from the Input Module.
3. **SENSOR:** The 16 bit 2's complement vector from the Sampling Module is read.
4. **FEEDMAVG:** The temperature value and window size are concatenated and written to the Moving Average Module.
5. **MAVG:** The averaged temperature is read.
6. **OUTPUT:** The averaged temperature vector is written to the Output Module. This state is followed by the INPUT state.

A state transition only occurs when the *ready* flag is set by the bus master (Except for the IDLE state).

4.1 clkdivide

All clocks are generated in the thermometer module.

1. The ABP bus clock is set to 25 MHz.
2. The display clock is set to 1 kHz. (For a cycle of 8 digits, this gives a refresh frequency of 125 Hz for each digit).
3. The sample rate is set to 250 ms per default. This can be adjusted presynthesis according to table 1

sampling rate value	corresponding sample time
1	0.25 sec
2	0.5 sec
4	1 sec
3	not valid

Table 1: Sampling time.

4.2 AmbaDemux4

All slaves have to share the PRDATA and PREADY port of the *ambaMaster* module. In order to read from the slaves the *AmbaDemux4* module demultiplexes these signals.

5 SensorSlave

The *SensorSlave* module reads data from the *temperature_2comp* register with a sampling rate of 250 ms, 500 ms or 1 s. The sampling rate can be set presynthesis in the *thermometer* module.

5.1 TempSensorCtl

The *TempSensorCtl* module is from the Nexsys Demo Version and intellectual property of Elod Gyorgy[3]. The module reads continiously from the ADT7420 temperature sensor on the board and writes the data to the *temperature_2comp* register. The interface between the sensor and the FPGA is shown in figure 4.

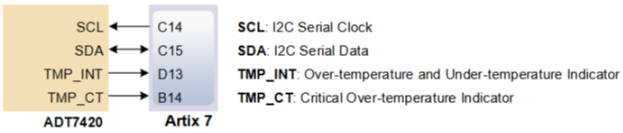

Figure 4: Temperature sensor interface.

For the correct implementation of the *TempSensorCtl* module in this core the resolution lenght of the temperature vector was adapted to 16 bit. Therefore, the configuration register (address 0x03) was set to 0x80.

6 InputSlave

The *InputSlave* connects the up and down button (BTN_U & BTN_D) from the Nexsys 4 DDR board to adjust the window size of the moving average algorithm during run time.

6.1 windowsize

The *windowsize* module permits four different window sizes, i.e. 1 x / 2 x / 4 x / 8 x the sampling period.

7 DSPSlave

The *DSPSlave* module reads from the master (bits 0-15: temperature data in 2's complement, bits 16-17: windowsize) ,processes it, and writes the averaged temperature back.

7.1 moving_average

The *moving_average* module is averaging the temperature according to the window size. For each new writing access, the temperature registers are shifted and the new value is safed in *temp_in_ex0*. To keep the 2's complement addition consistent the MSB is copied to three addtional bits. The division of the summed value is done by shifting. Please notice: Therefore only windowsizes of 2^n x sampling rates are feasable.

8 OutputSlave

8.1 parsing7seg

To display the temperature the 2's complement std_logic_vector is converted into an array, where each component represents one digit of the display. One LSB of the temperature sensor is equivalent to 0.0078 °C. The module first sets a signbit. Then the 2's complement vector is converted to an integer and multiplied by 78. This value represents now the temperature · 1E4. To get the individual digits, the value is divided by the highest decimal power and saved as an integer into the array. This integer times the decimal power is substracted from the original value to get rid of the digit and the process is repeated for the next smaller decimal power...

[3] https://github.com/Digilent/Nexys-Video-OLED/releases

8.2 whole7segment

The eight digits of the display have a common anode and individual cathodes to drive the seven segments. (See figure 5.) The digits are addressed periodically.

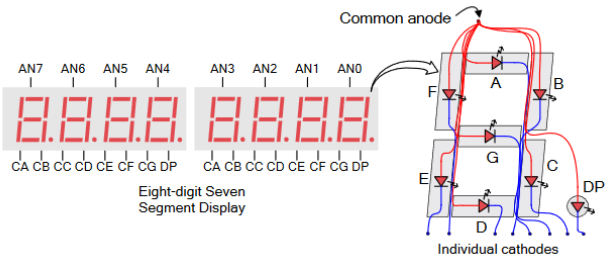

Figure 5: Eight-digit Seven Segment Display