label12table with the valuestable.caption.2 label@cref[table][1][]12 label22visualization of tablefigure.caption.3 label@cref[figure][2][]22

Points	Grade

Team: XX

1525864. Glinserer Andreas
...

Digital Integrated Circuits Lab (LDIS)

384.088, Summer Term 2019

Supervisors:
Christian Krieg, David Radakovits, Axel Jantsch

Task 2:

Design Characterization, Bus Communication

1 Characterize your design from Task 1

In order to make your design implementation comparable to other implementations, simulate your design using Vivado and perform the following measurements:

1. Timing analysis
2. Power analysis
3. Resource consumption

Create a design space vector for your design.The design space vector holds the following parameters, where t_{max} is the maximum delay for the critical path, P_{avg} is the average power consumption for your design (vector-less post-place-and-route power estimation), and r is the percentage of resources used in your implementation for the Nexys 4 DDR board (for sake of simplicity, use the percentage of slices):

$$\vec{v} = \begin{bmatrix} t_{max} \\ P_{avg} \\ r \end{bmatrix} \tag{1}$$

You will find information on power estimation[1] and timing analysis[2] on the web.

Get the design space vectors of your colleagues, and visualize the three-dimensional design space. Use black crosses for the vectors of your colleagues, and a filled circle for your own vector.

2 Create an AMBA APB interface

The goal of this subtask is to make your design accessible from other intellectual property (IP) cores via the advanced microcontroller bus architecture (AMBA) advanced peripheral bus (APB). Consult the APB specification,[3] and:

1. Implement a bus controller that implements the use case for your task
2. Implement a bus interface for any sub-component of your design from Task 1 (sampling, data processing, output).
3. Implement a module that takes the user input for runtime parameters and connect it to the bus

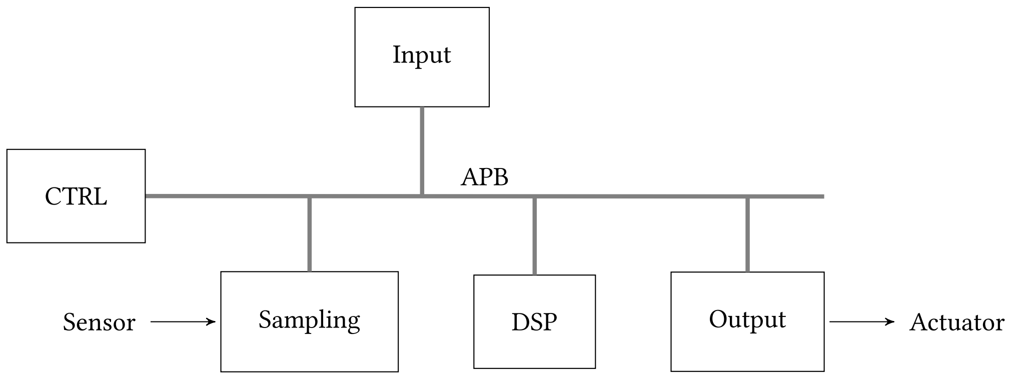


Figure 1: System of Task 1 equipped with an APB

Your system should be able to take the user input, parametrize the system according to the user input, and perform the actual task. Instead of directly connecting each of the sub-components directly, the modules should communicate over the APB. The audio system can be treated as one module; as it is kind of a real-time system, it wouldn't make a lot of sense to let it communicate over the APB. The user-input (e.g., cut-off freuqency) should be set over the APB.

[1] *Vivado Design Hub - Power Estimation & Optimization.* URL: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0008-vivado-power-estimation-and-optimization-hub.html.

[2] *Vivado Design Hub - Timing Closure & Design Analysis.* URL: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0006-vivado-design-analysis-and-timing-closure-hub.html.

[3] *AMBA 3 APB Protocol Specification.* Tech. rep. ARM Ltd., Aug. 17, 2004. URL: http://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf.

3 Characterization

Group	t_{max}[ns]	P_{avg}[mW]	r[%]
Philipp, Benedikt	5,44	88,00	6,60
Kratzmann	5,03	185,00	13,12
Essbüchl Jakob	37,62	140	1,36
Glinserer Andreas	5,89	110	1,54
Hauk Raphael	8,08	111	3,9
Philipp W.	2,557	121	0,477

Table 1: table with the values

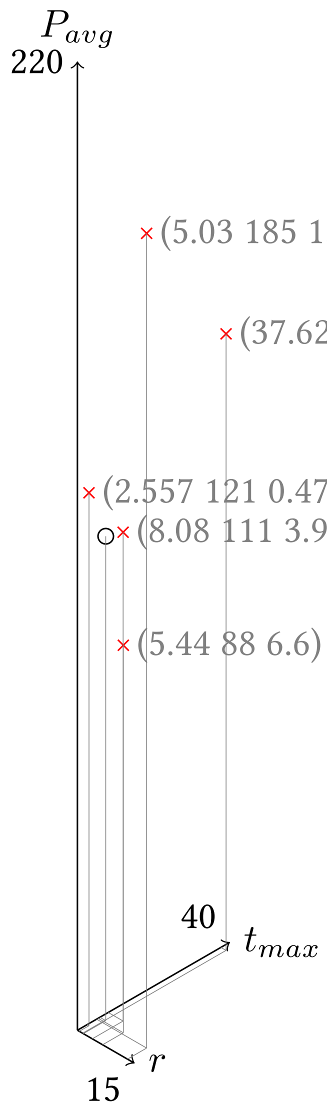


Figure 2: visualization of table

4 Implementation

My implementation is as follows: I have additional registers in front of every instantiated component. The APB bus then reads and writes into/from these registers. Due the lack of documentation I implemented the PREADY signal with a tristates. It would also be possible to implement this as a wire for each module on its own.

4.1 Master CTRL

The master is amplemented with 2 state machines. Every state machine is implemented with 2 processes. 1 process which handles the outputs and another process which handles the evaluation of the next state. The state change happens in a synchronous process. 1 of the state machines handles the APB Protocol and Data exchange on the line. It is built after the state machine (fig.3) in the AMBA protocol pdf.

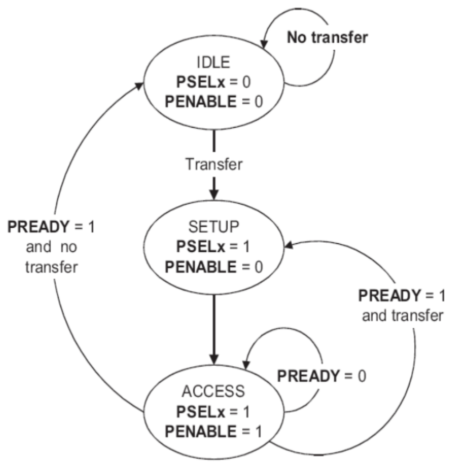


Figure 3: APB state diagram. Source:[4]

The second state machine handles the program procedure (fig.4). The procedure is as follows: Every 100 ms it checks if the switches changed. If a change happened then the new size gets transmitted to the DSP. Every $\frac{1}{\text{samples}}$ s it reads the Sampling part, writes this data then to the DSP, reads the DSP and then transmits the output from the DSP to the output.

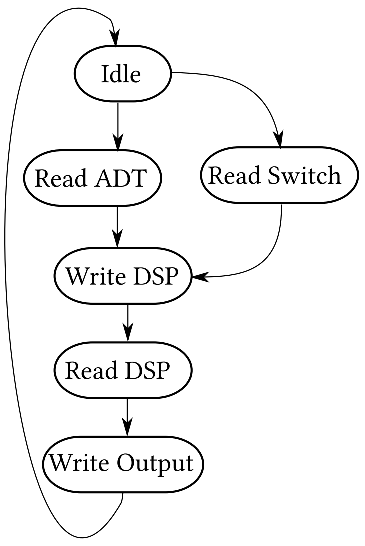


Figure 4: master state diagram. Source: Drawn in Inkscape

4.2 APB slaves

The slaves were implemented with the goal to leave the working parts untouched. This goal was reached by implementing the slaves as registers. The master writes to the registers and the slave component reads from the register. Therefore every slave looks the same with slight modifications:

```
enable_write <= in_PENABLE and in_PWRITE and in_PSELx;
enable_read  <= not in_PWRITE and in_PSELx;     -- data is read everytime to be ready on the first cycle

out_PREADY <= int_pready;

```

```
-- register with the enable sigs as clock
WRITE: process(in_PRESETn, enable_write)
begin
        if in_PRESETn ='1' then
                int_data_write <= x"0000";

        elsif rising_edge(enable_write) then
                int_data_write <= in_PWDATA;
        end if;
end process WRITE;

-- register with the enable sigs as clock
READ: process(in_PRESETn, enable_read)
begin
        if in_PRESETn ='1' then
                out_PRDATA <= x"0000";

        elsif rising_edge(enable_read) then
                out_PRDATA <= int_data_read;

        end if;
end process WRITE;

PREADYP: process(enable_write, enable_read)
begin
        if enable_write='1' or enable_read='1' then
                int_PREADY <= '1';
        else
                int_PREADY <= '0';
        end if;
end process PREADYP;

-- process to forward the data to whatever module or to write into the read register
SFORWARD: process(in_PRESETn, in_PCLK, enable_write, in_PADDR)
begin

        if rising_edge(in_PCLK) then
                if in_PRESETn = '0' then
                        int_data_read <= x"0000";
                else
                        -- here comes handling of signals to write into the read register

                end if; -- rst
        end if; -- rising clock
end process SFORWARD;
```

For every slave a new .vhd file was created with the template above in it and an instantiation of the component it is the interface to. The ADT slave samples values and if the CTRL requests values faster it just gets the old value back. If you write to the DSP part, the DSP holds the bus transmission by holding the PREADY signal low until the new value is computed.

4.3 LEDs

LD15	LD14	LD13	LD12	LD11	LD10	LD09	LD08
Buffer size of the DSP binary encoded							

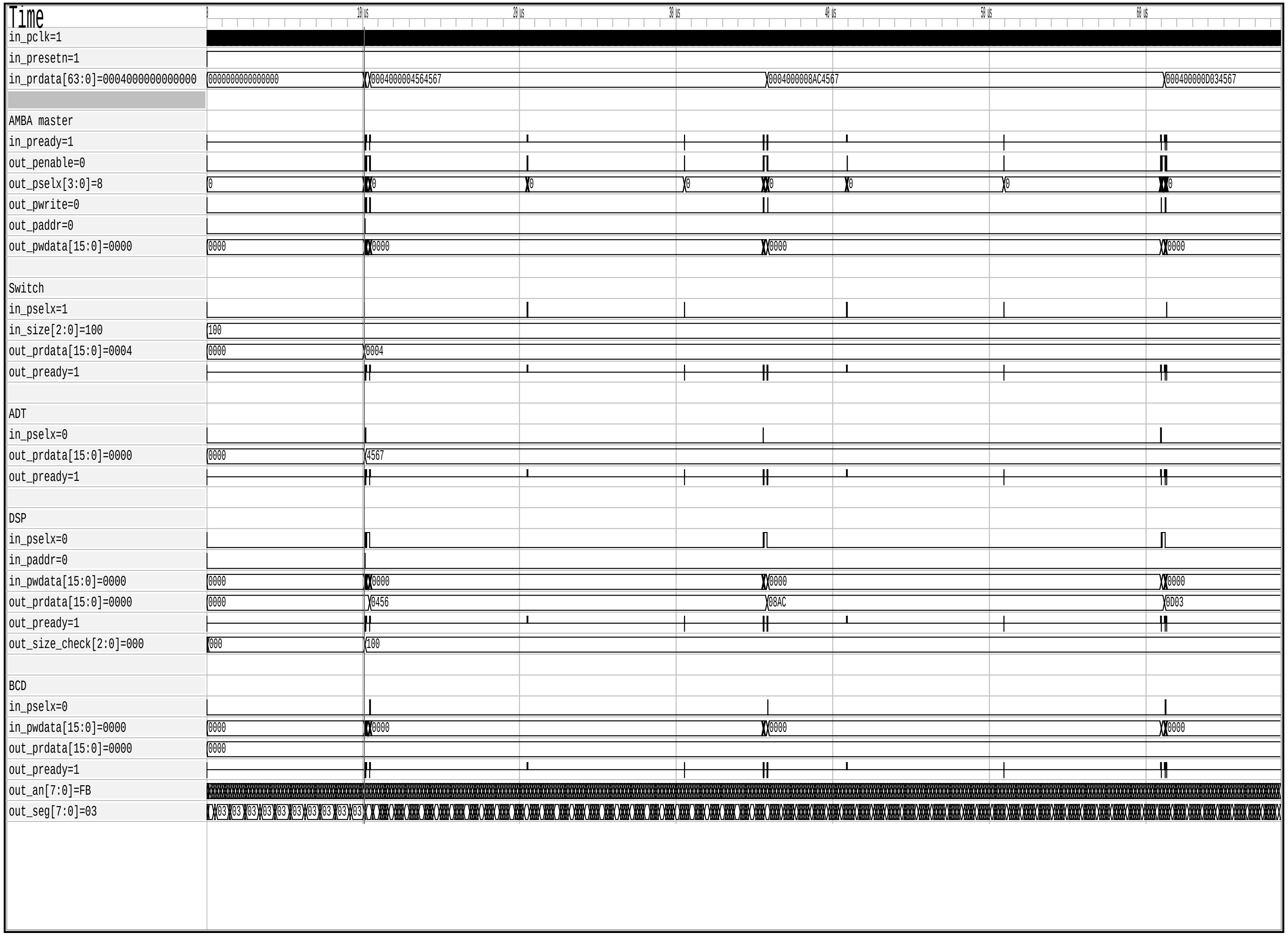
LD07	LD06	LD05	LD04	LD03	LD02	LD01	LD00
ADT	CTRL size			STATE	DSP size		

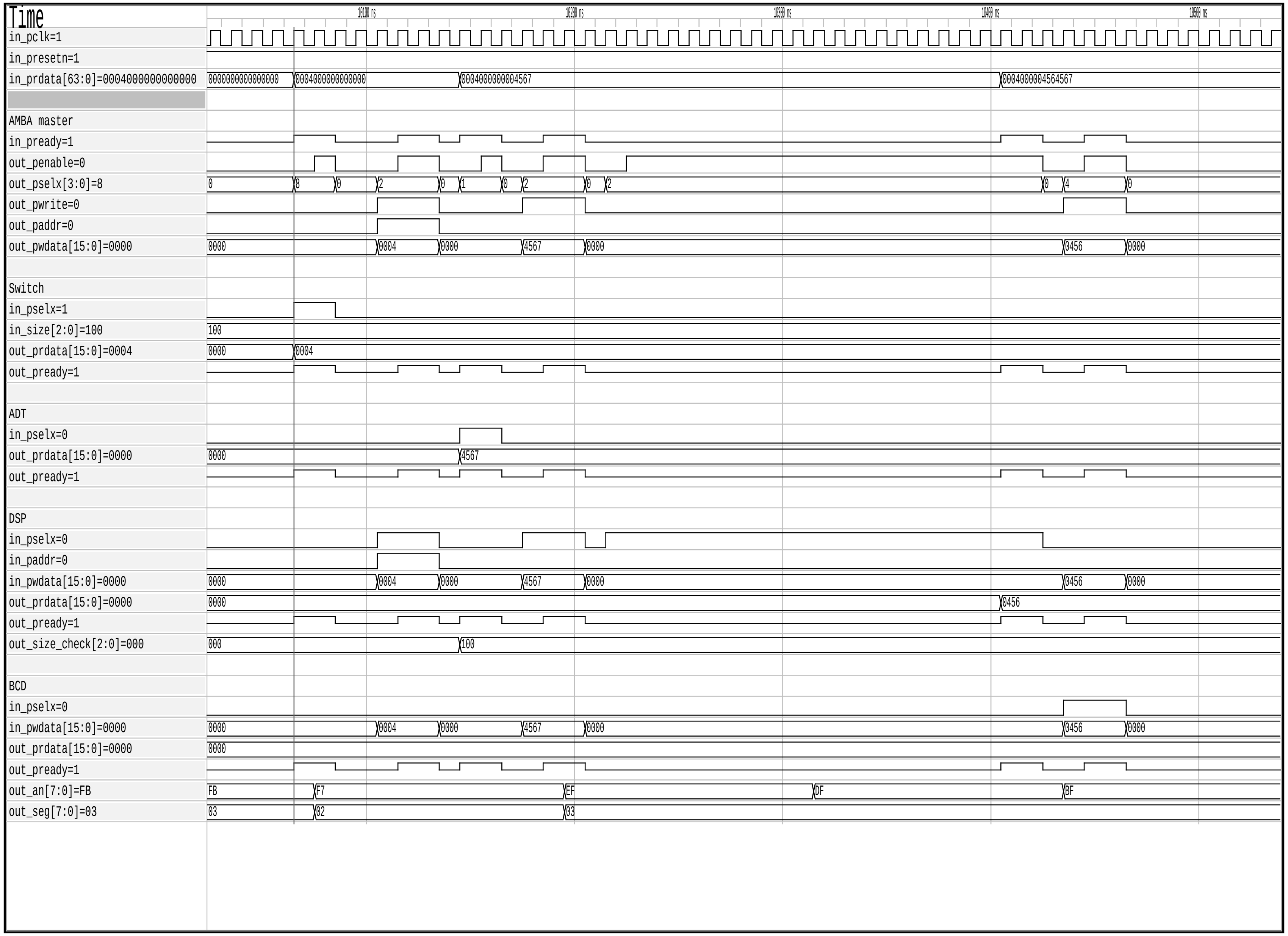
Table 2: LED explanation

- ADT - signals everytime the ADT samples a new value
- CTRL size - the size saved in CTRL part as power of two. E.g. b’110 $= 6- > 2^6 = 64$ buffer size.
- STATE - toggles everytime the CTRL goes into the idle state
- DSP size - same as CTRL size but saved in the DSP part

4.4 Simulation

Here are two images from the simultion. The first one shows the way the master works in respect to polling the switch slave. The switch slave select wire is choosen more often. And corresponding to the countersize it starts a normal cycle. The normal cycle can be seen in the second simulation image.


Time
0
10 us
20 us
30 us
40 us
50 us
60 us
in_pclk=1
in_presetn=1
in_prdata[63:0]=0004000000000000
0000000000000000
0004000004564567
0004000008AC4567
000400000D034567
AMBA master
in_pready=1
out_penable=0
out_pselx[3:0]=8
0
out_pwrite=0
out_paddr=0
out_pwdata[15:0]=0000
0000
Switch
in_pselx=1
in_size[2:0]=100
100
out_prdata[15:0]=0004
0000
0004
out_pready=1
ADT
in_pselx=0
out_prdata[15:0]=0000
0000
4567
out_pready=1
DSP
in_pselx=0
in_paddr=0
in_pwdata[15:0]=0000
0000
out_prdata[15:0]=0000
0000
0456
08AC
0D03
out_pready=1
out_size_check[2:0]=000
000
100
BCD
in_pselx=0
in_pwdata[15:0]=0000
0000
out_prdata[15:0]=0000
0000
out_pready=1
out_an[7:0]=FB
out_seg[7:0]=03
03


Time
10100 ns
10200 ns
10300 ns
10400 ns
10500 ns
in_pclk=1
in_presetn=1
in_prdata[63:0]=0004000000000000
0000000000000000
0004000000000000
0004000000004567
0004000004564567
AMBA master
in_pready=1
out_penable=0
out_pselx[3:0]=8
0
8
0
2
0
1
0
2
0
2
0
4
0
out_pwrite=0
out_paddr=0
out_pwdata[15:0]=0000
0000
0004
0000
4567
0000
0456
0000
Switch
in_pselx=1
in_size[2:0]=100
100
out_prdata[15:0]=0004
0000
0004
out_pready=1
ADT
in_pselx=0
out_prdata[15:0]=0000
0000
4567
out_pready=1
DSP
in_pselx=0
in_paddr=0
in_pwdata[15:0]=0000
0000
0004
0000
4567
0000
0456
0000
out_prdata[15:0]=0000
0000
0456
out_pready=1
out_size_check[2:0]=000
000
100
BCD
in_pselx=0
in_pwdata[15:0]=0000
0000
0004
0000
4567
0000
0456
0000
out_prdata[15:0]=0000
0000
out_pready=1
out_an[7:0]=FB
FB
F7
EF
DF
BF
out_seg[7:0]=03
03
02
03

4.5 Problems

I had many problems with my first implementation because I misunderstood a central point in the AMBA APB specification. Thanks to this mistake I had race conditions in my design which caused my simulation to work but my implementation to only work sometimes. Much time was wasted by trying it this way. To clarify my mistake: I tried not to write into registers but to directly write into the components. By doing it this way I got many additional conditions to wait for values and multiple sources for enable signals. Somewhere in between then I got the race condition.

4.6 Reports

For the reports look into the generated files in the report directory. It was not possible to include them in the report because of undefined latex errors.

4.6.1 timing report

In the timing report you can see many pins which are not driven by a clock pin. This is per design, because APB can react asynchronously to operations. Waiting for the rising clock edge everytime would always add a wait state.