Prof. Dr.-Ing. Dr. h.c. J. Becker

# Digital Hardware Design Laboratory (DHL)

**Exercise 3 – State Machine with BRAM**

Institut für Technik der Informationsverarbeitung, Karlsruher Institut für Technologie (KIT)

## 1 Introduction

In this exercise a storage game needs is to be implemented. Here, the user can store desired information in a memory in a specific location. Afterwards the stored data sets can be checked and the result is visualized by a fast (success) or slowly (wrong input) flashing LED. The game models a cache file, a processor can store data to and which needs to be checked in order to determine whether the datum is present inside the memory.

The core of this exercise is a final state machine for directing the control flow of the automat. Secondly, since cache files can be very large, a Block RAM (BRAM) needs to be used in order to reduce the storage's complexity. A BRAM is a so-called hard-IP which is present (in limited number) as dedicated slices in specific columns of the FPGA fabric.

## 2 Preparation

Prior to the laboratory afternoon for this exercise you should get familiar with the following topics:

- Finite State Machines and how they can be modelled in VHDL
  - See also additional material "VHDL State Machine Example"
- Basics on how to use BRAMs on Xilinx FPGAs
  - Which types of BRAM are available on Xilinx FPGAs?
  - How can BRAMs be instantiated in VHDL designs?
  - See additional material „ug901 Vivado Synthesis Guide":
    - *Appendix A - RAM_STYLE Attribute*

        - *Appendix C - RAM HDL Coding Guidelines*

# 3 Task description

The automat should behave in the following way:

**States**:

- There need to be four different states for
    - **Doing nothing** (“*idle*”)
    - **Storing** information (“*store_information*”; mode ‘1’)
    - Check input (mode ‘0’)
        - **Check input** information (“*check_input*”)
        - **Output** the result of the comparison (“*output_result*”)
- In *“idle”* state the **mode switch** determines the next state
- In *“store_information”* and *“check_input”* the mode switch can interrupt the operation → idle state
- In *“store_information”,* pressing the **enter signal** stores information “led_pin” to the BRAM. Afterwards a transition to the idle states occurs. The current content of the selected BRAM is displayed by the LED_pin signal.
- In *“check_input”,* by **pressing enter**, the information is *buffered* and a transition to the *“output_result”* state is triggered. The current input is displayed at **“led_pin”.** The comparison is not done in this state because of the readout delay of the BRAM of one clock cycle.
- In *“output_result”* the buffered signal is compared with the output from the BRAM (**only once** when entering the state). The **result of the comparison is visualized** by the flashing LED led_status. The current input is displayed at **“led_pin”**. **Pressing enter** causes a transition to the idle states and stops the result visualization

**Inputs**:

- The debouncing and one-clock-cycle conversion from exercise one needs to be used in order to determin a stable signal for “ButtonC”
- The inputs need to be accessed (and potentially buffered) **by synchronous processes only**
- **Switches**:
    - Switch 7: Select mode to ‘1’ or ‘0’
    - SW6 and SW5: Control address which accesses the BRAM
    - SW3 downto SW0: Input information
- **Push button:**
    - the cleaned BTNC signal is used as an “enter” signal

**LEDs**:

- A **fast** and a **slow** flasher needs to be used
    - On Success: “led_status” needs to flash fast
    - On “Wrong input”: “led_status” needs to flash slowly
    - “led_status” is high in the “Store_information” state
    - Else the LED is turned off
- **led_pin signal** (3 downto 0): Output of information from input or BRAM data output

**BRAM:**

- Use a reasonable depth of the BRAM

- Make the BRAM 5 bits wide. Initialize the entries to “11111”. Stored signals have the format “ ’0’, SW3, SW2, SW1, SW0”. On reads the 5-bit output is compared with the input signal a ‘0’ is attached to in front. Like this, the BRAM entries will be initiated to values which will never occur as input data and thus no comparison to the initially “empty” BRAM will cause the result “success”.

# 4 Subtasks

## 4.1 Flasher component and BRAM implementation

You need to extend the flasher component in order to be able to set the flashing rate of the LEDs by modifying the input parameters (generics). Use a factor of eight between the fast a slowly flashing LED’s toggling rate.

The BRAM should be implemented as separate entity. Complete the BRAM architecture given in the code templates to get a working BRAM module.

## 4.2 Creation of the state machine

Visualize the state machine using pen and paper and add the state transition conditions and the values of the output signal changes.

## 4.3 Generation of the Testbench stimuli

Apply the knowledge gained from the last exercise in order to create the Testbench which is able to test for the correct operation of the final statemachine.

Derive the required information which needs to be input to the state machine for the following operation sequence – in the following description all signals are of the format (x downto 0):

1. Store information “0001” to address “00”
2. Trigger a check of input “0000” to address “00” which fails
3. Trigger a check of input “0001” to address “00” which succeeds
4. Trigger a check of input “0001” to address “10” which fails

## 4.4 Test on the Hardware

Synthesize your design and test it on the ZEDBOARD hardware. Does everything work as expected?