lab 05

PicoBlaze KCPSM6

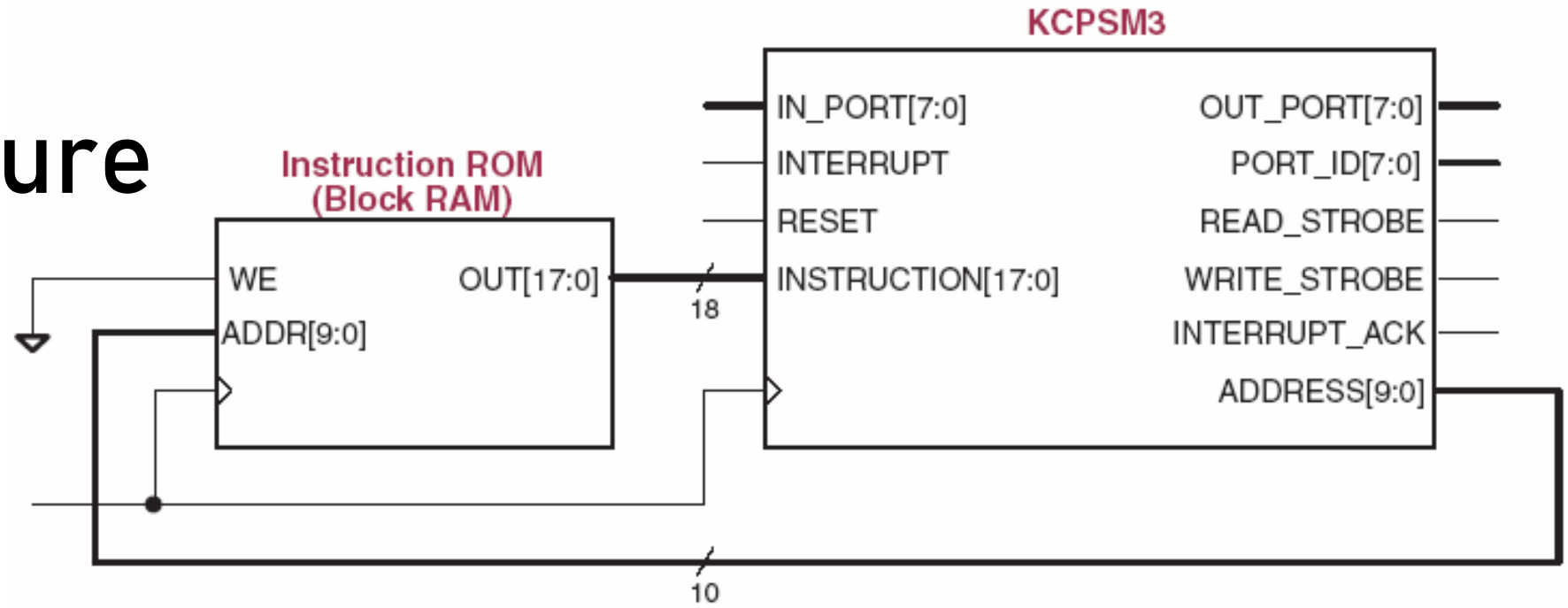
Digital design – laboratory exercises

assistant: Nejc Ilc

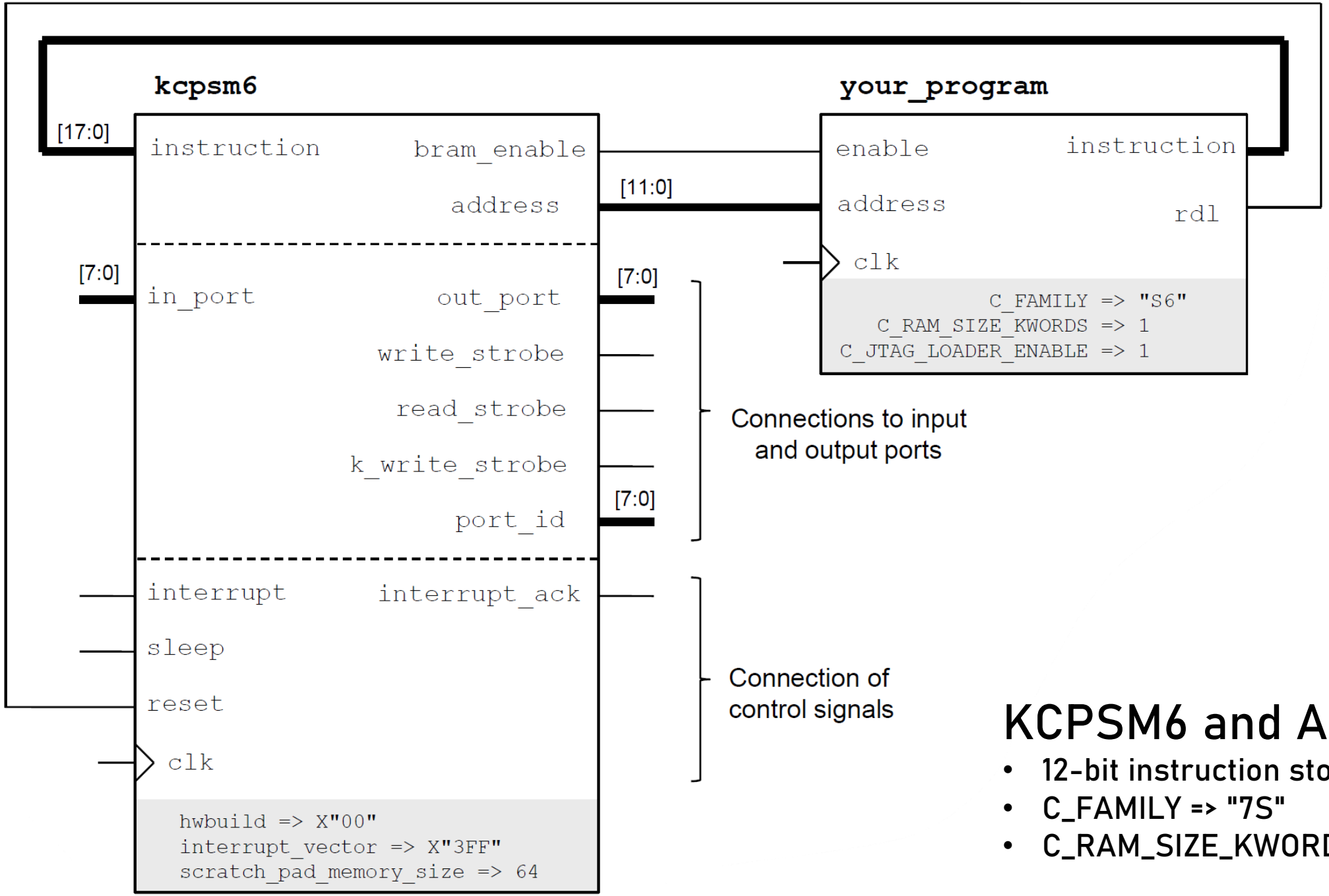
Introduction

- KCPSM
  - Ken Chapman's PSM
  - Konstant Coded Programmable State Machine
- Simple microcontroller (soft processor) totally embedded within Xilinx FPGAs
  - architecture: 8-bit RISC
  - performance: 52-119 MIPS
  - resource-friendly – it takes 0.3-5 % of FPGA resources (26 slices)
  - free sources, royalty-free re-use within Xilinx FPGAs
- Typical use case: non-timing cruical complex control functions (state machines), easy programming in assembly

Architecture



- Implemented using two modules: a microcontroller and an instruction store (ROM)
  - 18-bit instructions,
  - instruction store contains up to 4 K instructions (up to 2 K on Artix-7),
  - instruction store is implemented using block RAM" (BRAM). Signal WE is put to 0, which makes it ROM.
  - The upper figure presents KCPSM3, which features 10-bit wide addresses.



KCPSM6 and Artix-7

- 12-bit instruction store address
- C_FAMILY => "7S"
- C_RAM_SIZE_KWORDS => 2

Architecture

- 18-bit instructions
  - two clock cycles per instruction (RISC)
  - ALU operations affect the Zero (Z) in Carry (C) flags
- 2 banks of 16 general purpose registers, 8 bits wide
  - s0-sF
  - two banks for rapid switching between unrelated tasks (servicing interrupts)
- (64/128/256)-byte scratchpad RAM
- Input/output
  - 8-bit address of I/O port PORT_ID (256 possible inputs and outputs)
  - 8-bit data that is read from the input IN_PORT or is writen to the output OUT_PORT
- Interrupts
  - an input for interrupts (INTERRUPT)
  - an output for interrupt acknowledging (INTERRUPT_ACK)
  - interrupt response time in 3-4 clock cycles

Instruction set

- input/output: INPUT, OUTPUT
- registers: LOAD
- arithmetic/logic: ADD, SUB, AND, OR, XOR, RL, RR, SL, SR,
- comparison: COMPARE, TEST
- scratchpad RAM: FETCH, STORE
- register banks: REGBANK A, REGBANK B, STAR
- jumps, calls: JUMP, CALL, RETURN
- Complete instruction set: see next slide or https://docs.xilinx.com/v/u/en-US/ug129, page 15, Table 3-1

KCPSM6 Instruction Set

```
aaa : 12-bit address 000 to FFF
 kk : 8-bit constant 00 to FF
 pp : 8-bit port ID 00 to FF
  p : 4-bit port ID 0 to F
 ss : 8-bit scratch pad location 00 to FF
  x : Register within bank s0 to sF
  y : Register within bank s0 to sF
```

Page	Opcode	Instruction
	Register loading	
55	00xy0	LOAD sX, sY
55	01xkk	LOAD sX, kk
71	16xy0	STAR sX, sY
	Logical	
56	02xy0	AND sX, sY
56	03xkk	AND sX, kk
57	04xy0	OR sX, sY
57	05xkk	OR sX, kk
58	06xy0	XOR sX, sY
58	07xkk	XOR sX, kk
	Arithmetic	
59	10xy0	ADD sX, sY
59	11xkk	ADD sX, kk
60	12xy0	ADDCY sX, sY
60	13xkk	ADDCY sX, kk
61	18xy0	SUB sX, sY
61	19xkk	SUB sX, kk
62	1Axy0	SUBCY sX, sY
62	1Bxkk	SUBCY sX, kk
	Test and Compare	
63	0Cxy0	TEST sX, sY
63	0Dxkk	TEST sX, kk
64	0Exy0	TESTCY sX, sY
64	0Fxkk	TESTCY sX, kk
65	1Cxy0	COMPARE sX, sY
65	1Dxkk	COMPARE sX, kk
66	1Exy0	COMPARECY sX, sY
66	1Fxkk	COMPARECY sX, kk

Page	Opcode	Instruction
	Shift and Rotate	
67	14x06	SL0 sX
67	14x07	SL1 sX
67	14x04	SLX sX
67	14x00	SLA sX
67	14x02	RL sX
68	14x0E	SR0 sX
68	14x0F	SR1 sX
68	14x0A	SRX sX
68	14x08	SRA sX
68	14x0C	RR sX
	Register Bank Selection	
70	37000	REGBANK A
70	37001	REGBANK B
	Input and Output	
73	08xy0	INPUT sX, (sY)
73	09xpp	INPUT sX, pp
74	2Cxy0	OUTPUT sX,(sY)
74	2Dxpp	OUTPUT sX, pp
78	2Bkkp	OUTPUTK kk, p
	Scratch Pad Memory	
	(64, 128 or 256 bytes)	
81	2Exy0	STORE sX,(sY)
81	2Fxss	STORE sX, ss
82	0Axy0	FETCH sX, (sY)
82	0Bxss	FETCH sX, ss

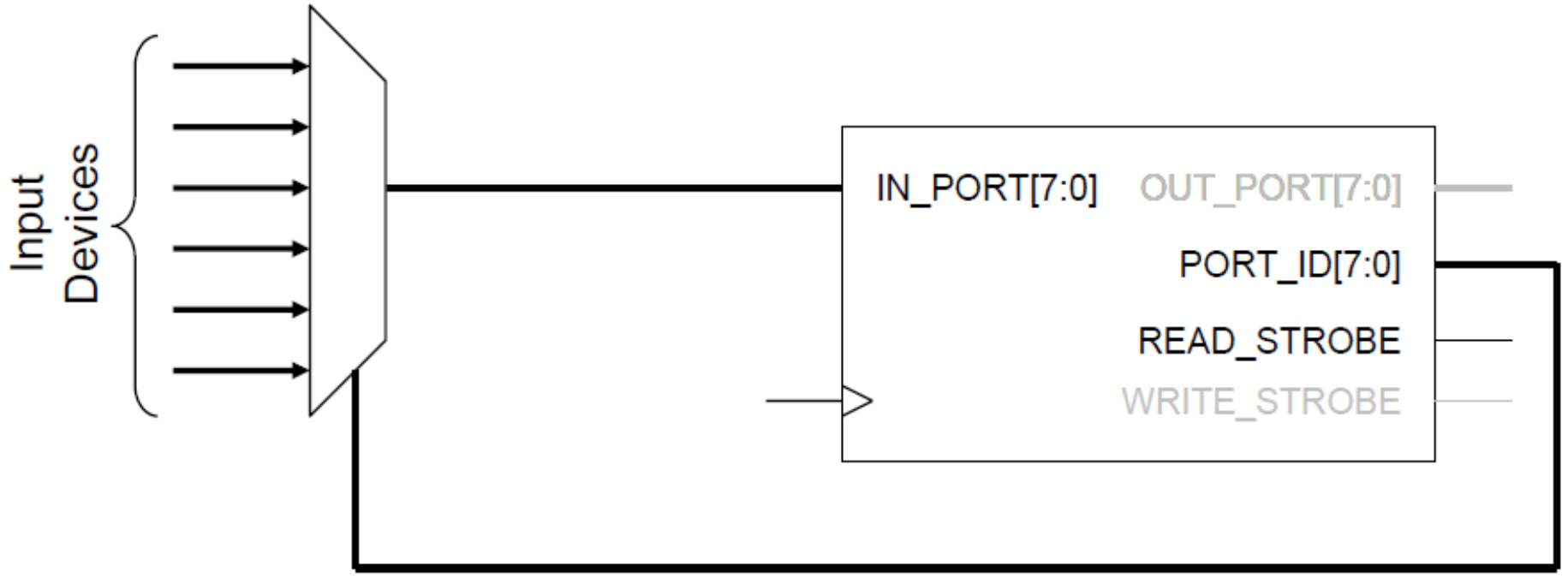
Page	Opcode	Instruction
	Interrupt Handling	
83	28000	DISABLE INTERRUPT
83	28001	ENABLE INTERRUPT
84	29000	RETURNI DISABLE
84	29001	RETURNI ENABLE
	Jump	
87	22aaa	JUMP aaa
88	32aaa	JUMP Z, aaa
88	36aaa	JUMP NZ, aaa
88	3Aaaa	JUMP C, aaa
88	3Eaaa	JUMP NC, aaa
89	26xy0	JUMP@ (sX, sY)
	Subroutines	
92	20aaa	CALL aaa
93	30aaa	CALL Z, aaa
93	34aaa	CALL NZ, aaa
93	38aaa	CALL C, aaa
93	3Caaa	CALL NC, aaa
94	24xy0	CALL@ (sX, sY)
96	25000	RETURN
97	31000	RETURN Z
97	35000	RETURN NZ
97	39000	RETURN C
97	3D000	RETURN NC
98	21xkk	LOAD&RETURN sX, kk
	Version Control	
101	14x80	HWBUILD sX

Reading inputs (1)

- Read the value on an input using instruction INPUT
- Syntax:
  - `INPUT sX, pp`
    - read value on input port location `pp` into register `sX`
  - `INPUT sX, (sY)`
    - read value on input port location pointed to by register `sY` into register `sX`

Reading inputs (2)

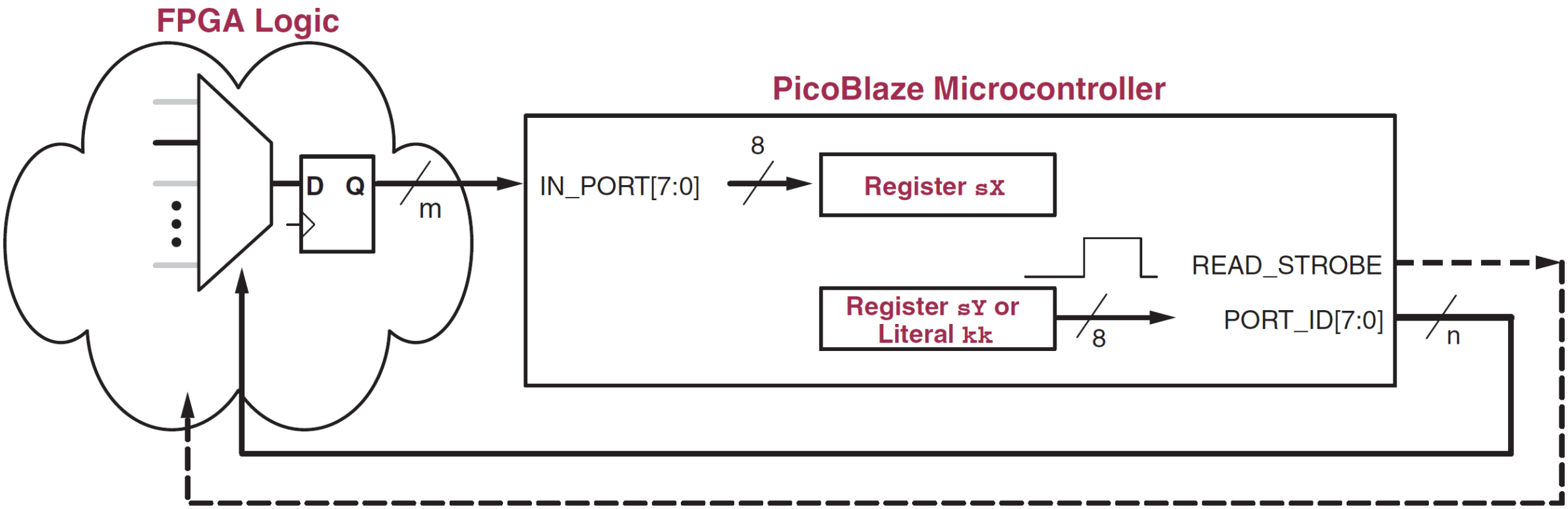
- The value appears on IN_PORT, which we read into register
- PORT_ID defines the input port from where we read



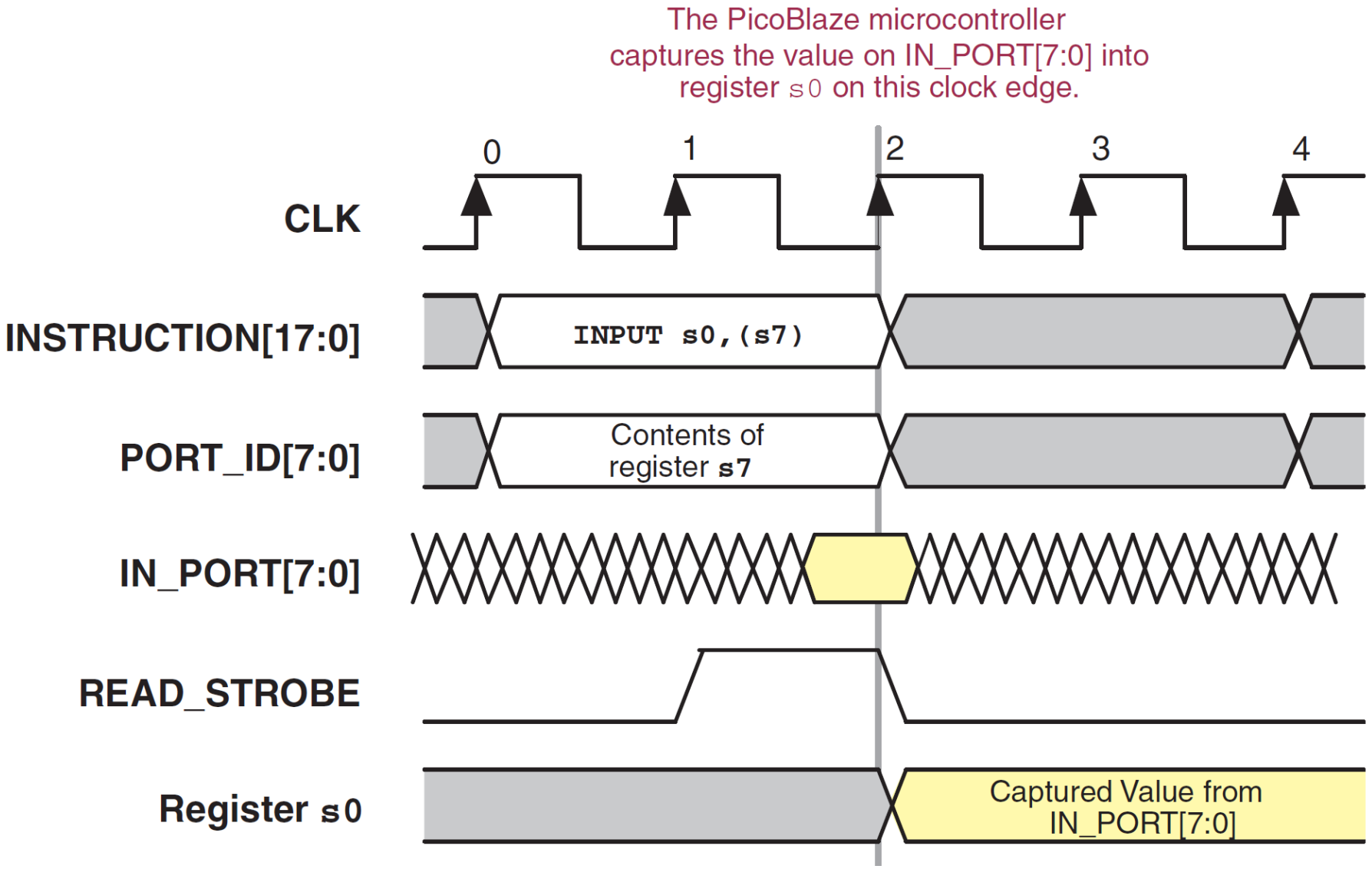
Reading inputs (3)

- READ_STROBE will pulse high for one clock cycle when processor executes an INPUT instruction and indicates that the data is being captured on the next rising edge of the clock
- It is used by peripheral logic when it needs to know that the data has been read. For example, FIFO buffer needs to know when to discard the oldest record.

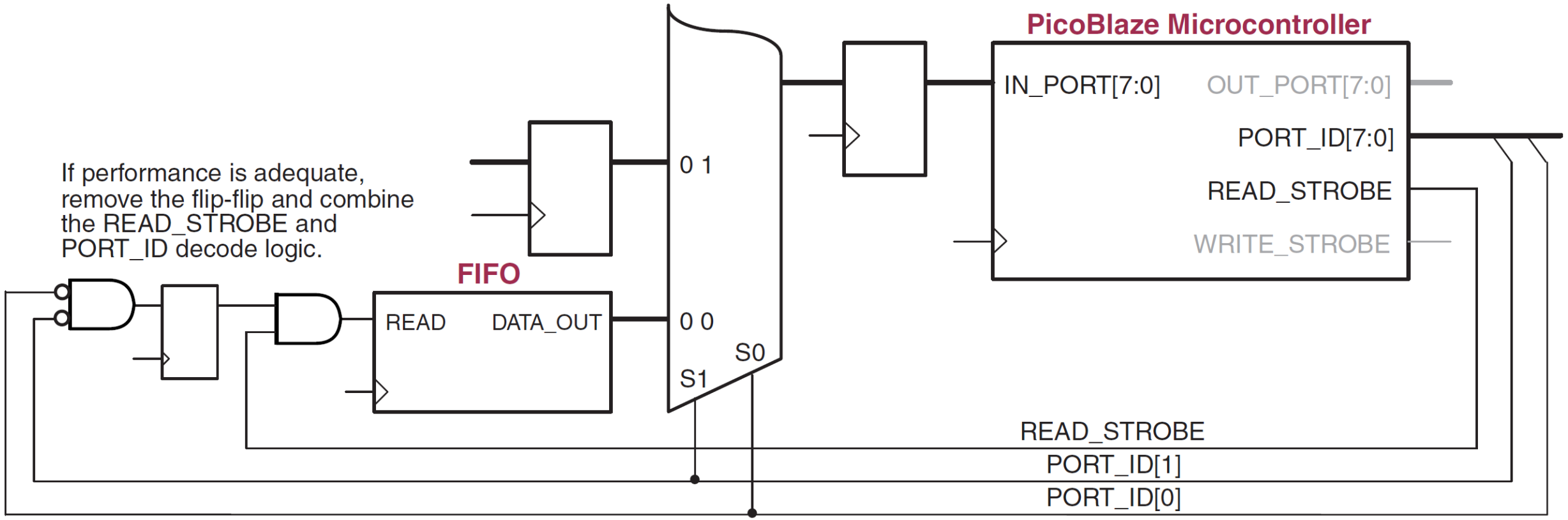
Reading inputs (4)



Reading inputs (5)



Reading inputs (6)

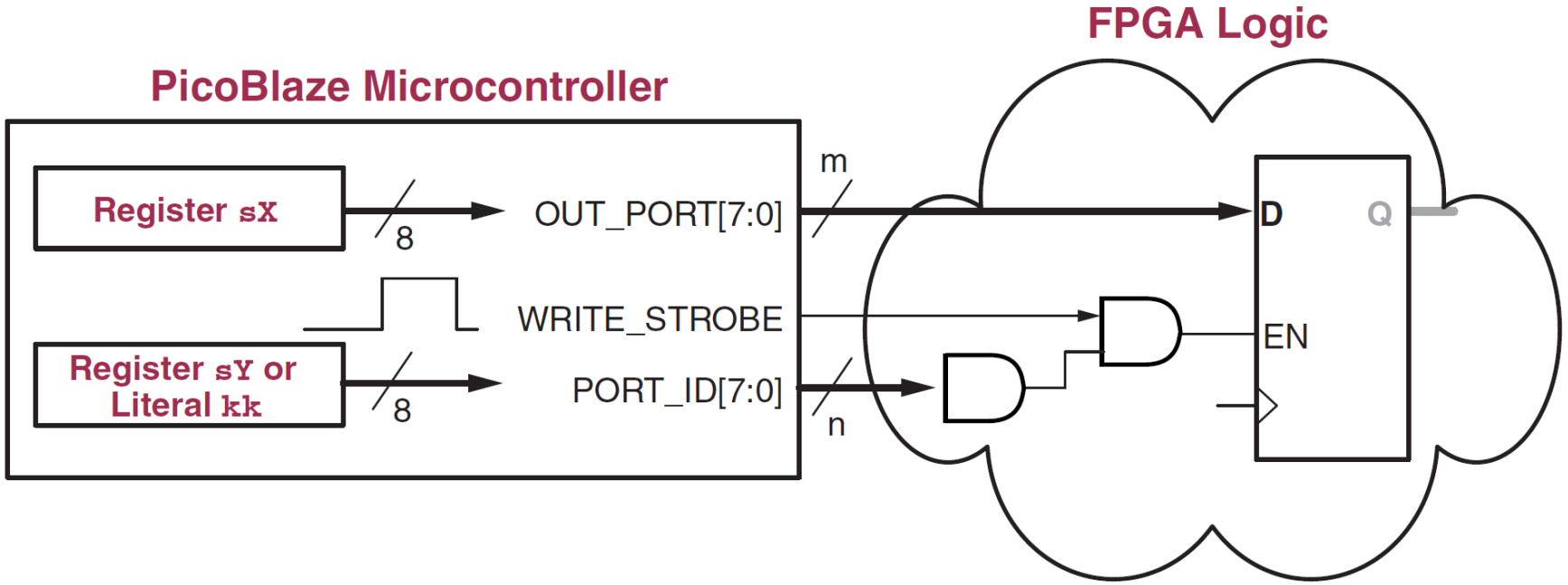


Writing outputs (1)

- With the instruction OUTPUT, we set a value of the output port
- Syntax:
  - `OUTPUT sX, pp`
    - write register `sX` to output port location `pp`
  - `OUTPUT sX, (sY)`
    - write register `sX` to output port location pointed to by register `sY`

Writing outputs (2)

- Output port location is determined by the signal PORT_ID
- Output value appears on the signal OUT_PORT
- On the second cycle of the OUTPUT execution, WRITE_STROBE pulses high



Getting started (1)

- KCPSM6 sources and assembler are available on Moodle and on official webpage
- File *kcpsm6_design_template.vhd* includes a starting template. Copy the following into your project:
  - declarations:
    - components `kcpsm6` and instruction store `<your_program>`, which must match the name of PSM file
    - internal signals

Getting started (2)

- after `begin`
  - instance `processor: kcpsm6`
    - if necessary change the name of the clock signal (`clk`)
  - `kcpsm6_sleep <= '0'; -- disable sleep function`
  - `interrupt <= interrupt_ack; -- we will cover the interrupts next week`
  - instance `program_rom` with the same name as PSM program (no spaces!)
    - if necessary change the name of the clock signal (`clk`)
    - `C_FAMILY => "7S"`
    - `C_RAM_SIZE_KWORDS => 2`
    - `C_JTAG_LOADER_ENABLE => 1`
  - if needed, code for mapping inputs/outputs from/to FPGA
    - follow the rules for effcient decoding of `port_id`
      - inputs: address only as many bits as necessary: `case port_id(1 downto 0) is`
      - outputs: "one hot": `if port_id(1) = '1' then -- port 0x2`

Getting started (3)

- In a project folder *<iproject_name>.srcs/sources_1/new* copy:
  - *kcpsm6.exe*
  - *kcpsm6.vhd*
  - *ROM_form.vhd*
- Create a PSM file with assembly code in the same folder
  - we can edit the program in Vivado editor; however, change the file type: Sources → select PSM file → right click → Source File Properties → change Type into ASM
- Launch assembler
  - `kcpsm6.exe <program_name>.psm`
  - formatted *.fmt* file
  - opensource implementation Opbasm: https://kevinpt.github.io/opbasm
- Vivado: Add sources → *kcpsm6.vhd and <program_name>.vhd*

Challenge

Use the PicoBlaze soft core for controlling LEDs:

- we want to control all 16 LEDs with switches
- with button BTNC we select mode:
  - BTNC = 0 → state of a LED equals switch state: LED[i] = SW[i]
  - BTNC = 1 → inverted: LED[i] = not SW[i]