Designing a Custom AXI-lite Slave Peripheral

LECTURE 9

IOULIIA SKLIAROVA

Custom Hardware

Custom hardware blocks are used to:

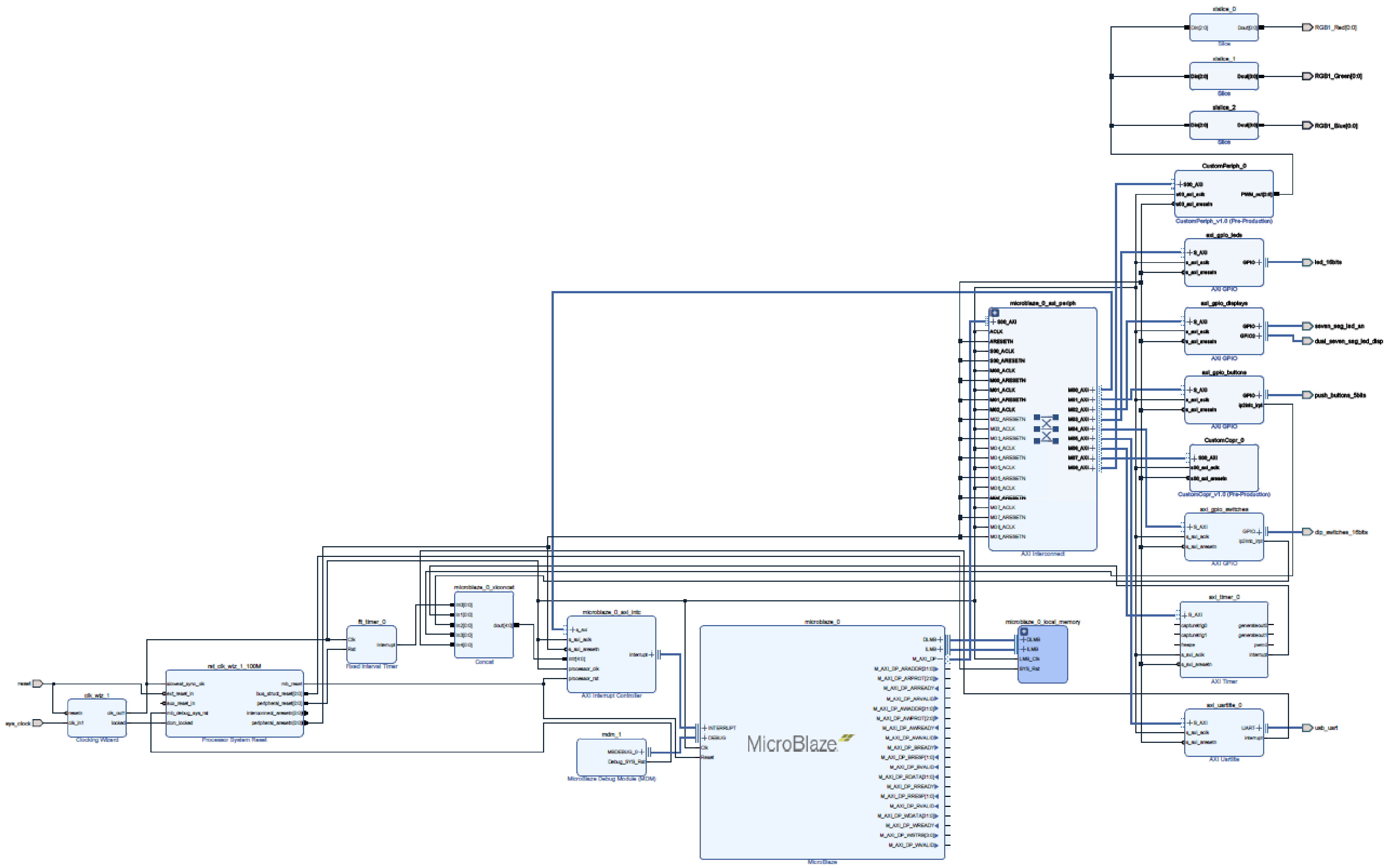
- Delegate to hardware time-critical functions
- Save MicroBlaze resources

One example will be considered, which includes:

- A custom coprocessor with no output ports
- A custom peripheral with output ports

Next lab (lab. 7) will be dedicated to the design and use of a custom peripheral IP – Display Driver.

Block Design (BD)



AXI4 Protocol

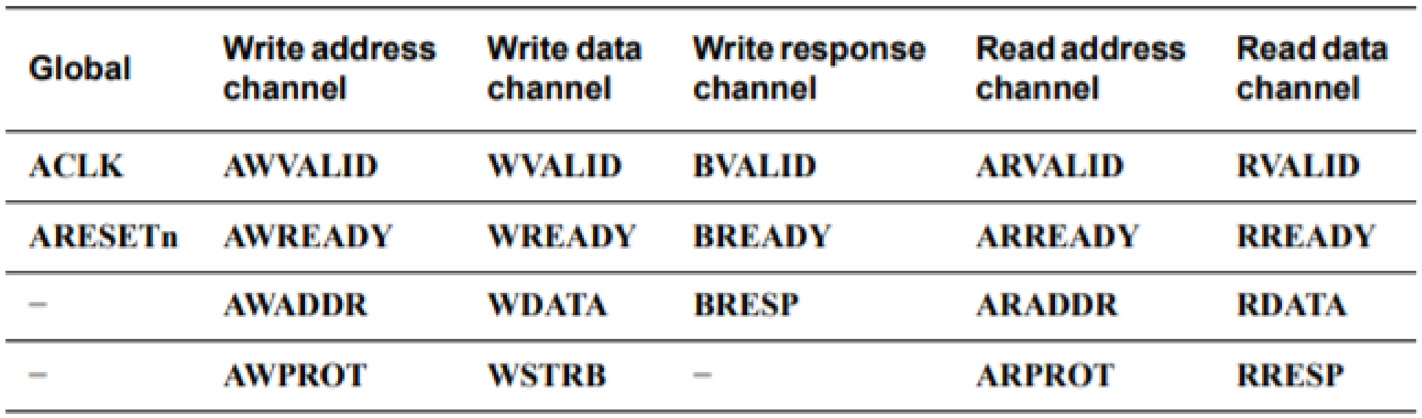
Consult **Vivado AXI Reference Guide** and **AMBA AXI Protocol Specification**

There are three types of AXI4 interfaces:

- **AXI4**: For high-performance memory-mapped requirements.
- **AXI4-Lite**: For simple, low-throughput memory-mapped communication (for example, to and from control and status registers).
- **AXI4-Stream**: For high-speed streaming data.

AXI-Lite:

- all transactions are of burst length 1
- all data accesses use the full width of the data bus
- AXI4-Lite supports a data bus width of 32-bit or 64-bit
- all accesses are non-modifiable, non-bufferable



Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

AXI4-lite Handshaking Signals

Consistent across the five channels.

Based on a simple “Ready” and “Valid” principle:

- “Ready” is used by the recipient to indicate that it is ready to accept a transfer of a data or address value.
- “Valid” is used to clarify that the data (or address) provided on that channel by the sender is valid so that the recipient can then sample it.

“Assert Ready and wait for Valid” ✓

“Assert Valid and wait for Ready” ✓

“Wait for Ready before asserting Valid” ✗

Example 1 – Adder with 3 Operands

Import Block Design

Create and Package new IP (*CustomCopr*)

Add IP

Edit in IP Packager

CustomCopr:

- Adds the contents of 3 registers (written by software)
- Puts the result in the 4th register (read by software)

Apply options from “Aula 6” (Problems and Results – slide 19)

Generate output products

Create HDL Wrapper

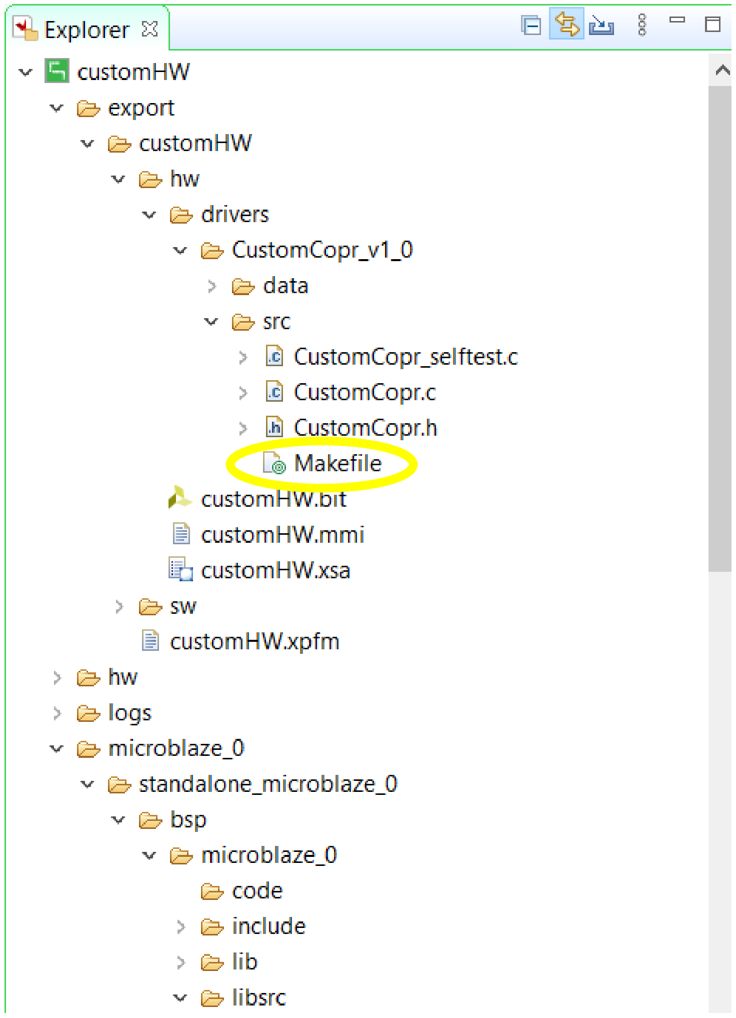
Generate Bitstream

Export Hardware

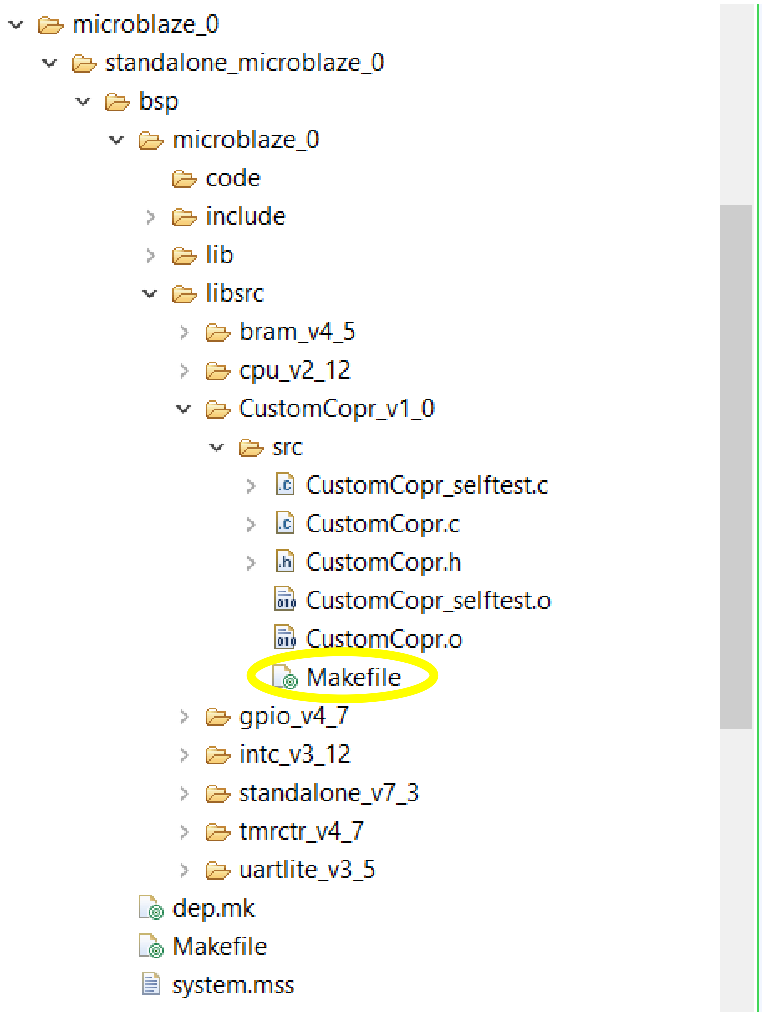
Launch Vitis

Vitis

If build errors do appear, update **all the makefiles** related to the custom IP:



```
Explorer
customHW
  export
    customHW
      hw
        drivers
          CustomCopr_v1_0
            data
            src
              CustomCopr_selftest.c
              CustomCopr.c
              CustomCopr.h
              Makefile
        customHW.bit
        customHW.mmi
        customHW.xsa
      sw
      customHW.xpfm
  hw
  logs
  microblaze_0
    standalone_microblaze_0
      bsp
        microblaze_0
          code
          include
          lib
          libsrc
```



```
microblaze_0
  standalone_microblaze_0
    bsp
      microblaze_0
        code
        include
        lib
        libsrc
          bram_v4_5
          cpu_v2_12
          CustomCopr_v1_0
            src
              CustomCopr_selftest.c
              CustomCopr.c
              CustomCopr.h
              CustomCopr_selftest.o
              CustomCopr.o
              Makefile
          gpio_v4_7
          intc_v3_12
          standalone_v7_3
          tmrctr_v4_7
          uartlite_v3_5
      dep.mk
      Makefile
      system.mss
```

Correct Makefile

```
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)

OBJECTS = $(addsuffix .o, $(basename $(wildcard *.c)))
ASSEMBLY_OBJECTS = $(addsuffix .o, $(basename $(wildcard *.S)))

libs:
          echo "Compiling CustomCopr..."
          $(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
          $(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OBJECTS} ${ASSEMBLY_OBJECTS}
          make clean

include:
          ${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
          rm -rf ${OBJECTS} ${ASSEMBLY_OBJECTS}
```

After Correcting the Makefiles

1. Clean the projects
2. Build the hardware platform
3. Analyze the file xparameters.h (correct the main code if necessary)
4. Build the software application
5. Run the application (source code available on eLearning)

Nexys-4 Tri-color LEDs

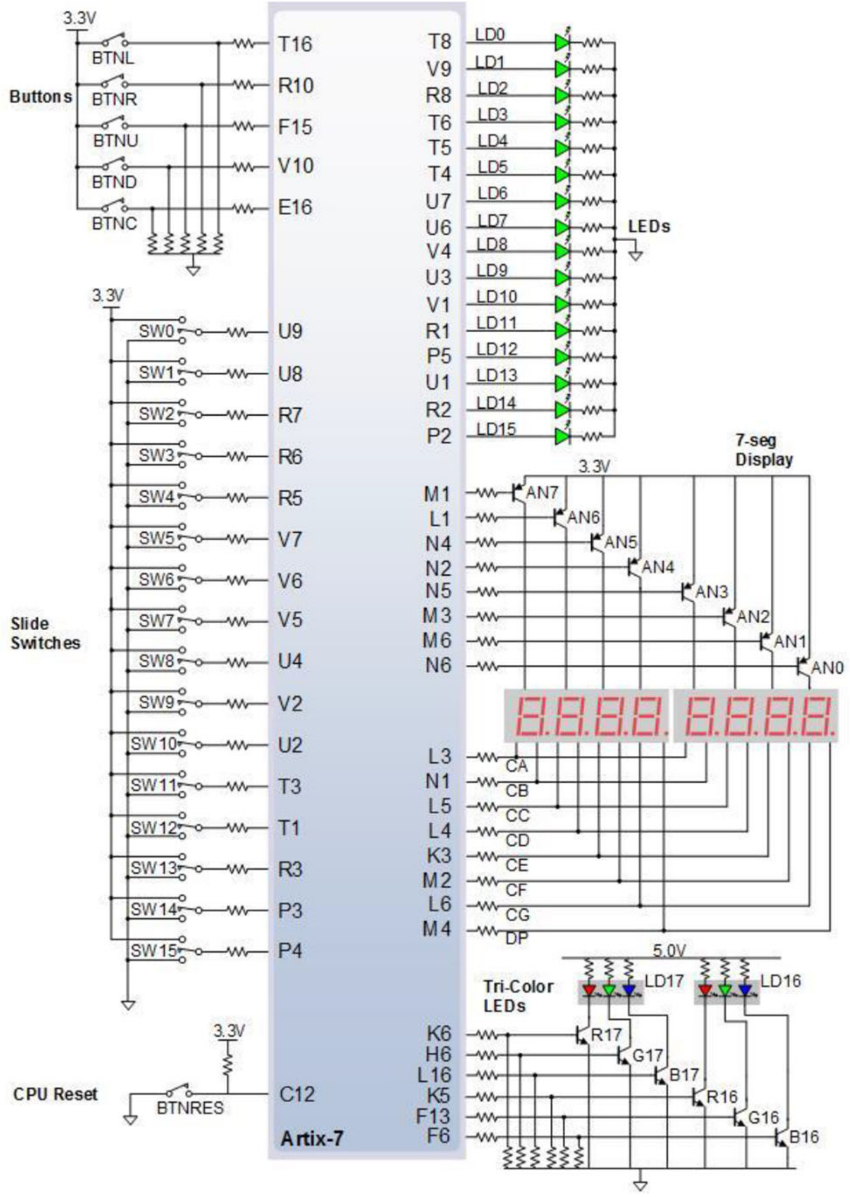
Each tri-color LED has three input signals that drive the cathodes of three smaller internal LEDs: one red, one blue, and one green.

Driving the signal corresponding to one of these colors high will illuminate the internal LED.

The input signals are driven by the FPGA through a transistor, which inverts the signals. Therefore, to light up the tri-color LED, the corresponding signals need to be driven high.

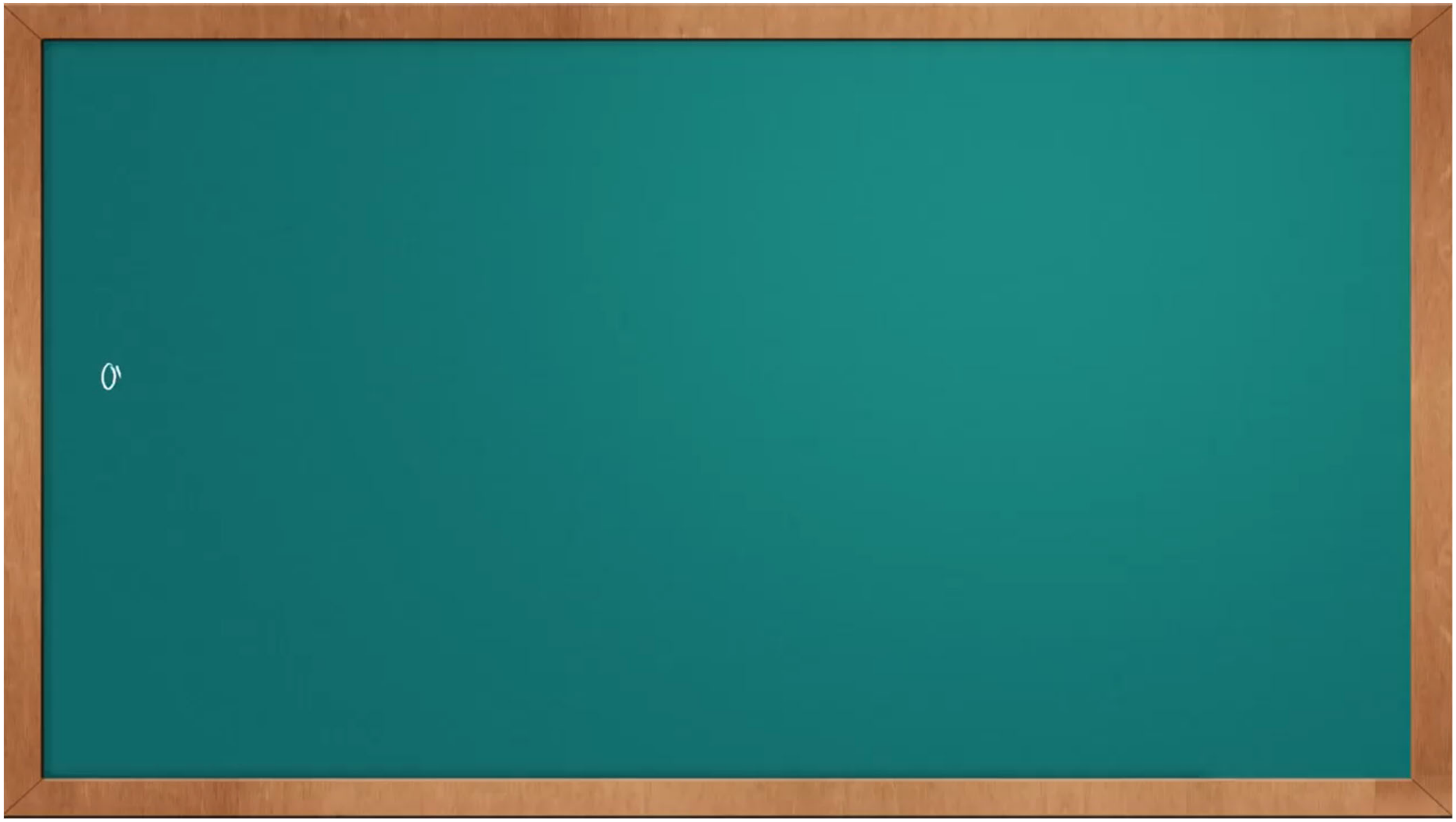
The tri-color LED will emit a color dependent on the combination of internal LEDs that are currently being illuminated.

Note: Driving any of the inputs to a steady logic ‘1’ will result in the LED being illuminated at an uncomfortably bright level. You can avoid this by ensuring that none of the tri-color signals are driven with more than a 50% duty cycle.



Example 2 – triple PWM Generator

PWM – Pulse Width Modulation



PWM

Frequency – how many times per second the LED is switched on and off:

- Cannot be very slow to avoid flicker

Resolution - indicates how many intermediate steps (also called „units“) a PWM cycle has:

- In 8 bit mode, there are 256 levels of brightness

One easy way to build a PWM generator is to use two counters:

- Counter 1 (*s_clkEnbCnt*) limits the frequency of PWM pulses
- Counter 2 (*s_pwmCounter*) controls duty-cycle of PWM pulses (pulse width) within the chosen resolution
- PWM frequency =

  = system clock / (resolution * counter_1_MaxValue)

Final Project - Operation

Select an **operation** suitable for hardware (to increase the performance, to have a clearer implementation)

- an instruction/function not supported by MB (~~popcnt~~, vector operations, complex bitwise/shift logic, specific peripheral, cryptography…)

Repeated operations are not allowed among students

Operations considered during classes are not allowed

Either bring your proposals to lab on May 16 or send them to me by e-mail

- title of the project
- brief description of the functionality (one phrase)
- brief description of the proposed architecture
- why to use the suggested custom hardware module?
- test procedure and user interaction

Final Project – Proposal Example

Title of the project:

- Accelerating Population Count with a Hardware Co-Processor

Brief description of the functionality (one phrase):

- System with a hardware accelerator for calculating population count over a configurable-length array of 32-bit vectors

Brief description of the proposed architecture:

- The system will include a custom hardware module executing the population count operation over a 32-bit input. The module will interact with the MicroBlaze through AXI-Stream interface. The partial results will be accumulated in software. The performance of software and hardware implementations will be analyzed and compared.

Why to use the suggested custom hardware module?

- To reduce the processing time.

Test procedure and user interaction

- Input data will be randomly generated. Software will check the hardware results. Testbench for the accelerator. User interaction through UARTLite and serial terminal.

Lab. 7

Direct the display refresh functions to custom hardware

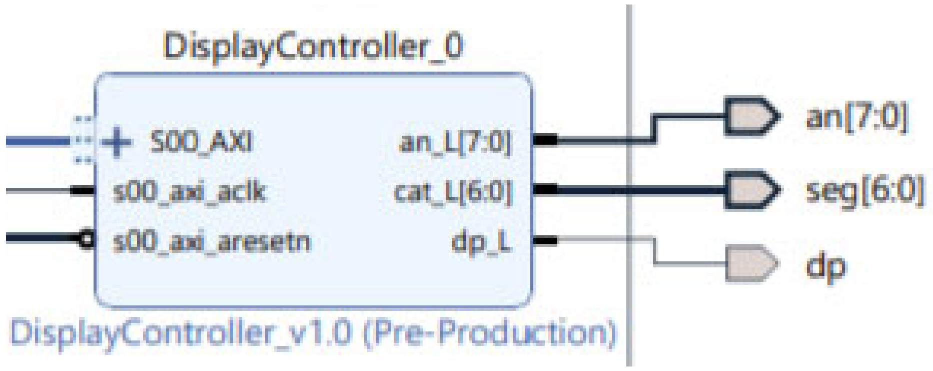
The custom display driver will receive from the MicroBlaze (through AXI-Lite interface):

- Digit enables – 8 bits
- Decimal point enables - 8 bits
- Digit values - 32 bits (8 x 4 bits)

The custom display driver will produce on its outputs:

- an – 8 bits
- seg – 7 bits
- dp – 1 bit

The original GPIO_Displays must be deleted



Final Remarks

At the end of this lecture you should be able to:

- Design custom hardware modules interacting with the MicroBlaze through AXI-Lite interface
- write C programs that make use of custom hardware

To do:

- Construct the considered hardware platforms
- Test the given applications in Vitis
- Do lab. 7