MIPS Open Developer Day Package. Lab MO2 - Using switchable clock to observe the CPU cycle-by-cycle

0. Note. This documentation is in the process of being updated

This documentation is in the process of being updated to match the latest version of the source code. The planned updates include:

1. Informing the user how to use makefiles instead of short Linux scripts, Windows batch files and GUI. To see all available make actions just run *make* in any board or program directory example:

   make in subdirectories of *boards* directory:

   ```
   make help   - show this message
   make all    - clean, create the board project and run the synthesis
   make clean  - delete synth folder
   make create - create the board project
   make open   - open the board project
   make build  - build the board project
   make load   - program the FPGA board
   ```

   make in subdirectories of *programs* directory:

   ```
   make help      - show this message
   make all       - alternative for: compile program size disasm readmemh srecord
   make program   - build program.elf from sources
   make compile   - compile all C sources to ASM
   make size      - show program size information
   make disasm    - disassemble program.elf
   make readmemh  - create verilog memory init file for simulation
   make srecord   - create Motorola S-record file to use it with UART loader
   make clean     - delete all created files
   make load      - load program into the device memory, run it and detach gdb
   make debug     - load program into the device memory, wait for gdb commands
   make attach    - attach to the device, wait for gdb commands
   make uart      - load program into the device memory using UART loader
   make modelsim  - simulate program and device using Modelsim
   make icarus    - simulate program and device using Icarus Verilog
   make gtkwave   - show the result of Icarus Verilog simulation in GTKWave
   ```

2. Adding support for *make uart UART=N* where *N* is USB-to-UART device number (*/dev/ttyUSB0, 1, 2, ... N*).

3. Light Sensor in peripheral integration lab is now integrated not as an addional GPIO (General Purpose I/O) device inside GPIO AHB-Lite slave, but as a separate AHB-Lite slave.

4. Source code for pipeline bypass lab (*Lab MO6 - The first glance into pipelining*) does not match the documentation. Please look inside the code to see what it does.

5. The new CorExtend / UDI - User Defined Instructions example does not have an instruction. Please refer to MIPS Open Day slides to figure out what it does.

1. Introduction

This lab introduces switchable clock, a feature of MIPSfpga that enables the whole new category of student labs. With this feature you can first run the processor with its usual multi megahertz frequency, then switch it to the frequence of a few clocks per second and observe how the processor works live. A typical usage is to connect to external LEDs the CPU signals that control cache evictions or pipeline forwarding and observe the LED patterns when running different sequence of code. This usage is demonstrated in *MIPSfpga. Lab MO5 - The first glance into caches* and *MIPSfpga. Lab MO6 - The first glance into pipelining*.

2. The theory of operation

When implemented in ASIC using 28 nm technology, MIPS microAptiv UP can run up to 500 MHz; when implemented using 65 nm - more than 300 MHz.

When MIPS microAptiv UP is synthesized for FPGA together with block memory in MIPSfpga system, the frequency is much lower - around 50-60 MHz, both for Xilinx and Altera. The best case for any FPGA is around 150 MHz, with the smallest amount of block memory on some FPGAs.

The introductory student boards tested for MIPSfpga all have clock generators able to generate a clock signal with the frequency of 50-100 MHz. This frequency can be increased or decreased using phase-locked loop (PLL). Unfortunately, PLL cannot be used to lower the frequency below approximately 100 KHz. In order to lower the frequency even further, other methods have to be used.

Altera has a special macro for such situation called *ALTCLKCTLR*, but it does not work for all cases. As a result, the switchable clock in MIPSfpga is implemented using a combination of a counter and a global buffer (Xilinx macro *BUFG* and Altera macro *global*). The buffer is necessary, without it the synthesizer will not treat the output of the switchable clock module as a clock and the design will not work. It is also necessary to have timing constraints in the constraint file for Altera Quartus.

The frequency is controlled by two switches on the board. Switches also have to be debounced.

Figure 1 show how the switchable clock module is instantiated in the module hierarchy for Digilent Nexys4 DDR board that carries Xilinx Artix-7 FPGA. **Figure 2** shows the same for Terasic DE0-CV board that carries Altera Cyclon V FPGA.

Figure 1. MIPSfpga module hierarchy, including the switchable clock module, for <u>Digilent Nexys4 DDR board</u> that carries Xilinx Artix-7 FPGA
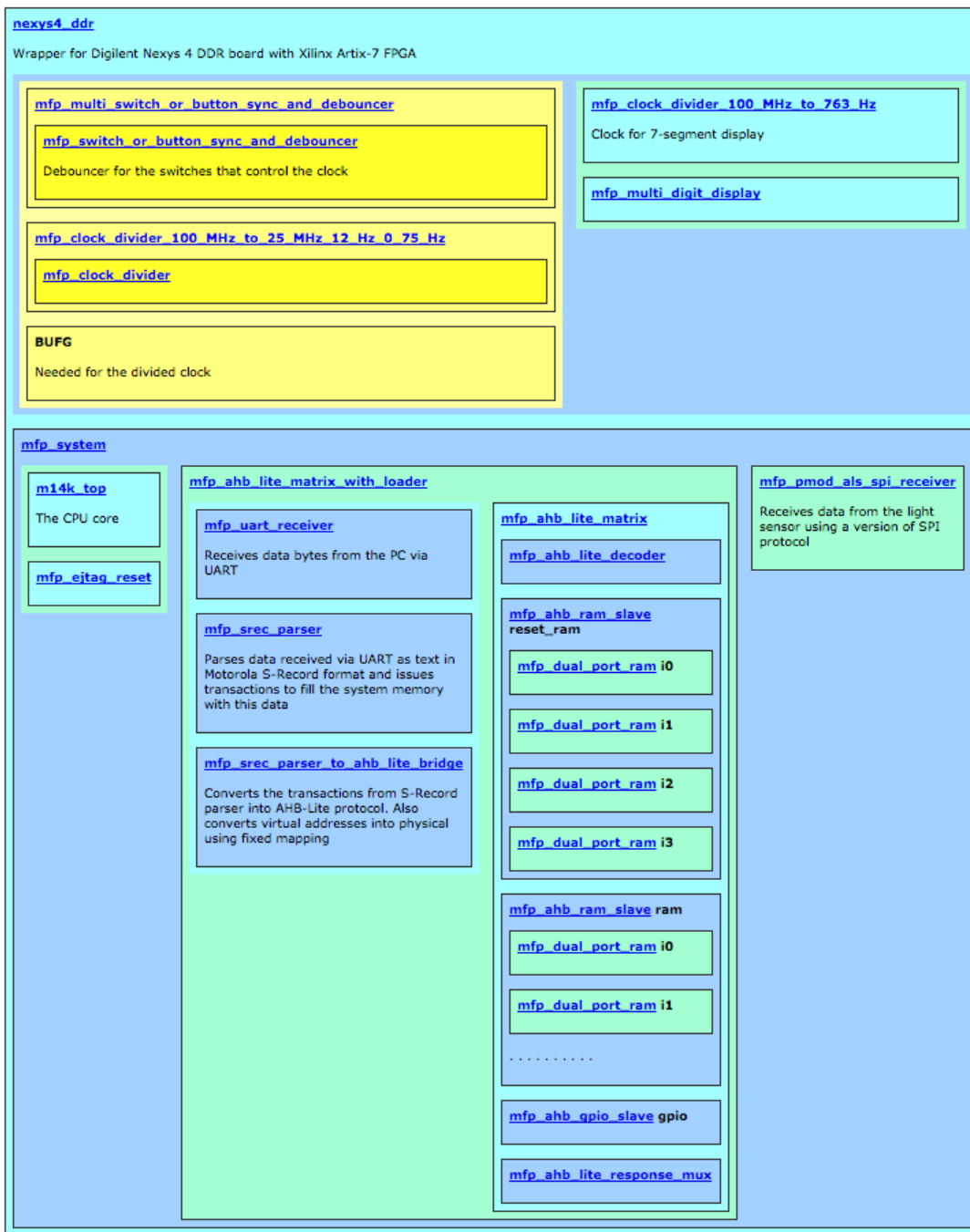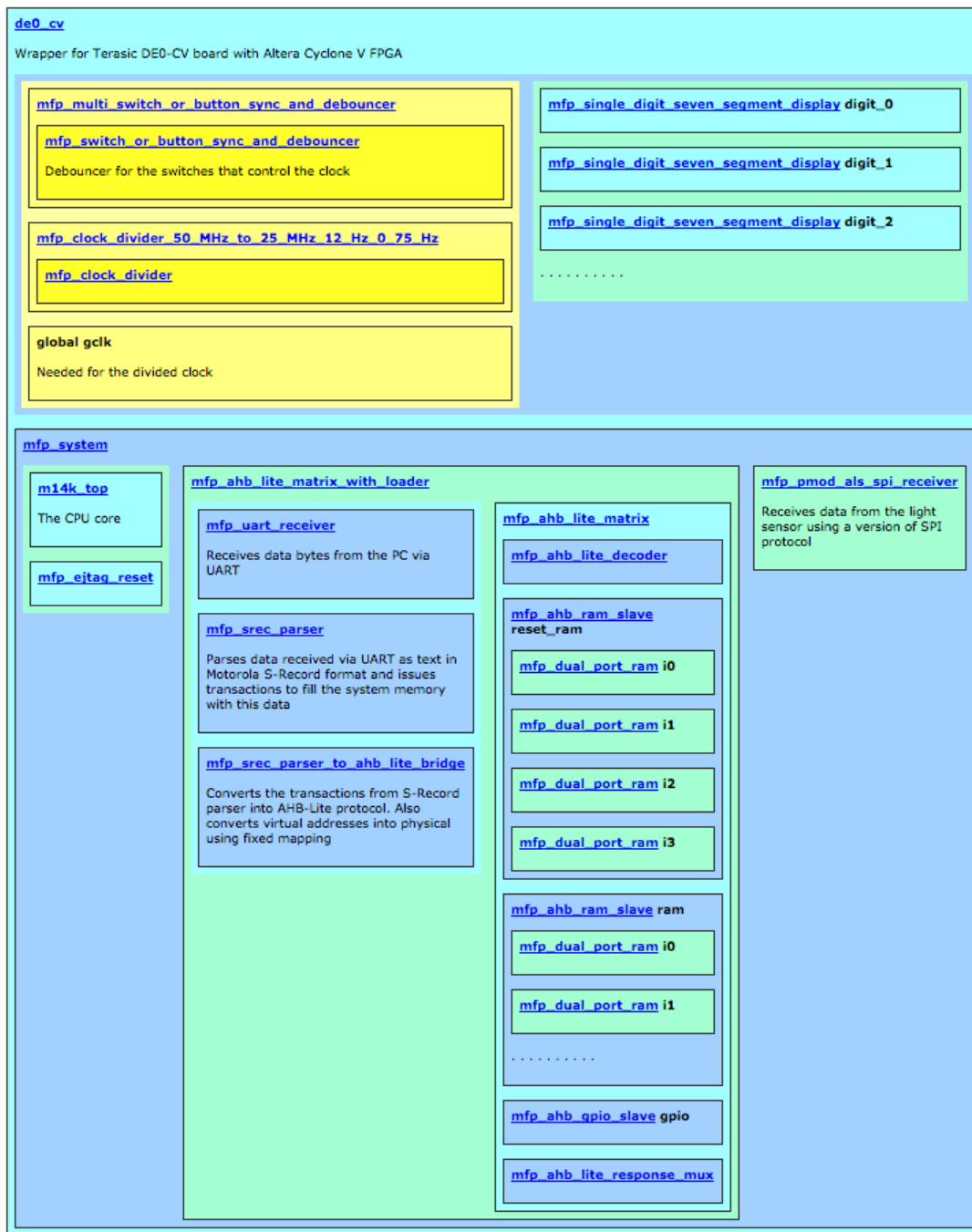
nexys4_ddr

Wrapper for Digilent Nexys 4 DDR board with Xilinx Artix-7 FPGA

mfp_multi_switch_or_button_sync_and_debouncer

mfp_switch_or_button_sync_and_debouncer

Debouncer for the switches that control the clock

mfp_clock_divider_100_MHz_to_25_MHz_12_Hz_0_75_Hz

mfp_clock_divider

BUFG

Needed for the divided clock

mfp_clock_divider_100_MHz_to_763_Hz

Clock for 7-segment display

mfp_multi_digit_display

mfp_system

m14k_top

The CPU core

mfp_ejtag_reset

mfp_ahb_lite_matrix_with_loader

mfp_uart_receiver

Receives data bytes from the PC via UART

mfp_srec_parser

Parses data received via UART as text in Motorola S-Record format and issues transactions to fill the system memory with this data

mfp_srec_parser_to_ahb_lite_bridge

Converts the transactions from S-Record parser into AHB-Lite protocol. Also converts virtual addresses into physical using fixed mapping

mfp_ahb_lite_matrix

mfp_ahb_lite_decoder

mfp_ahb_ram_slave
reset_ram

mfp_dual_port_ram i0

mfp_dual_port_ram i1

mfp_dual_port_ram i2

mfp_dual_port_ram i3

mfp_ahb_ram_slave ram

mfp_dual_port_ram i0

mfp_dual_port_ram i1

. . . .

mfp_ahb_gpio_slave gpio

mfp_ahb_lite_response_mux

mfp_pmod_als_spi_receiver

Receives data from the light sensor using a version of SPI protocol

Figure 2. MIPSfpga module hierarchy, including the switchable clock module, for Terasic DE0-CV board that carries Altera Cyclon V FPGA

de0_cv

Wrapper for Terasic DE0-CV board with Altera Cyclone V FPGA

mfp_multi_switch_or_button_sync_and_debouncer

mfp_switch_or_button_sync_and_debouncer

Debouncer for the switches that control the clock

mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz

mfp_clock_divider

global gclk

Needed for the divided clock

mfp_single_digit_seven_segment_display digit_0

mfp_single_digit_seven_segment_display digit_1

mfp_single_digit_seven_segment_display digit_2

. . . .

mfp_system

m14k_top
The CPU core

mfp_ejtag_reset

mfp_ahb_lite_matrix_with_loader

mfp_uart_receiver

Receives data bytes from the PC via UART

mfp_srec_parser

Parses data received via UART as text in Motorola S-Record format and issues transactions to fill the system memory with this data

mfp_srec_parser_to_ahb_lite_bridge

Converts the transactions from S-Record parser into AHB-Lite protocol. Also converts virtual addresses into physical using fixed mapping

mfp_ahb_lite_matrix

mfp_ahb_lite_decoder

mfp_ahb_ram_slave
reset_ram

mfp_dual_port_ram i0

mfp_dual_port_ram i1

mfp_dual_port_ram i2

mfp_dual_port_ram i3

mfp_ahb_ram_slave ram

mfp_dual_port_ram i0

mfp_dual_port_ram i1

. . . .

mfp_ahb_gpio_slave gpio

mfp_ahb_lite_response_mux

mfp_pmod_als_spi_receiver

Receives data from the light sensor using a version of SPI protocol

This lab also uses a special module to drive multi-digit seven-segment display on the board. A variant of such module for Digilent boards contains the code that handles dynamic displays and a variant for Terasic boards handles static displays.

3. Lab steps

This section outlines the sequence of steps, necessary to complete the lab. Almost all generic steps in this lab are the same as in *MIPSfpga 2.0 Lab MO1. Using MIPSfpga with Serial Loader Flow that does not require BusBlaster board and OpenOCD software*. Such generic steps are not described in this section. Only the steps different from *Lab MO1* are explained in details.

3.1. Review the lab code modifications in *boards* and *system_rtl* directories

Search for *MFP_USE_SLOW_CLOCK_AND_CLOCK_MUX* symbol in *boards* and *system_rtl* directories. Review the code fragments where that symbol occurs.

3.1.1 System configuration

Review and possibly modify the configuration parameters in the file *system_rtl/mfp_ahb_lite_matrix_config.vh* as follows:

File *mfp_ahb_lite_matrix_config.vh*

```
//
//  Configuration parameters
//

// `define MFP_USE_WORD_MEMORY
// `define MFP_INITIALIZE_MEMORY_FROM_TXT_FILE
`define MFP_USE_SLOW_CLOCK_AND_CLOCK_MUX
`define MFP_USE_UART_PROGRAM_LOADER
// `define MFP_DEMO_LIGHT_SENSOR
// `define MFP_DEMO_INTERRUPTS
// `define MFP_DEMO_CACHE_MISSES
// `define MFP_DEMO_PIPE_BYPASS
```

3.1.2 Review board wrapper file for Xilinx

The board wrapper for Digilent Nexys4 DDR board with Xilinx Artix-7 FPGA instantiates three new modules:

1. A module that divides the clock - *mfp_clock_divider_100_MHz_to_25_MHz_12_Hz_0_75_Hz*
2. A module that debounces the switches that control the clock - *mfp_multi_switch_or_button_sync_and_debouncer*
3. A Xilinx-specific module *BUFG* that tells Vivado software that the generated signal should be treated as a clock by synthesis and routed like a clock during place-and-route

File *nexys4_ddr.v*

```
`include "mfp_ahb_lite_matrix_config.vh"

module nexys4_ddr
(
    input         CLK100MHZ,
    input         CPU_RESETN,

    . . . . . . . . . . . . . . . . . . .

    input  [15:0] SW,

    . . . . . . . . . . . . . . . . . . .
);

    wire clk;

    `ifdef MFP_USE_SLOW_CLOCK_AND_CLOCK_MUX

    wire        muxed_clk;
    wire [1:0] sw_db;

    mfp_multi_switch_or_button_sync_and_debouncer
    # (.WIDTH (2))
    mfp_multi_switch_or_button_sync_and_debouncer
    (
        .clk    ( CLK100MHZ ),
        .sw_in  ( SW [1:0]  ),
        .sw_out ( sw_db     )
    );

    mfp_clock_divider_100_MHz_to_25_MHz_12_Hz_0_75_Hz
    mfp_clock_divider_100_MHz_to_25_MHz_12_Hz_0_75_Hz
    (
        .clki    ( CLK100MHZ ),
        .sel_lo  ( sw_db [0] ),
        .sel_mid ( sw_db [1] ),
        .clko    ( muxed_clk )
    );
```

```
        BUFG BUFG_slow_clk (.O ( clk ), .I ( muxed_clk ));

        `else

        clk_wiz_0 clk_wiz_0 (.clk_in1 (CLK100MHZ), .clk_out1 (clk));

        `endif

        . . . . . . . . . . . . . . . . . . . .

        mfp_system mfp_system
        (
            .SI_ClkIn           (   clk             ),
            .SI_Reset           ( ~ CPU_RESETN      ),

            . . . . . . . . . . . . . . . . . .
        );

        . . . . . . . . . . . . . . . . . . . .

    endmodule
```

3.1.3 Review board wrapper file and timing constraints for Altera

The board wrapper for Terasic DE0-CV board with Altera Cyclone V FPGA instantiates three new modules:

1. A module that divides the clock - *mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz*
2. A module that synchronizes and debounces the switches that control the clock - *mfp_multi_switch_or_button_sync_and_debouncer. Without debouncing the control signals, the clock may not work reliably during the mode switching.*
3. An Altera-specific module *global* that tells Vivado software that the generated signal should be treated as a clock by synthesis and routed like a clock during place-and-route

File *de0_cv.v*

```
`include "mfp_ahb_lite_matrix_config.vh"

module de0_cv
(
    input           CLOCK2_50,
    input           CLOCK3_50,
    inout           CLOCK4_50,
    input           CLOCK_50,

    input           RESET_N,

    input   [ 3:0]  KEY,
    input   [ 9:0]  SW,

    . . . . . . . . . . . . . . . . . . . .
);

    wire clk;

    `ifdef MFP_USE_SLOW_CLOCK_AND_CLOCK_MUX

    wire        muxed_clk;
    wire [1:0]  sw_db;

    mfp_multi_switch_or_button_sync_and_debouncer
    # (.WIDTH (2))
    mfp_multi_switch_or_button_sync_and_debouncer
    (
        .clk    ( CLOCK_50 ),
        .sw_in  ( SW [1:0] ),
```

```
                .sw_out ( sw_db    )
        );

        mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz
        mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz
        (
                .clki    ( CLOCK_50  ),
                .sel_lo  ( sw_db [0] ),
                .sel_mid ( sw_db [1] ),
                .clko    ( muxed_clk )
        );

        global gclk
        (
                .in     ( muxed_clk  ),
                .out    ( clk        )
        );

        `else

        assign clk = CLOCK_50;

        `endif

        . . . . . . . . . . . . . . . . . . .

        mfp_system mfp_system
        (
                .SI_ClkIn          (   clk              ),
                .SI_Reset          ( ~ RESET_N          ),

                . . . . . . . . . . . . . . . . .
        );

        . . . . . . . . . . . . . . . . . . .

    endmodule
```

It is also necessary to modify the timing constrains in the board-specific SDC file:

File *de0_cv.sdc*

```
#***************************************************************
# Create Clock
#***************************************************************

create_clock -period 20 [get_ports CLOCK2_50]
create_clock -period 20 [get_ports CLOCK3_50]
create_clock -period 20 [get_ports CLOCK_50]
create_clock -period 20 [get_ports CLOCK4_50]
create_clock -period 40 [get_nets mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz|clko]
create_clock -period 20 [get_nets de0_cv|clk]
create_clock -period 20 [get_ports GPIO_1[17]]


. . . . . . . . . . . . . . . . . . . . . .
```

3.1.4 Review clock divider modules

The clock divider modules use simple counter to generate a clock with the needed frequency:

File *mfp_clock_dividers.v*

```
module mfp_clock_divider
(
    input       clki,
```

```verilog
    input       sel_lo,
    input       sel_mid,
    output reg clko
);

    // 50 MHz / 2 ** 26 = 0.75 Hz

    parameter DIV_POW_FASTEST = 1;
    parameter DIV_POW_SLOWEST = 26;

    reg [DIV_POW_SLOWEST - 1: 0] counter;

    always @ (posedge clki)
        counter <= counter + 1'b1;

    always @ (posedge clki)
        clko <= sel_lo  ? counter [DIV_POW_SLOWEST - 1] :
                sel_mid ? counter [DIV_POW_SLOWEST - 5] :
                          counter [DIV_POW_FASTEST - 1] ;

endmodule

//----------------------------------------------------------------------

module mfp_clock_divider_100_MHz_to_25_MHz_12_Hz_0_75_Hz
(
    input  clki,
    input  sel_lo,
    input  sel_mid,
    output clko
);

    mfp_clock_divider # (.DIV_POW_FASTEST (2), .DIV_POW_SLOWEST (27))
    mfp_clock_divider (clki, sel_lo, sel_mid, clko);

endmodule

//----------------------------------------------------------------------

module mfp_clock_divider_50_MHz_to_25_MHz_12_Hz_0_75_Hz
(
    input  clki,
    input  sel_lo,
    input  sel_mid,
    output clko
);

    mfp_clock_divider # (.DIV_POW_FASTEST (1), .DIV_POW_SLOWEST (26))
    mfp_clock_divider (clki, sel_lo, sel_mid, clko);

endmodule

    . . . . . . . . . . . . . . . . . . . . . . .
```

3.1.5 Review the debouncing modules

Synchronizing and debouncing the clock control signals is necessary for the clock to work reliably during the mode switching. Otherwise signal bouncing and a possibility of metastable state can introduce bugs in the work of the processor.

File *mfp_switch_and_button_debouncers.v*

```verilog
module mfp_switch_or_button_sync_and_debouncer
# (
    parameter DEPTH = 8
)
```

```verilog
(
    input       clk,
    input       sw_in,
    output reg sw_out
);

    reg  [ DEPTH - 1 : 0] cnt;
    reg  [         2 : 0] sync;
    wire                  sw_in_s;

    assign sw_in_s = sync [2];

    always @ (posedge clk)
        sync <= { sync [1:0], sw_in };

    always @ (posedge clk)
        if (sw_out ^ sw_in_s)
            cnt <= cnt + 1'b1;
        else
            cnt <= { DEPTH { 1'b0 } };

    always @ (posedge clk)
        if (cnt == { DEPTH { 1'b1 } })
            sw_out <= sw_in_s;

endmodule

//--------------------------------------------------------------------

module mfp_multi_switch_or_button_sync_and_debouncer
# (
    parameter WIDTH = 1, DEPTH = 8
)
(
    input                     clk,
    input  [ WIDTH - 1 : 0 ] sw_in,
    output [ WIDTH - 1 : 0 ] sw_out
);

    genvar sw_cnt;

    generate
        for (sw_cnt = 0; sw_cnt < WIDTH; sw_cnt = sw_cnt + 1)
        begin : GEN_DB_B

            mfp_switch_or_button_sync_and_debouncer
            # (.DEPTH (8))
            mfp_switch_or_button_sync_and_debouncer
            (
                .clk    ( clk             ),
                .sw_in  ( sw_in  [sw_cnt] ),
                .sw_out ( sw_out [sw_cnt] )
            );

        end
    endgenerate

endmodule
```

3.2. Connect the board to the computer

For *Digilent* boards, such as *Nexys4*, *Nexys4 DDR* or *Basys3*, this step is obvious. For *Altera/Terasic* boards some additional steps required:

1. Connect USB-to-UART connector to FPGA board. Either *FT232RL* or *PL2303TA* that you can by from AliExpress or Radio Shack will do the job. *TX* output from the connector (green wire on *PL2303TA*) should go to pin 3 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano) and *GND* output (black wire on *PL2303TA*) should be connected to pin 6 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano). Please consult photo picture in *Lab MO1* to avoid short-circuit or other connection problems.
2. For *FT232RL* connector: make sure to set 3.3V/5V jumper on *FT232RL* part to 3.3V.
3. For the boards that require external power in addition to the power that comes from USB, connect the power supply. The boards that require the extra power supply include *Terasic DE2-115*.
4. Connect FPGA board to the computer using main connection cable provided by the board manufacturers. Make sure to put USB cable to the right jack when ambiguity exists (such as in *Terasic DE2-115* board).
5. Make sure to power the FPGA board (turn on the power switch) before connecting the UART cable from USB-to-UART connector to the computer. Failing to do so may result in electric damage to the board.
6. Connect USB-to-UART connector to FPGA board.

3.3 Run the synthesis and configure the FPGA with the synthesized MIPSfpga system

This step is identical to the synthesis step in *Lab MO1*

3.4 Go to the lab directory and clean it up

Under Windows:

```
cd programs\lab_mo2
00_clean_all.bat
```

Under Linux:

```
cd programs/lab_mo2
00_clean_all.sh
```

3.5 Review the lab program code

The lab program code is a simple counter. The value output to multi-digit seven-segment display is constructed in a way that shows the incrementing digits with both fast multi-MHz clock and slow ~1 Hz clock:

File *main.c*

```
#include "mfp_memory_mapped_registers.h"

int main ()
{
    int n;

    for (n = 0;; n ++)
    {
        MFP_RED_LEDS      = n;
        MFP_GREEN_LEDS    = n;
        MFP_7_SEGMENT_HEX = ((n >> 8) & 0xffffff00) | (n & 0xff);
    }

    return 0;
}
```

3.6 Prepare the first software run

Following the procedure described in *Lab MO1*, compile and link the program, generate Motorola S-Record file and upload this file into the memory of the synthesized MIPSfpga-based system on the board.

Under Windows:

1. cd programs\lab_mo2
2. run 02_compile_and_link.bat
3. run 08_generate_motorola_s_record_file.bat
4. run 11_check_which_com_port_is_used.bat
5. edit 12_upload_to_the_board_using_uart.bat based on the result from the previous step - set the working port in "set a=" assignment.

6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. run 12_upload_to_the_board_using_uart.bat

Under Linux:

If uploading program to the board first time during the current Linux session, add the current user to *dialout* Linux group. Enter the *root* password when prompted:

```
sudo adduser $USER dialout
su - $USER
```

After that:

1. cd programs/lab_mo2
2. run ./02_compile_and_link.sh
3. run ./08_generate_motorola_s_record_file.sh
4. run ./11_check_which_com_port_is_used.sh
5. edit ./12_upload_to_the_board_using_uart.sh based on the result from the previous step - set the working port in "set a=" assignment
6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. ./run 12_upload_to_the_board_using_uart.sh

3.7 Run the software on the board

Reset the processor. The reset buttons for each board are listed in the table below:

Board	Reset button
Digilent Basys3	Up
Digilent Nexys4	Dedicated CPU Reset
Digilent Nexys4 DDR	Dedicated CPU Reset
Terasic DE0	Button/Key 0
Terasic DE0-CV	Dedicated reset button
Terasic DE0-Nano	Button/Key 0
Terasic DE1	Button/Key 0
Terasic DE2-115	Button/Key 0
Terasic DE10-Lite	Button/Key 0

At this moment you should see the counter running and generating the output on seven-segment diplay on FPGA board. Turn the switch 0 on. This will switch the system clock from 25 MHz to 0.75 Hz, less than one beat per second. Can you see the sharp difference in counter output? If you don't, check the synthesis configuration.

Now try to turn switch 1 on and switch 0 off. What happened with the speed of counting? Now you are ready to use the switchable clock feature for the subsequent labs.