Programmable Processors

Test Processor
The Key Conditioner
Two Clocks .sdc file

Refresh – The Sizes of Things

- Instruction memory (ROM):128 X 16
- Data memory (RAM): 256 X 16
- Register file: 16 X 16
- ALU: Two 16-bit inputs, 16-bit output

- Instruction Set – List of allowable instructions and their representation in memory
- Each of our instructions is 16 bits long
- Most of them contain some address information
- General form : **operation source destination**

NOOP instruction – **0000 0000 0000 0000**

STORE instruction – **0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**

LOAD instruction – **0010 $d_7d_6d_5d_4d_3d_2d_1d_0$ $r_3r_2r_1r_0$**

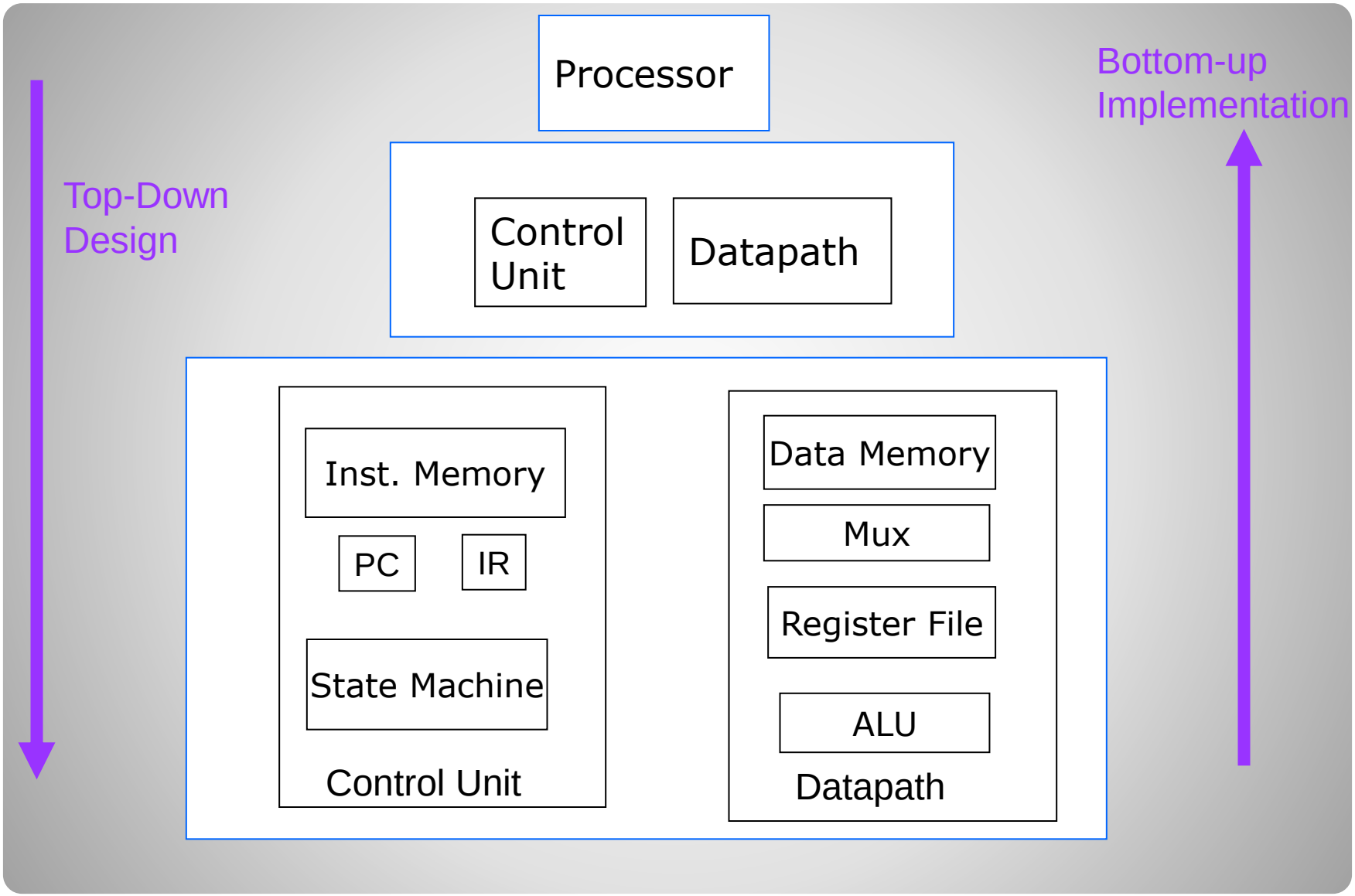
ADD instruction – **0011 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

SUBTRACT instruction – **0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

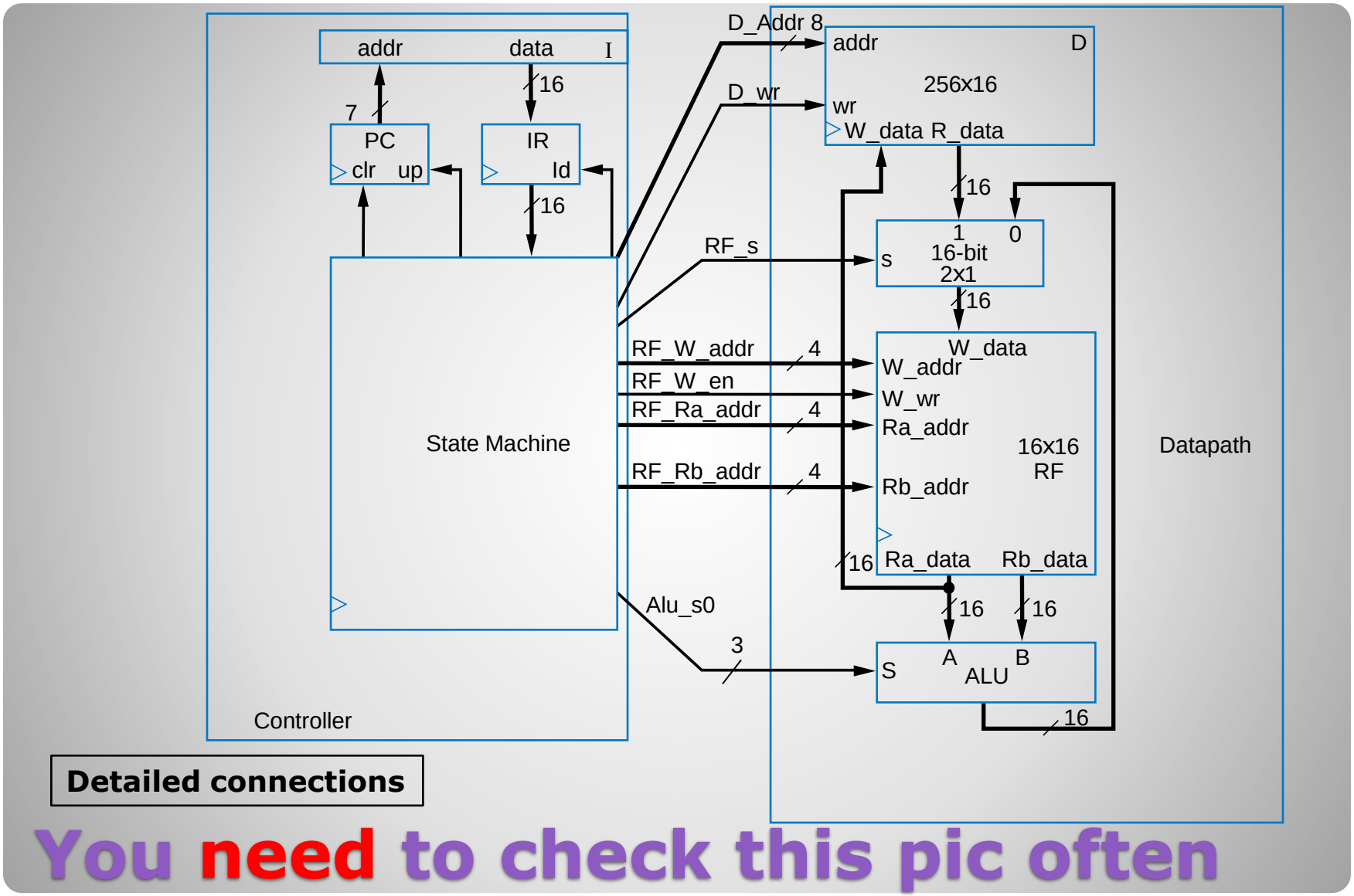
HALT instruction – **0101 0000 0000 0000**

'r's are Register File locations (4 bits each)
'd's are Data Memory locations (8 bits each)

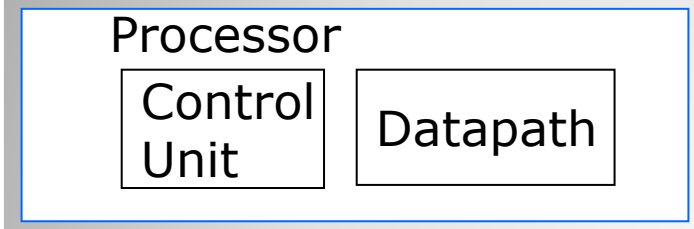
Refresh – 6-Instruction Processor



Pic of Processor Modules



You need to check this pic often when connecting multi-modules



Processor module should instantiate the following submodules:

```
ControlUnit.sv
DataPath.sv
```

Required input signals: Clk, Reset

Required output signals: IR_Out, PC_Out, State, NextState, ALU_A, ALU_B, ALU_Out

Combine Control Unit and DP in Processor Module and Test it!

Testing your Processor

- Test submodules from lowest level.
- Test your control unit module and ensure it works as expected
  - Initialize the ROM with translated instructions from Lab4
- Test your datapath module and ensure it works as expected
  - Initialize the RAM with given data in the project assignment
- Write the Processor by instantiating Control Unit module and Datapath module and wiring them together.
- Test the processor module by running runrtlFSM.do
- At this point you should be able to notice the state machine works but ALU_A, ALU_B, and ALU_out are all zeros.
- Check your memory.v file; revise it as shown on next page
- NOTE: if testing with ModelSim doesn't work it won't help by trying testing on DE2 board.

Inside the memory.v file

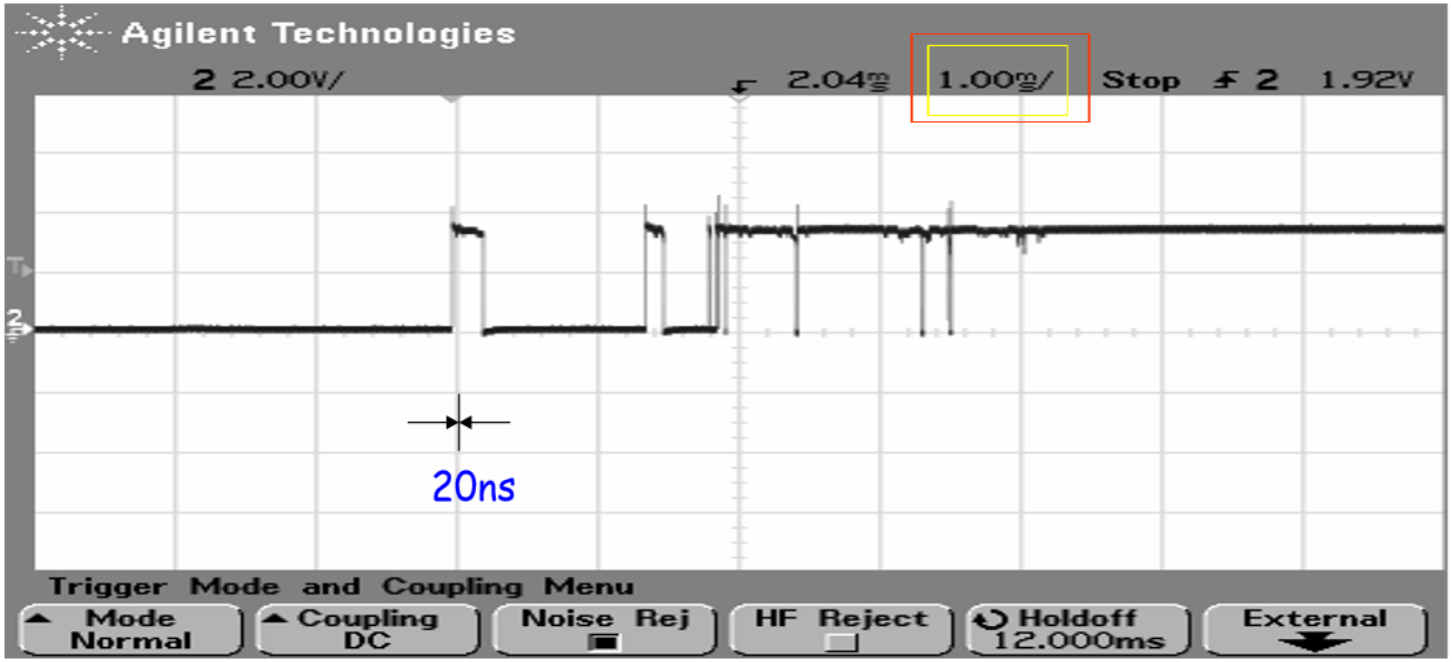
Here should be the mif file you created for your RAM

```
defparam
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
        altsyncram_component.init_file = "DataMemory.mif",
        altsyncram_component.intended_device_family = "Cyclone IV E",
        altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=YES,INSTANCE_NAME=DATA",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 256,
        altsyncram_component.operation_mode = "SINGLE_PORT",
        altsyncram_component.outdata_aclr_a = "NONE",
        altsyncram_component.outdata_reg_a = "UNREGISTERED",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.read_during_write_mode_port_a = "NEW_DATA_NO_NBE_READ",
        altsyncram_component.widthad_a = 8,
        altsyncram_component.width_a = 16,
        altsyncram_component.width_byteena_a = 1;
```

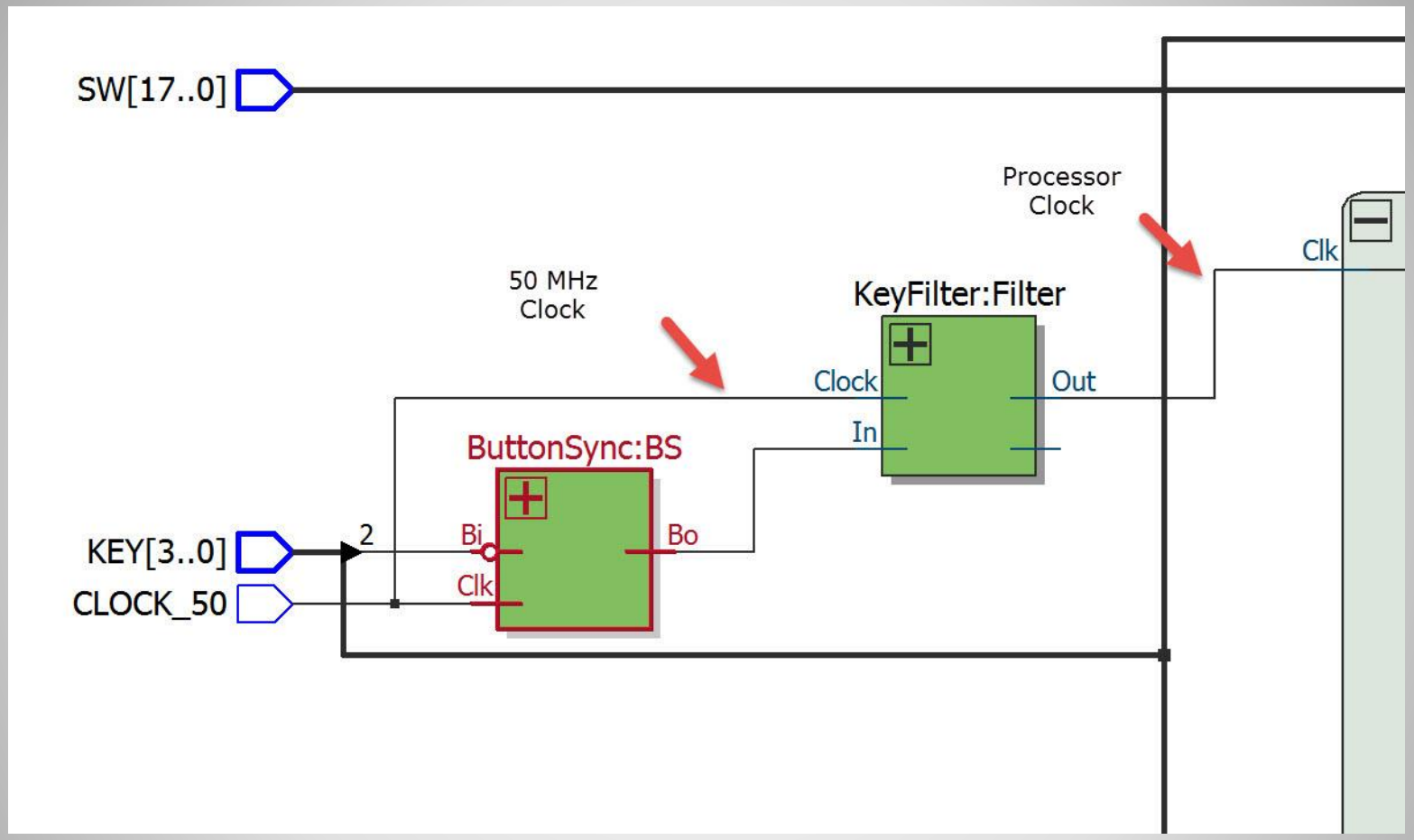
Here if showing “clock” change it to be “UNREGISTERED”

The Key Conditioner

Using ButtonSync and a Filter



Mechanical Switch



Key Conditioner

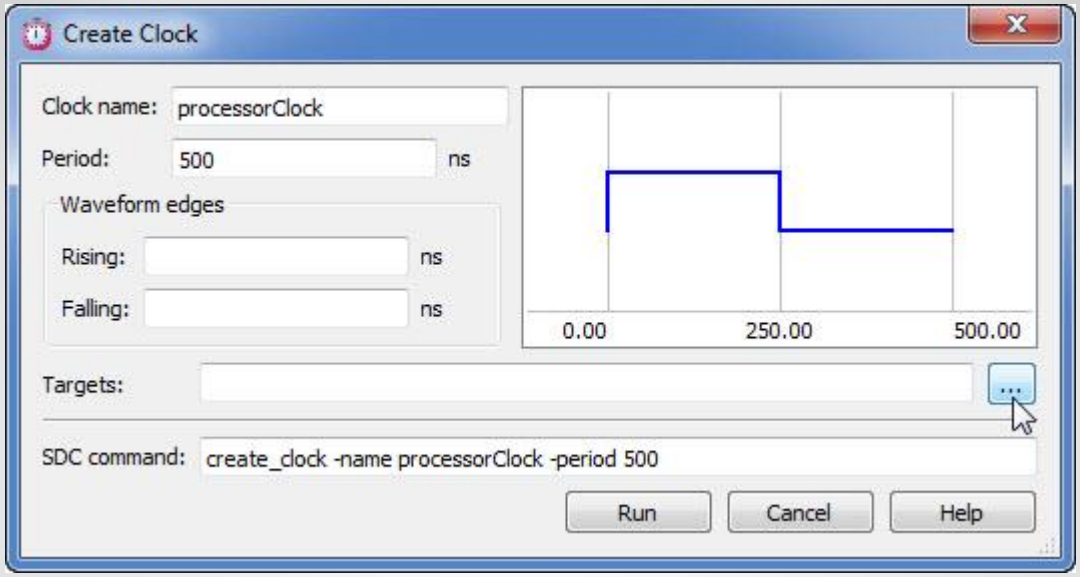
```
//TCES 330 Spring 2025
//The Key Filter follows a button synchronizer
//allows at most 10 output signals per second
//with a 50MHz clock

module KeyFilter (Clk, In, Out);
           input Clk, In;        //Clock and input signal of the system
           output logic Out;     //Output of the filter

           localparam DUR = 5_000_000 - 1;
           logic [32:0] Countdown = 0;

           always @(posedge Clk) begin

                      Out <= 0;  //initial output value
                      if(Countdown == 0) begin
                        if(In) begin
                            Out <= 1;
                            Countdown <= DUR;
                        end
                      end
                      else begin
                                Countdown <= Countdown - 1;
                      end
           end
endmodule
```

Allows at most 10 output signals per second (assuming a 50 MHz clock).

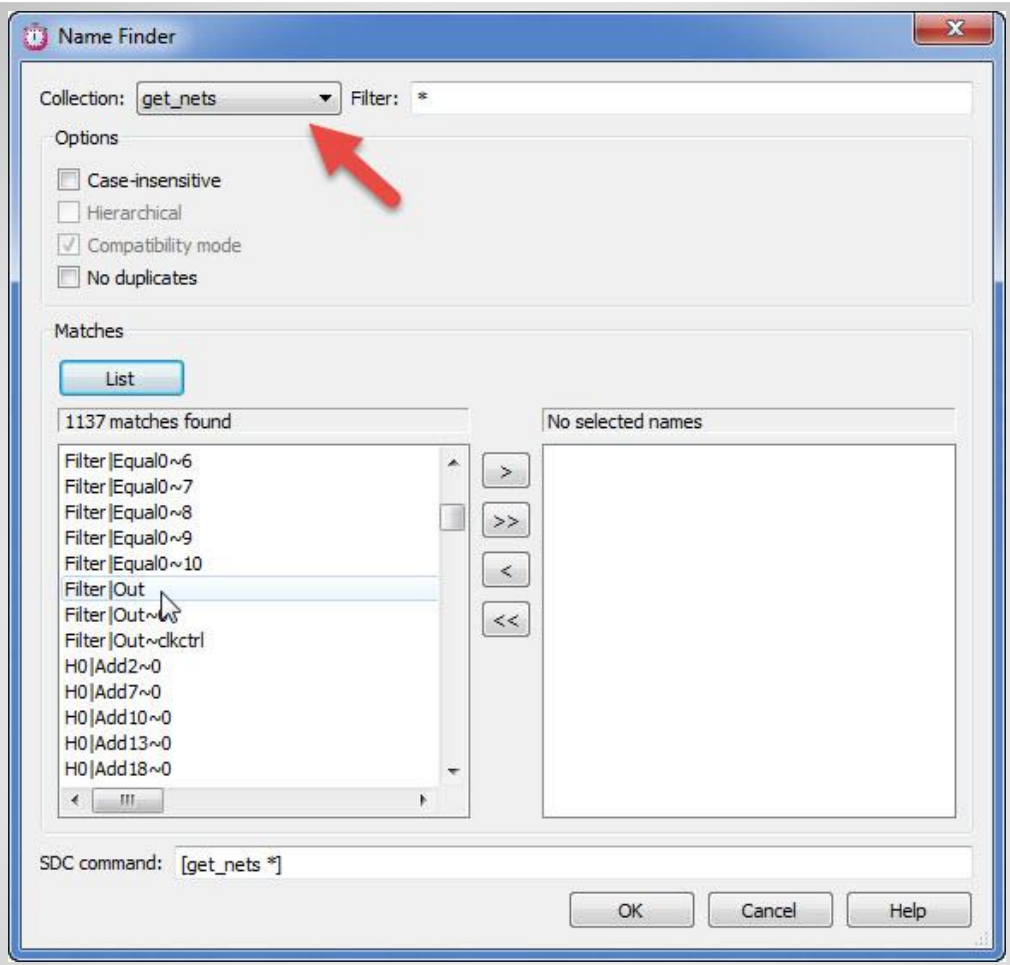
Key Filter

TimeQuest: Two Clocks

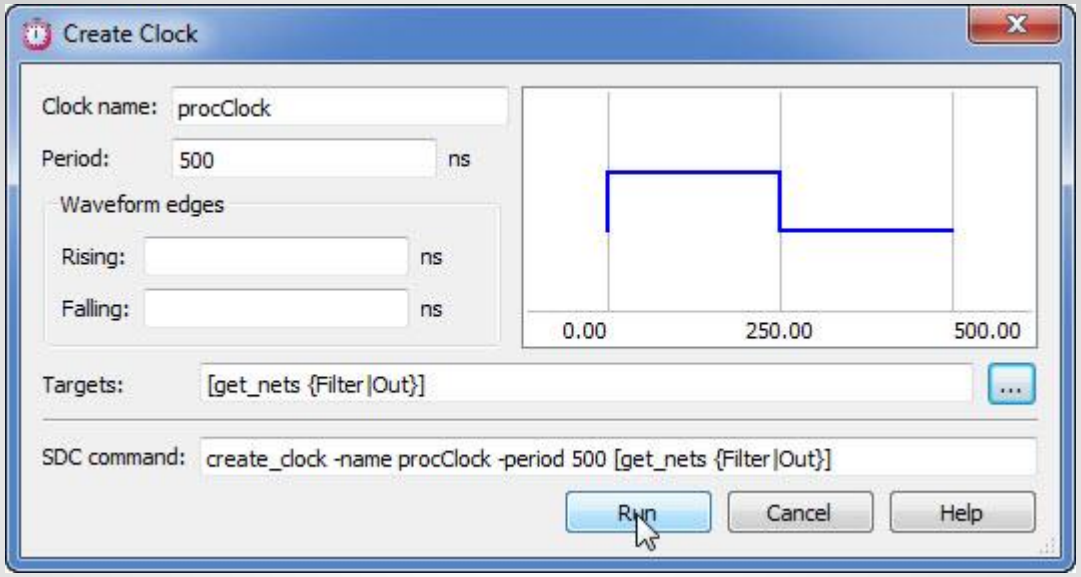
Processor clock (from the KEY) and the 50 MHz clock for the filter circuit



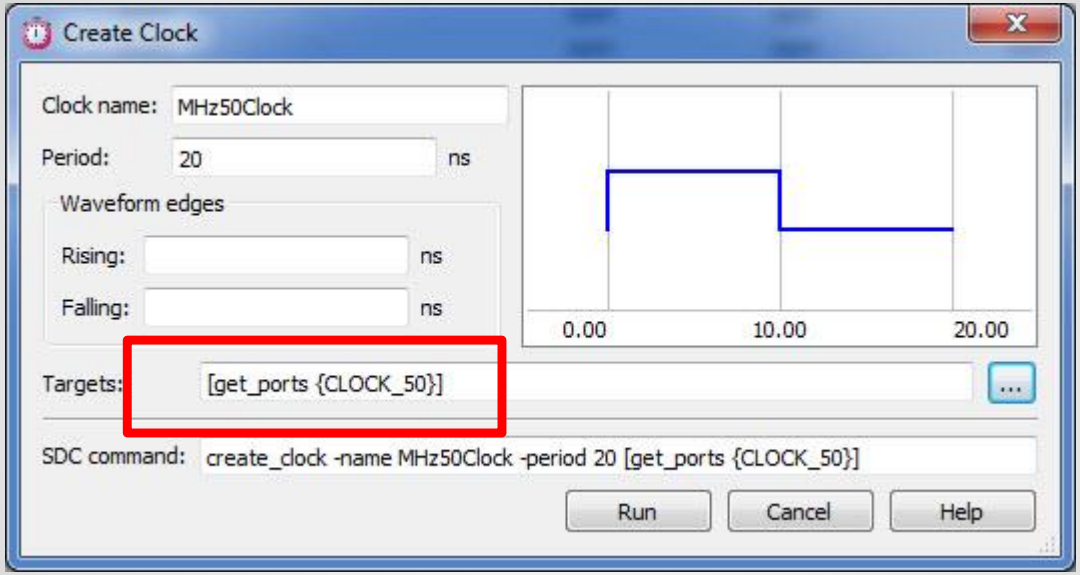
To Create Processor Clock



To Find the Processor Clock



Run to Create Processor Clock



Create 50 MHz Clock as Usual

```
37
38 #**************************************************************
39 # Create Clock
40 #**************************************************************
41
42 create_clock -name {sysClock} -period 20.000 -waveform { 0.000 10.000 } [get_ports {CLOCK_50}]
43 create_clock -name {buttonClock} -period 500.000 -waveform { 0.000 250.000 } [get_nets {Filter|Out}]
44
45
46 #**************************************************************
47 # Create Generated Clock
48 #**************************************************************
49
```

Your sdc File Should Show Two Clocks

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

create_clock -name {sysClock} -period 20.000 -waveform { 0.000 10.000 } [get_ports {CLOCK_50}]
create_clock -name {buttonClock} -period 500.000 -waveform { 0.000 250.000 } [get_nets {KF|Out}]
create_clock -name {altera_reserved_tck} -period 100.000 -waveform { 0.000 50.000 } [get_ports {altera_reserved_tck}]

#**************************************************************
# Create Generated Clock
#**************************************************************



#**************************************************************
# Set Clock Latency
#**************************************************************



#**************************************************************
# Set Clock Uncertainty
#**************************************************************

set_clock_uncertainty -rise_from [get_clocks {buttonClock}] -rise_to [get_clocks {buttonClock}]  1.000
set_clock_uncertainty -rise_from [get_clocks {buttonClock}] -fall_to [get_clocks {buttonClock}]  1.000
set_clock_uncertainty -fall_from [get_clocks {buttonClock}] -rise_to [get_clocks {buttonClock}]  1.000
set_clock_uncertainty -fall_from [get_clocks {buttonClock}] -fall_to [get_clocks {buttonClock}]  1.000
set_clock_uncertainty -rise_from [get_clocks {sysClock}] -rise_to [get_clocks {sysClock}]  1.000
set_clock_uncertainty -rise_from [get_clocks {sysClock}] -fall_to [get_clocks {sysClock}]  1.000
set_clock_uncertainty -fall_from [get_clocks {sysClock}] -rise_to [get_clocks {sysClock}]  1.000
set_clock_uncertainty -fall_from [get_clocks {sysClock}] -fall_to [get_clocks {sysClock}]  1.000
set_clock_uncertainty -rise_from [get_clocks {altera_reserved_tck}] -fall_to [get_clocks {altera_reserved_tck}]  1.000
set_clock_uncertainty -fall_from [get_clocks {altera_reserved_tck}] -rise_to [get_clocks {altera_reserved_tck}]  1.000
set_clock_uncertainty -rise_from [get_clocks {altera_reserved_tck}] -rise_to [get_clocks {altera_reserved_tck}]  1.000
set_clock_uncertainty -fall_from [get_clocks {altera_reserved_tck}] -fall_to [get_clocks {altera_reserved_tck}]  1.000
```

In Some Cases, .sdc File Needs to Include a Third Clock