Qsys System
Timing Wrapper Creator

Motivation

The slides that follow illustrate a methodology that can be used to verify and validate the timing requirements of arbitrary circuits within an FPGA fabric. The idea is that you have a circuit that you are targeting for a given FPGA fabric, and you wish to know how fast that circuit will operate in that fabric.

The easiest way to accomplish this is to create a test circuit that instantiates your circuit with a simple shift register structure wrapped around it such that all inputs to your circuit are fed by individual registers from the shift chain, and all or your circuit outputs feed back into the shift chain. If the shift chain is connected to an input pin and an output pin of the FPGA, and a clock and reset are derived from FPGA pins, then we can easily configure timing constraints to allow Quartus to place, route, and time our circuit of interest, with little overhead added by our shift chain.

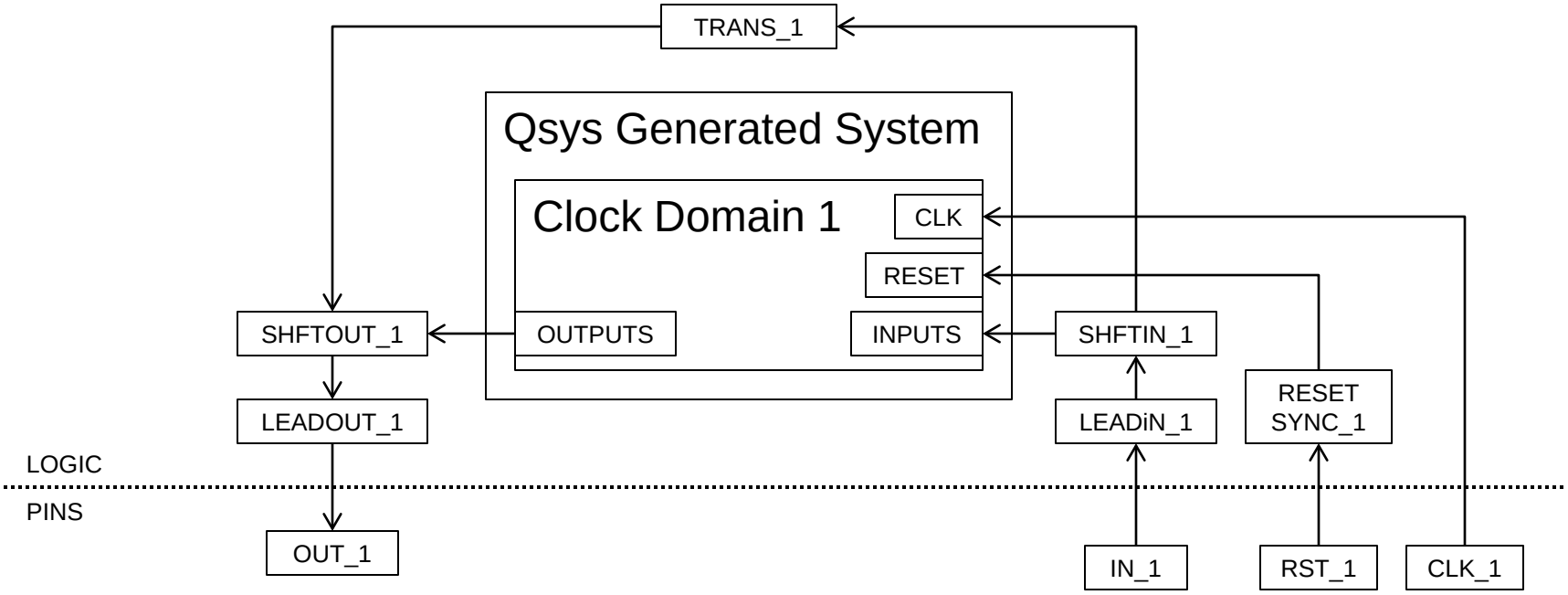
To make this flow easier, something that could be automated to a large degree, it's suggested that we package our circuit of interest within a Qsys system and use the generated output HDL from that Qsys system as the instance that we wrap with our shift chain. While this is not a requirement, it does make the overall flow a bit simpler for the user.

The results of this type of Fmax measurement can be leveraged when creating complex systems out of smaller building blocks. This method can be applied to the smaller blocks first to get an understanding of the Fmax limits inherent within them, and then as these smaller blocks are incorporated into larger and larger portions of the final system you can use this same technique to validate the intermediate subsystem configurations to ensure that you can maintain your target Fmax performance for the larger system.

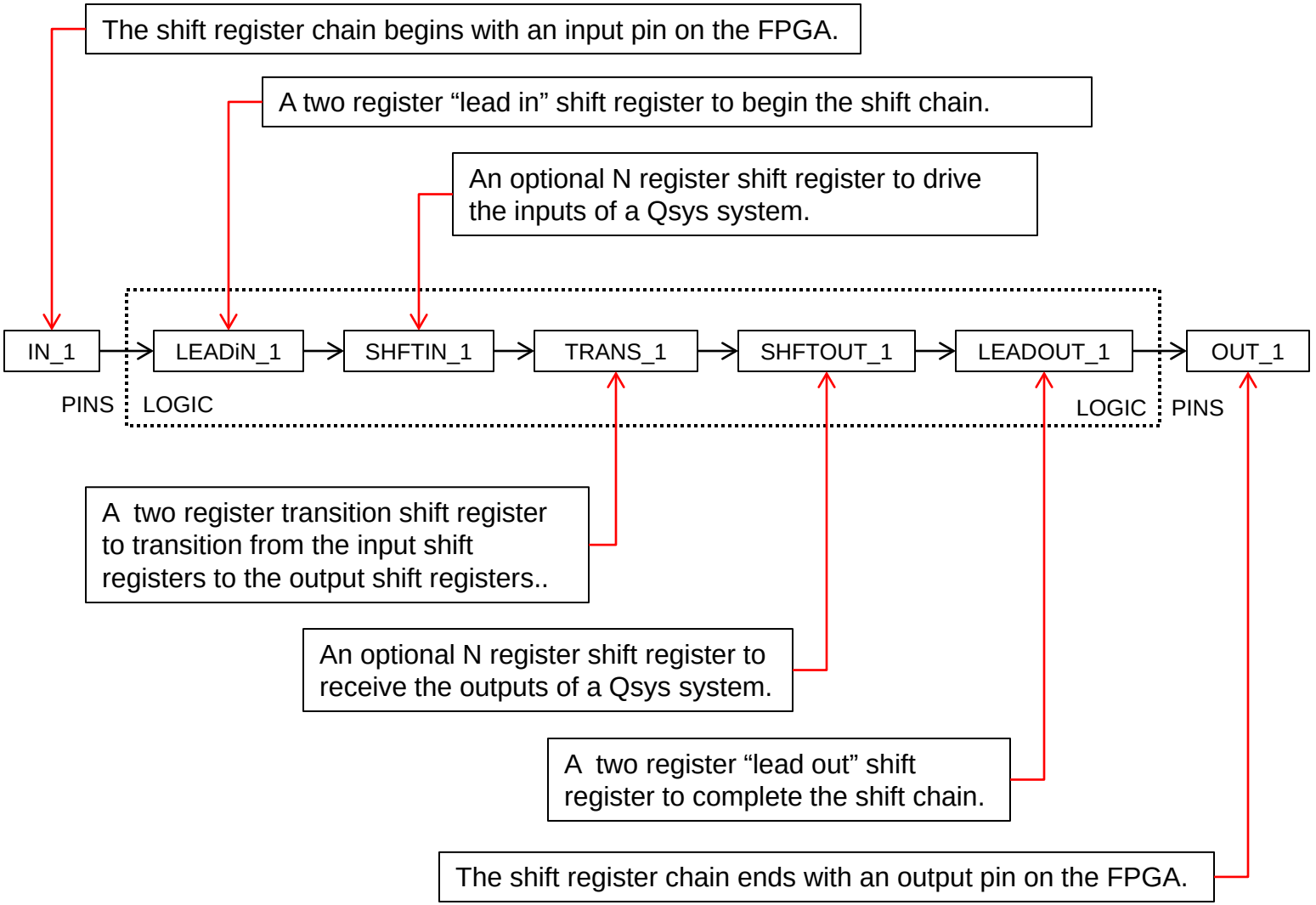
Concept

Wrap a shift register around each clock domain in the Qsys system.

As described earlier, we take our Qsys generated output and we instantiate that into a new top level HDL file and we wrap the shift register chain around that instantiation. We attach the input and output of the shift chain to physical FPGA pins, and we provide a clock and reset from FPGA pins as well. The clock that drives this shift chain must be the clock that drives the input and output logic of the system that it wraps so that all of the shift chain registers and the input and output registers in the system are all in the same clock domain.

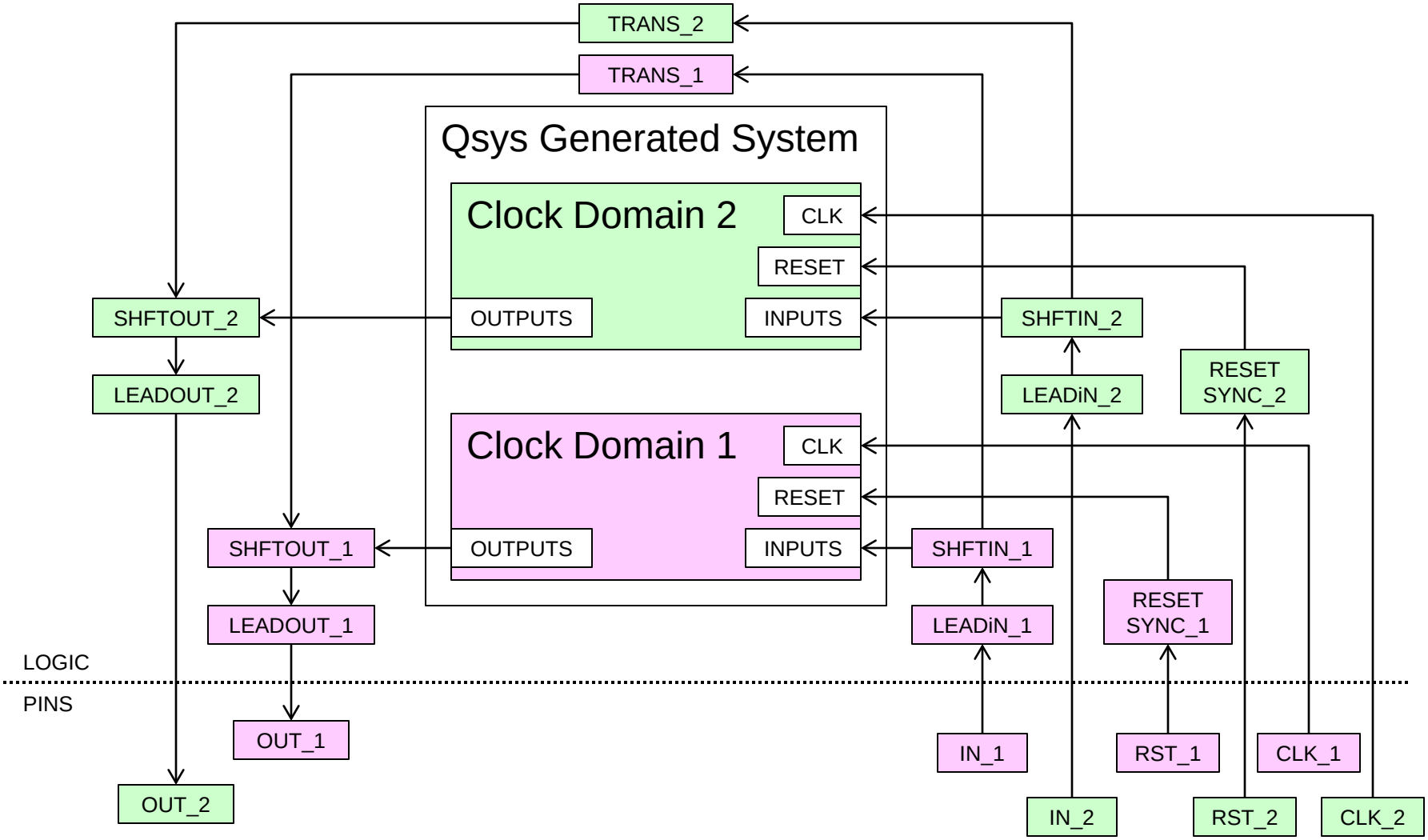


Break down of the shift register chain.



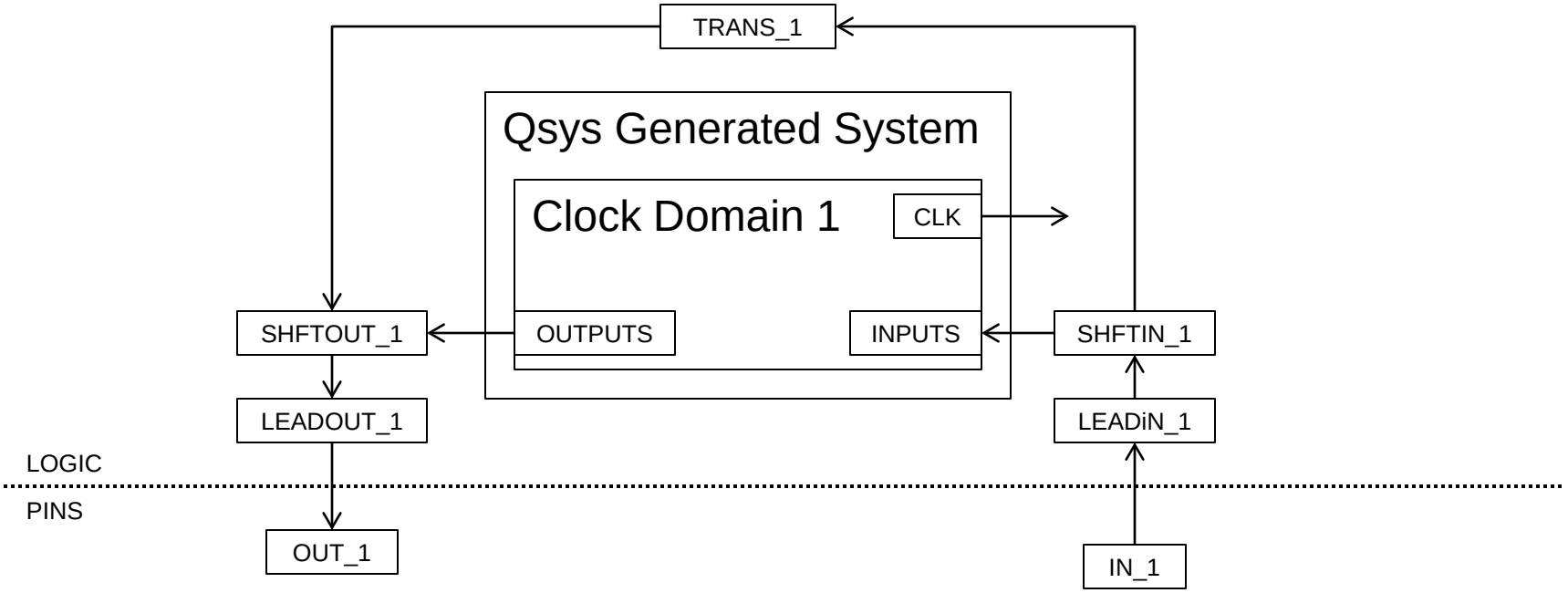
Multiple Clock Domains

We can deal with multiple clock domains by creating additional shift chains externally to attach to those additional clock domains.



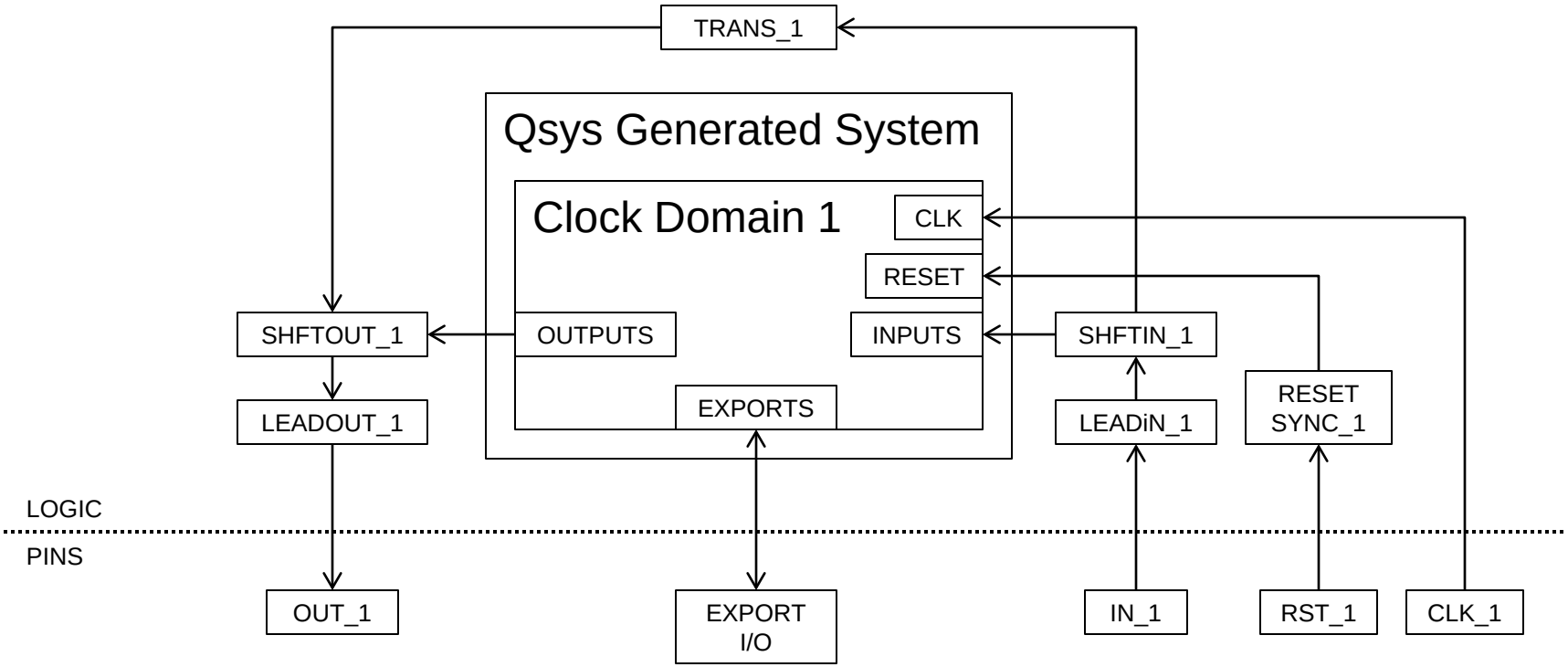
Clock domains sourced from the Qsys system.

In some configurations it also makes sense that the clock source for a given clock domain may come from the Qsys system itself. In this case there is no need to have a clock pin and reset pin associated with the shift chain, we can simply clock the shift chain logic with the clock provided by the Qsys system. Systems which contain embedded PLLs are likely to exhibit this configuration.



Dealing with asynchronous or external I/O requirements.

For circuits that have bidir ports or other asynchronous I/O to and from the test module, we simply export these ports to FPGA pins so that we can provide proper constraints to them for timing closure purposes.



Automated Implementation

createTimingWrapper.jar

A utility called createTimingWrapper.jar has been created to implement the concept described in the previous slides. You simply point createTimingWrapper.jar at the Qsys generated output directory and describe the ports that are exported as either clock domains or asynchronous and it will automatically wrap the Qsys system with a top level verilog module containing shift chains connected to the inputs and outputs of the targeted Qsys system.

The slides that follow contain information about the usage of createTimingWrapper.jar as well as a number of examples of how it can be applied. The examples demonstrate a single clock domain, multiple clock domains, asynchronous ports, and Qsys supplied clock domains. All of the examples discussed in these slides are also provided in source form along with the createTimingWrapper.jar utility.

The output of the createTimingWrapper.jar utility is a new project directory that contains a generated verilog file with a top level module that instantiates the Qsys system module and the shift chains that are wrapped around it, an SDC file that provides a few potential constraints for the new project, and a QIP file that includes references to the generated verilog and SDC files as well as the Qsys system QIP file. These three files can be quickly imported into a new Quartus project and used to generate timing closure for the Qsys system under test. The creation of such a Quartus project is left as an exercise for the user, as there may be various desires that the user expects to apply to the Quartus project environment which this utility cannot possibly know about. The examples that are provided with this utility demonstrate a few common methods for creating Quartus projects and leveraging the SDC file that is provided. Please read thru the build scripts that are provided with the examples to understand the flow and methods that are used to implement them.

Usage

bash-3.1$ ${QUARTUS_ROOTDIR}/bin64/jre/bin/java -jar createTimingWrapper.jar --help

USAGE: java alterawiki.utility.CreateTimingWrapper [options]

OPTIONS:

--help
Displays this usage screen.

--qsysGenerationDirectory <directoryPath>
This is the path to a Qsys generated output directory. Within this directory Qsys should have placed a directory called "synthesis". And within that synthesis directory Qsys should have created a qip file, an hdl file and a "submodules" directory. This program wraps the module defined in that hdl file with the timing shift register circuit. This program generates all of its output in the qsysGenerationDirectory in a directory called "timingWrapperProject". This program will not run if a "timingWrapperProject" directory already exists.

--clockDomain "<portList>"
The <portList> provided to this option is the list of HDL ports which belong to a single clock domain. This option must occur at least once but may occur more than once to define multiple clock domains. The order of ports in this list is specific, the first port must be the clock signal for this domain. If the direction of the clock signal is input, then the second port must be the reset signal associated with this clock domain, also an input. If the direction of the clock signal is output, then there is no reset signal associated with the clock domain. All additional input and output ports associated with this domain must follow. Bidir ports and asynchronous ports must not be listed in a clock domain.

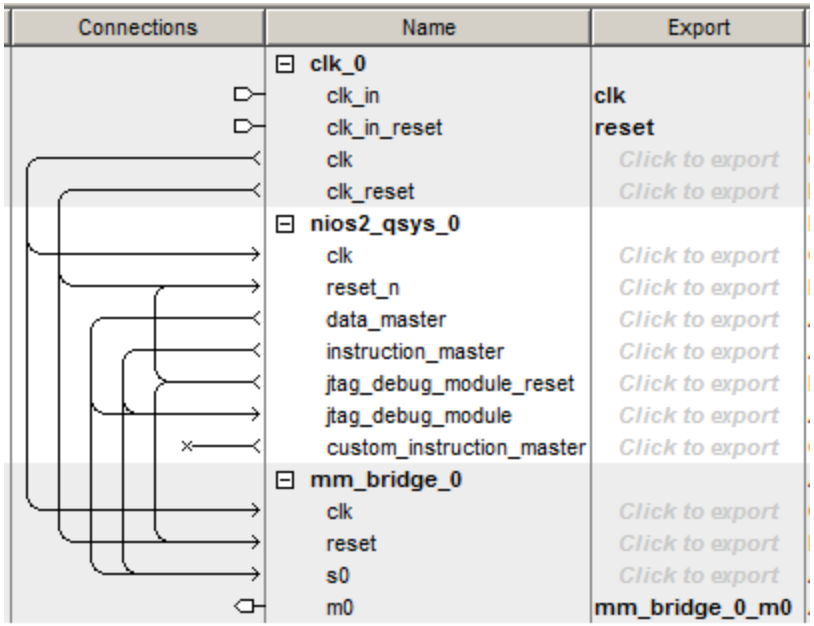
--asynchronousPorts "<portList>"
The <portList> provided to this option is the list of HDL ports which do not belong to any declared clock domain. This option is optional and may only occur once. Bidir ports are automatically exported from the timing wrapper asynchornously, so this list must only contain input and output ports.

--dumpPortsOnly
This option causes this program to only dump the port map of the module to be wrapped. The port map dump is formatted in such a way that you can cut and paste the ports into a clock domain option argument to be passed into this program again. When this option is used, only the --qsysGenerationDirectory option must be specified, all other options will be ignored.

Single Clock Domain Example

The Qsys system shown on the right exports the top level port map shown below the image. That port map is expressed to the createTimingWrapper.jar utility with the following arguments:

```
--clockDomain " \
    clk_clk \
    reset_reset_n \
    mm_bridge_0_m0_waitrequest \
    mm_bridge_0_m0_readdata \
    mm_bridge_0_m0_readdatavalid \
    mm_bridge_0_m0_burstcount \
    mm_bridge_0_m0_writedata \
    mm_bridge_0_m0_address \
    mm_bridge_0_m0_write \
    mm_bridge_0_m0_read \
    mm_bridge_0_m0_byteenable \
    mm_bridge_0_m0_debugaccess \
" \
```

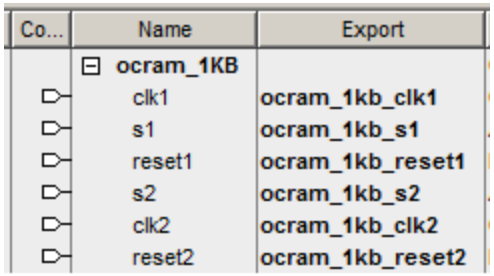



```
nios u0 (
    .clk_clk                       (),
    .reset_reset_n                 (),
    .mm_bridge_0_m0_waitrequest    (),
    .mm_bridge_0_m0_readdata       (),
    .mm_bridge_0_m0_readdatavalid  (),
    .mm_bridge_0_m0_burstcount     (),
    .mm_bridge_0_m0_writedata      (),
    .mm_bridge_0_m0_address        (),
    .mm_bridge_0_m0_write          (),
    .mm_bridge_0_m0_read           (),
    .mm_bridge_0_m0_byteenable     (),
    .mm_bridge_0_m0_debugaccess    ()
);
```

Multiple Clock Domain Example

The Qsys system shown on the right exports the top level port map shown below the image. That port map is expressed to the createTimingWrapper.jar utility with the following arguments:

```
--clockDomain " \
    ocram_1kb_clk1_clk \
    ocram_1kb_reset1_reset \
    ocram_1kb_s1_address \
    ocram_1kb_s1_chipselect \
    ocram_1kb_s1_clken \
    ocram_1kb_s1_readdata \
    ocram_1kb_s1_write \
    ocram_1kb_s1_writedata \
    ocram_1kb_s1_byteenable \
" \
--clockDomain " \
    ocram_1kb_clk2_clk \
    ocram_1kb_reset2_reset \
    ocram_1kb_s2_address \
    ocram_1kb_s2_chipselect \
    ocram_1kb_s2_clken \
    ocram_1kb_s2_readdata \
    ocram_1kb_s2_write \
    ocram_1kb_s2_writedata \
    ocram_1kb_s2_byteenable \
" \
```



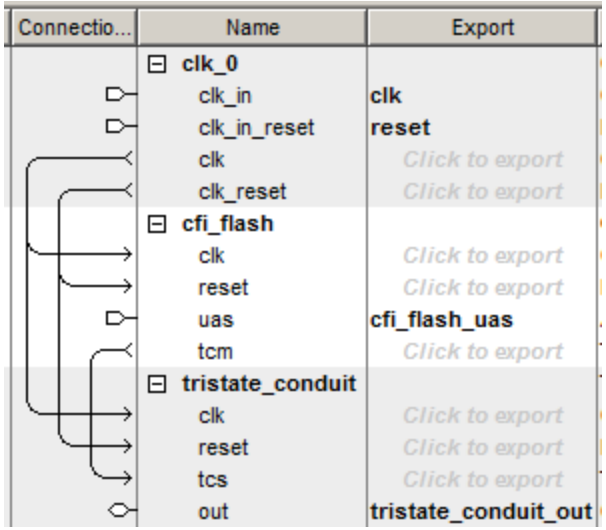
Co...	Name	Export
	ocram_1KB	
	clk1	ocram_1kb_clk1
	s1	ocram_1kb_s1
	reset1	ocram_1kb_reset1
	s2	ocram_1kb_s2
	clk2	ocram_1kb_clk2
	reset2	ocram_1kb_reset2

```
ocram_1KB u0 (
    .ocram_1kb_clk1_clk      (),
    .ocram_1kb_s1_address    (),
    .ocram_1kb_s1_chipselect (),
    .ocram_1kb_s1_clken      (),
    .ocram_1kb_s1_readdata   (),
    .ocram_1kb_s1_write      (),
    .ocram_1kb_s1_writedata  (),
    .ocram_1kb_s1_byteenable (),
    .ocram_1kb_reset1_reset  (),
    .ocram_1kb_s2_address    (),
    .ocram_1kb_s2_chipselect (),
    .ocram_1kb_s2_clken      (),
    .ocram_1kb_s2_readdata   (),
    .ocram_1kb_s2_write      (),
    .ocram_1kb_s2_writedata  (),
    .ocram_1kb_s2_byteenable (),
    .ocram_1kb_clk2_clk      (),
    .ocram_1kb_reset2_reset  ()
);
```

Asynchronous Ports Example

The Qsys system shown on the right exports the top level port map shown below the image. That port map is expressed to the createTimingWrapper.jar utility with the following arguments:

```
--clockDomain " \
    clk_clk \
    reset_reset_n \
    cfi_flash_uas_address \
    cfi_flash_uas_burstcount \
    cfi_flash_uas_read \
    cfi_flash_uas_write \
    cfi_flash_uas_waitrequest \
    cfi_flash_uas_readdatavalid \
    cfi_flash_uas_byteenable \
    cfi_flash_uas_readdata \
    cfi_flash_uas_writedata \
    cfi_flash_uas_lock \
    cfi_flash_uas_debugaccess \
" \
--asynchronousPorts " \
    tristate_conduit_out_tcm_address_out \
    tristate_conduit_out_tcm_read_n_out \
    tristate_conduit_out_tcm_write_n_out \
    tristate_conduit_out_tcm_data_out \
    tristate_conduit_out_tcm_chipselect_n_out \
" \
```

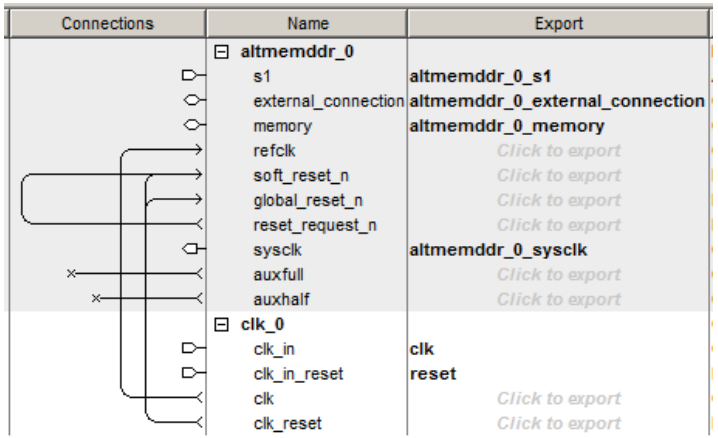



```
cfi_flash u0 (
    .cfi_flash_uas_address                    (),
    .cfi_flash_uas_burstcount                 (),
    .cfi_flash_uas_read                       (),
    .cfi_flash_uas_write                      (),
    .cfi_flash_uas_waitrequest                (),
    .cfi_flash_uas_readdatavalid              (),
    .cfi_flash_uas_byteenable                 (),
    .cfi_flash_uas_readdata                   (),
    .cfi_flash_uas_writedata                  (),
    .cfi_flash_uas_lock                       (),
    .cfi_flash_uas_debugaccess                (),
    .clk_clk                                  (),
    .reset_reset_n                            (),
    .tristate_conduit_out_tcm_address_out     (),
    .tristate_conduit_out_tcm_read_n_out      (),
    .tristate_conduit_out_tcm_write_n_out     (),
    .tristate_conduit_out_tcm_data_out        (),
    .tristate_conduit_out_tcm_chipselect_n_out ()
);
```

Qsys Generated Clock Domain Example

The Qsys system shown on the right exports the top level port map shown below the image. That port map is expressed to the createTimingWrapper.jar utility with the following arguments:

```
--clockDomain " \
    clk_clk \
    reset_reset_n \
" \
--clockDomain " \
    altmemddr_0_sysclk_clk \
    altmemddr_0_s1_address \
    altmemddr_0_s1_write \
    altmemddr_0_s1_read \
    altmemddr_0_s1_beginbursttransfer \
    altmemddr_0_s1_waitrequest_n \
    altmemddr_0_s1_readdata \
    altmemddr_0_s1_readdatavalid \
    altmemddr_0_s1_writedata \
    altmemddr_0_s1_byteenable \
    altmemddr_0_s1_burstcount \
" \
--asynchronousPorts " \
    altmemddr_0_memory_mem_odt \
    altmemddr_0_memory_mem_clk \
    altmemddr_0_memory_mem_clk_n \
    altmemddr_0_memory_mem_cs_n \
    altmemddr_0_memory_mem_cke \
    altmemddr_0_memory_mem_addr \
    altmemddr_0_memory_mem_ba \
    altmemddr_0_memory_mem_ras_n \
    altmemddr_0_memory_mem_cas_n \
    altmemddr_0_memory_mem_we_n \
    altmemddr_0_memory_mem_dq \
    altmemddr_0_memory_mem_dqs \
    altmemddr_0_memory_mem_dm \
    altmemddr_0_external_connection_local_refresh_ack \
    altmemddr_0_external_connection_local_init_done \
    altmemddr_0_external_connection_reset_phy_clk_n \
" \
```




```
ddr2 u0 (
    .clk_clk                                        (),
    .reset_reset_n                                  (),
    .altmemddr_0_memory_mem_odt                     (),
    .altmemddr_0_memory_mem_clk                     (),
    .altmemddr_0_memory_mem_clk_n                   (),
    .altmemddr_0_memory_mem_cs_n                    (),
    .altmemddr_0_memory_mem_cke                     (),
    .altmemddr_0_memory_mem_addr                    (),
    .altmemddr_0_memory_mem_ba                      (),
    .altmemddr_0_memory_mem_ras_n                   (),
    .altmemddr_0_memory_mem_cas_n                   (),
    .altmemddr_0_memory_mem_we_n                    (),
    .altmemddr_0_memory_mem_dq                      (),
    .altmemddr_0_memory_mem_dqs                     (),
    .altmemddr_0_memory_mem_dm                      (),
    .altmemddr_0_s1_address                         (),
    .altmemddr_0_s1_write                           (),
    .altmemddr_0_s1_read                            (),
    .altmemddr_0_s1_beginbursttransfer              (),
    .altmemddr_0_s1_waitrequest_n                   (),
    .altmemddr_0_s1_readdata                        (),
    .altmemddr_0_s1_readdatavalid                   (),
    .altmemddr_0_s1_writedata                       (),
    .altmemddr_0_s1_byteenable                      (),
    .altmemddr_0_s1_burstcount                      (),
    .altmemddr_0_sysclk_clk                         (),
    .altmemddr_0_external_connection_local_refresh_ack (),
    .altmemddr_0_external_connection_local_init_done   (),
    .altmemddr_0_external_connection_reset_phy_clk_n   ()
);
```