Tester

-Reconfigurable clock domain

The design under test (DUT) is isolated from the clock domain of the main board, since various designs may operate on different clock frequencies. In order to realize this, we used a dynamically reconfigurable Phase Lock Loop (PLL) to realize the separately reconfigurable clock domain without influencing the clock of the main board.

-Synchronizing with FIFOs

The data integrity between the two clock domains are ensured by Altera IP core Dual-Clock First-In-First-Out (DCFIFO) megafunction, which is widely used for data buffering in asynchronous clock domains. In the DCFIFO, the read and write signals are synchronized to the rdclk and wrclk clocks respectively.

When the wrfull port is low, assert wrreq to request for a write operation. Input data synchronized by wrclk, is hold in the stack when the wrreq is high. When the rdempty port is high, insert rdreq for a reading request. Then the data will be exhibited from the q port. aclr is used for clearing the data stored in the DCFIFO. The figures bellow shows the timing and behaviour of the write and read process.

-Test controller

Mem_if

'mem_if' arbitrates the access to the Avalon memory between 'stim' and 'check' module. Altera Avalon is the standard interface of Altera devices, allowing the design to access system resources (e.g. SRAM) on the DE2 board. The address-based read/write Interface Avalon Memory Mapped Interface (Avalon-MM) is used here. The read require from 'stim' is prior to the write require of memory from 'check'. Since the tester can run massive amount of tests in a very short time, the address-based Avalon-MM can ensure the test vectors and result vectors correspond to each other in the memory.

Stim

The 'stim' module reads test vector, command information, PLL reconfiguration data from the memory, and then sends the information to 'check' module and 'dut_if' module. The data to 'check' is through 'cfifo' (don't-care bits, result vector, address) and 'sc_data' (bit mask), while to 'dut_if' is through 'sfifo' (for test vectors) and 'dififo' (for request, command and trigger/clock pin mask). The 'stim' module is also responsible for sending the target select data to the 4-16 decoder for powering

up single design and sending counters configuration data to the PLL reconfiguration module and controlling the reconfiguration process.

The bit mask is used to eliminate the pins that are not included in the functional outputs (part of the physical chip; not part of the design, however), whose value does not influence the result. In some specific conditions, some pins are part of the design; however, their status is not part of the result, such as a 'busy' or a flag signal. Such pins are covered by the 'don't-care bits'.

The request types are read in the READ_META state. Depending on different request types, the 'stim' responds to the according tasks as follows:

+ setup bit mask (SETUP_BITMASK). In this task the bit mask and relative command type are sent to the 'check' module through a register output, bypassing the FIFO between 'stim' and 'check'.

+ read test vector (READ_TV, WR_FIFPOS). The test vecotor is firstly read from memory in serial, then sent to the 'dut_if' for the input of the design and to the 'check' for reference through FIFOs in this task.

+ send command/data to dut_if (SEND_DICMD, WR_DIFIFO). This task is used for assigning clock pins to the DUT inputs and setting up trigger pins to the DUT output pins. These are explained in the dut_if section.

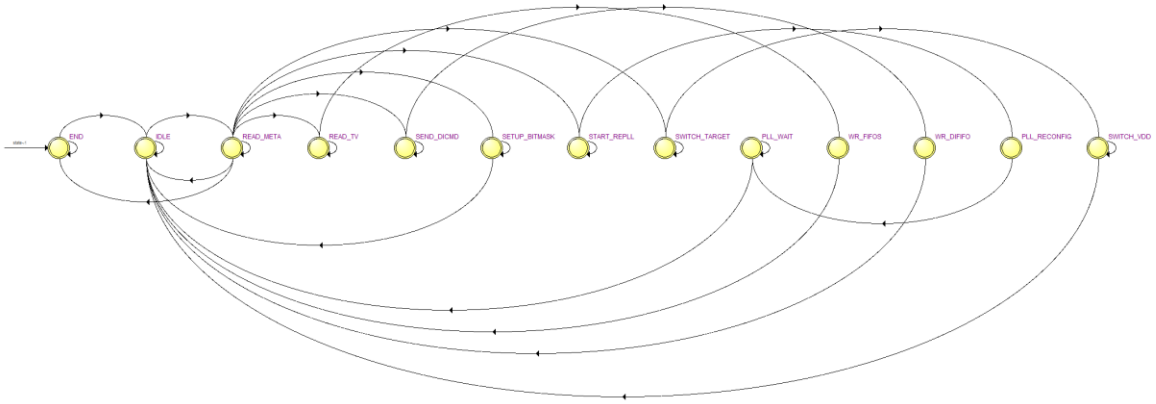
+switch design target (SWITCH_TARGET, SWITCH_VDD). As introduced previously, there are 16 designs at most on a single chip under test. Only one can be selected by powering up its exclusive VDD pin. In this task, a 4 bit signal indicating which design is selected is sent to the testing board. On the B1 board, a 4-16 decoder will decode the information and integrated switches array will turn on only one design.

+start reconfiguring the dynamic PLL (START_REPLL, PLL_RECONFIG, PLL_WAIT). In this task, the parameters for the PLL counters are read from the memory first. Then in the PLL_RECONFIG state a

trigger will be sent to the PLL_INTERFACE module with those parameters to start the configuration process. Then the PLL_WAIT state will halt the 'stim' until the new PLL output frequency is stable.

After each task is done, the 'stim' returns to the IDLE state, waiting for the next request. When a complete test is finished, the 'stim' goes to an END state and resets itself to the start IDLE state.

The state transmittion chart is as shown in the figure below.



States Behaviour Table

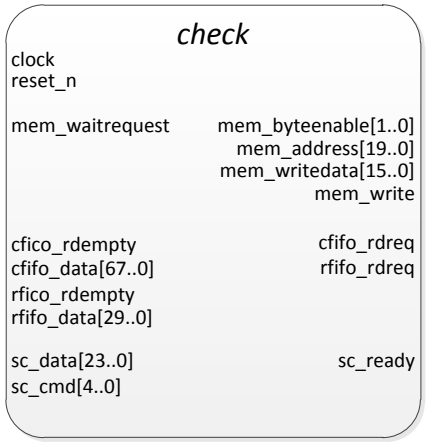
State	Behaviour
IDLE	reset variables (words_stored, read_requested)
READ_META	read metadata from memory
READ_TV	read test vector from memory
SWITCH_TARGET	read target information from memory and store in new_target_sel
SWITCH_VDD	change target power based on the target_sel data (updated by new_target_sel)
WR_FIFOS	send write request to FIFOs by sfifo_wrreq, cfifo_wrreq
SETUP_BITMASK	read bitmask from memory, send sc_cmd, sc_data to check module
SEND_DICMD	read command data for DUT_if
WR_DIFIFO	send write request to DUT_if through dififo
START_REPLL	read data for PLL reconfiguration from memory
PLL_RECONFIG	send trigger to PLL to start the reconfiguration process
PLL_WAIT	put the stim waiting until the PLL clock is stable
END	when all the FIFOs are empty, send done signal to upper level

Main states transitions table

Source State	Destination State	Condition
END	IDLE	sfifo_wrempty && cfifo_wrempty && enable
IDLE	READ_META	~sfifo_wrfull && ~cfifo_wrfull && ~mem_waitrequest
READ_META	END	words_stored && req_type==REQ_END
READ_META	IDLE	words_stored && req_type==REQ_PLLRECONFIG
READ_META	READ_META	

READ_META	SWITCH_TARGET	words_stored && req_type==REQ_SWITCH_TARGET
READ_META	READ_TV	words_stored && req_type==REQ_TEST_VECTOR
READ_META	SEND_DICMD	words_stored && req_type==REQ_SEND_DICMD
READ_META	SETUP_BITMASK	words_stored && req_type==REQ_SETUP_BITMASK
READ_TV	WR_FIFOS	words_stored == tv_len
SEND_DICMD	WR_DIFIFO	words_stored == 3 && ~dififo_wrfull && sfifo_wrempty && cfifo_wrempty
SETUP_BITMASK	IDLE	words_stored == 3
START_REPLL	PLL_RECONFIG	words_stored == 3 && pll_locked
PLL_RECONFIG	PLL_WAIT	
PLL_WAIT	IDLE	pll_stable
SWITCH_TARGET	SWITCH_VDD	sfifo_wrempty && cfifo_wrempty
SWITCH_VDD	IDLE	waitcnt == 0
WR_DIFIFO	IDLE	
WR_FIFOS	IDLE	

Check



The ‘check’ module compares the test result from ‘dut_if’ (through ‘rfifo’) and the reference result from ‘stim’ (through ‘cfico’) and decides whether the test is passed or failed. The information is then stored in metadata. The test result vector is also written into the memory by this module. As previously introduced, the test result vector and the reference result vector are firstly processed by the bitmask and don’t-care bits before comparing.

-dut_if

The ‘dut_if’ module is the interface between the VLSI system and the design under test (DUT). It uses the data from ‘stim’ to generate the test vector and send it to the DUT. After certain conditions are satisfied, the result is valid from the DUT, which is then sent to the check through ‘rfifo’.

The test vector is merged with clock pins before being sent to the DUT. The clock is from the reconfigurable clock domain. This process is, however, not necessary in every test. It depends on whether the design contains a clock input and whether the user defines it in the input data. A clock gating is also done in this part for optimizing power consumption.

In deciding whether the result is valid, the 'dut_if' supports two modes:

1) With a predefined clock cycle number (in metadata), the dut_if sets up a counter to wait until the clock cycles are reached. Then the result is considered valid. If the clock cycle number is 0, the result is taken within the same clock cycle, which corresponds to a combinational logic circuit. Otherwise, the DUT should be a sequential design with a fixed clock cycle to output the valid result.

2) With a set of pins which is defined as the trigger (e.g. a 'done' or 'ready' signal). The result is considered valid when the triggers change. This corresponds to asynchronous handshake behaviour. This is implemented by the trigger mask.

Those different modes of operation are set by the user in the metadata, depending on specific designs.

The speed of the 'dut_if' is boosted by pipelining technique, including fetch, execute and writeback stages. The fetch stage checks whether the 'sfifo' is empty and decide if the process should go to the next stage. The execute stage handles the testing process from sending the test vector to receiving the result vector. After a execute stage, the writeback stage is in place to write the data (result vector, metadata) to the memory. The pipelining enables the 'dut_if' to process the different stages of three tasks at the same time.

- PLL_INTERFACE

The reconfigurable clock domain uses the Altera IP cores ALTPLL and ALTPLL_RECONFIG to realize the dynamically reconfigurable clock. The ALTPLL generates stable clock. The input clock frequency is divided and multiplied by the internal counters, resulting in a variable output frequency. The parameters are initialized by a scan chain. Reinitializing the PLL with a new scan chain is possible and can change the output frequency with different parameters. However the scan chain contains too much information that is not expected to change during our process (counters that are not changed, current, phase information, etc.). ALTPLL_RECONFIG module can change specific parameters based on a complete scan chain and reinitialize the ALTPLL.

The ALTPLL_RECONFIG module requires input to be in serial and specific timing requirements. The process is coordinated by REPLL_CONTROL, responding to an input trigger, transferring the parallel input into serial, and offering a stable signal when the reconfiguration is done. All these modules are enveloped in PLL_INTERFACE.

The counters used here are M (feedback multiplier counter), N (pre-divider counter) and C (post-divider counter) counters. The output frequency = input frequency * M/(N*C). Currently the established minimum output frequency is 0.005x the input clock frequency, which is 100MHz of the DE2 board by default. The maximum can reach 10x the input clock, and theoretically can be increased. However, further testing is not done since 1GHz is already big enough for our hardware. Should further testing require lower frequency (<500KHz), another cascaded PLL_INTERFACE module could be implemented as a possible solution.

The formats of the data for different requests used in this part of the design are listed below:

Test vector:

0	8	16	31
metadata		input vector[0]	
input vector[1..2]			
output vector[0..1]			
output vector[2]		metadata2	
x mask[0..1]			
x mask[2]		padding	

Change target:

0	8	16	31
metadata		design number	
padding			
padding			

Change bitmask

0	8	16	31
metadata		bitmask[0]	
bitmask[1..2]			
padding			

Send dicmd

0	8	16	31
metadata		payload [0]	
payload[1..2]			
padding			

PLL reconfiguration

0	8	16	31
metadata		multiply factor	
divide factor		post-divide factor	
padding			

Mem end

0	8	16	31
metadata		unused	
padding			
padding			

Metadata:

0	1	2	3	4	5	6	7

request type	command type

Metadata 2:

0	1	2	3	4	5	6	7
test run	cycle count					test failed	unused