Partial Reconfiguration

A script is provided to automate the generation of blanking and partial bitstreams for partial reconfiguration using Xilinx Vivado.

The user of this script is responsible for generating Verilog modules for: - A Verilog top level module - Verilog for each Reconfigurable Module (RM) - An `.xdc` file

The user must also have an idea of how large the above modules are (i.e., what is the FPGA footprint) in order to create appropriately-sized Reconfigurable Partitions (RP). These sizes are reflected in the `.xdc` file.

`design_complete.tcl`

This script was designed in accordance with Xilinx User's Guide 947: Partial Reconfiguration for Vivado

Requirements:

- Vivado 2014.1+
- Xilinx Partial Reconfiguration license
- 7-Series FPGA
  - The script ships with the variant of the Xilinx Artix-7 family on the Digilent Basys3 as the default, but it can be easily modified

File System Hierarchy

The script requires the directory structure oulined below. A populated file structure is provided in the "*Partial_Reconfiguration" directory. Note that a file called "_file*" exists in what should be empty directories, as github will not allow committing empty directories.

Bitstreams - Create this folder and leave it empty. This is where you will find your bitstreams once the script has run

Implement - This folder will store checkpoints created in the process of executing the script

Sources - This folder will contain your design files in the following subfolders:

*Sources**hdl* - Create a separate folder (\Sources\hdl\rm1, \Sources\hdl\rm2, \Sources\hdl\top, etc.) for each reconfigurable module as well as a separate folder for your top level module.

*Sources**xdc* - Place your `.xdc` file here

Synth - This folder will contain all post-synthesis checkpoints

Tcl - This folder should contain all subscripts invoked by `design_complete.tcl`. **DO NOT MODIFY THESE SCRIPTS UNLESS YOU REALLY KNOW WHAT YOU'RE DOING!!** These scripts include:

- design_utils.tcl
- eco_utils.tcl
- hd_floorplan_utils.tcl
- impl.tcl
- impl_utils.tcl
- log.tcl
- ooc_impl.tcl
- pr_impl.tcl
- run.tcl
- step.tcl
- synth.tcl
- synth_utils.tcl

Changing Hardware Target

Change the device, package and speed-grade of the target device in the section of code below, just as you would in a project-mode GUI.

```
##################################################################
### Define Part, Package, Speedgrade
##################################################################
set device       "xc7a35t"
set package      "cpg236"
set speed        "-1"
set part         $device$package$speed
```

File Naming

Top-Level Module

- The default name for the top-level module is `top.v`. This can be easily modified though one line of code, found below:

```
set top "top"
```

Reconfigurable Partitions

- A naming scheme for all reconfigurable partitions is established in the top-level module
- This is accomplished by instantiating an instance of a module specified in the `design_complete.tcl` script.
  - Example:

```
#design_complete.tcl
##########################################################################
### RP Module Definitions
##########################################################################
set module1 "shift"
```

```
set module1_variant1 "shift_right"
set variant $module1_variant1

...

set module1_inst "inst_shift"
```

- The reconfigurable partition "*shift*" would be instantiated in the `top.v` file by instantiating a "*shift*" module with an instance name of "*inst_shift*" like so:

```
\\top.v
\\ instantiate RP shift
shift inst_shift (
    en      (rst),
    clk     (gclk),
    addr        (count[34:23]),
    data_out    (shift_out)
);
```

Invoking the Script

To run the script simply `cd` into the project directory where the `design_complete.tcl` script is located. Run the following commands

```
$ vivado -mode tcl
% source design_complete.tcl
```

The above two commands will go complete the entire process from RP definition, synthesis, out of order synthesis of RMs, P&R to bitstream generation.

Setting Flow Control

Should the design fail anywhere in the synthesis-to-bitstream process, it is possible to fix the issue and resume at the point at which the process failed. To skip any portion of the process, simply set the attribute associated with that particular function to `0` in the codeblock below.

```
####flow control
set run.topSynth       1
set run.rmSynth        1
set run.prImpl         1
set run.prVerify       1
set run.writeBitstream 1
```

LCD Displaying Color Data

Devices

- Basys 3
- Digilent PmodCLS
- RGB Color Sensor

Hardware Connections

- Green Tx line from color sensor is connected to port K17 and N17 (JC[1] and JC[3]), simulating two color sensors
- Color sensor red VCC line takes 3.3V
- Jumper (JP2) on PmodCLS should be set to 011 (0 meaning short, 1 meaning open) to work in SPI mode
- J1 on PmodCLS should be connected accordingly to JA[0:6] on Basys3

Verilog Modules

Async.v

- Forked from fpgafun.com
- Set baud rate to 38400 to read data

LCD.v

- Top-level module for displaying data on LCD
- Forked from Digilent.com
- Submodules:
  - command_lookup.v
  - master_interface.v
  - spi_interface.v

command_lookup.v

- Data from color sensors are transferred to "**reg [7:0] command[0:45]" via "wire [7:0] RxD_data"** in top module
- There are two logic sections handling the process of data buffering
- The buffer (**reg [7:0] command[0:45]**) contains both format control and data. If the size of it changes, the iteration "**count_sel**" in "**stateWaitRun**" in master_interface.v also need to be changed accordingly

master_interface.v

- A finite state machine looping through the buffer in command_lookup.v and transfer the data to spi_interface.v through send_data.
- Receiving control signal from **SW[1]=clear**, **SW[2]=start**, **SW[3]=continue display**
- State will go through:

stateIdle -> stateDisplay -> stateWaitRun -> stateWaitSS -> stateFinished ->(go back to stateIdle when posedge@start OR **highlevel@continue_display)**

OR

stateIdle -> stateClearing -> stateWaitRun -> stateWaitSS -> stateFinished ->

spi_interface.v

- Unchanged from basecode
- Send data to LCD using through SPI