Dynamic Partial Reconfiguration

Objectives

- Implement a project that can be dynamically reconfigured using the ZYBO (or ZYBO Z7-10) Board.
- Learn the Partial Reconfiguration (PR) TCL-based non-project flow from HDL to bitstream (Vivado 2019.1)

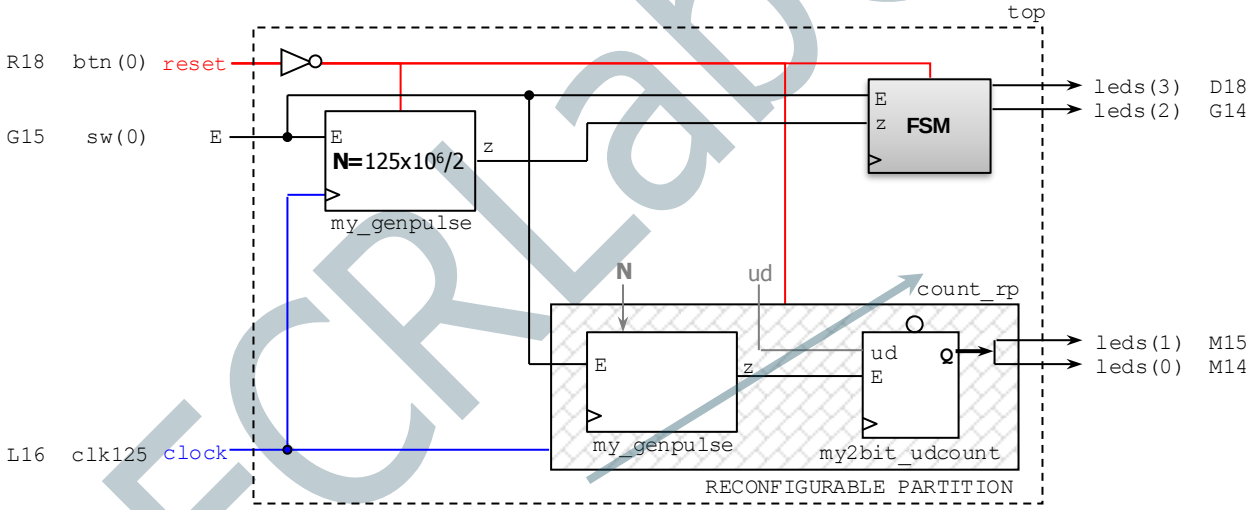
Vivado Partial Reconfiguration - Documentation

- UG909: Vivado Design Suite User Guide – Partial Reconfiguration.
- UG947: Vivado Design Suite Tutorial – Partial Reconfiguration. You can follow this for the Xilinx-provided `ug947-vivado-partial-reconfiguration-tutorial.zip` file (this is a Verilog design for the KC705 demonstration board)

Test Project – 1 RP

- **LED pattern control:** The circuit, written in VHDL, controls the pattern on the `leds(3..0)` signal.
  - ✓ `leds(3..2)`: It is controlled by a state machine that switches between `10` and `01`. The rate of change is controlled by the `my_genpulse` circuit that issues an enable pulse of 1 cycle every 0.5 s (1 cycle = 8 ns in ZYBO Board)
  - ✓ `leds(1..0)`: It is controlled by a 2-bit updown counter whose rate of change is controlled by the parameter `N` (the `my_genpulse` circuit issues an enable pulse of 1 cycle every N cycles).

- **RECONFIGURABLE PARTITION (RP)**: This is the dynamic (or run-time alterable) region. This RP has 2 parameters: `N` and `ud`. By combining these parameters' values, we can create a large set of variants (known as Reconfigurable Modules (RM)). `count_rp.vhd`: Wrapper file where we can modify the RP parameters in order to create different variants (RMs).
- We will modify (at run-time) the Reconfigurable Partition by utilizing two variants (the parameter `N` is set to $125 \times 10^6/2$).
  - ✓ `ud=1`: Up counter
  - ✓ `ud=0`: Down counter
- **Two Configurations**: Counter up (`count_rp` has `ud=1`), Counter down (`count_rp` has `ud=0`).



Procedure

- This procedure is adapted from the UG947: Vivado Design Suite Tutorial – Partial Reconfiguration. Changes were made to some `.tcl` files to allow us to use VHDL files and the Zynq-7000 PSoCs.
- Extract the **`my_dynled.zip`** file. Notice the file structure:
  - ✓ `design_complete.tcl`: Master script where the design sources, parameters, and structure are defined. It runs the entire design, from RTL to bitstreams. The supporting TCL scripts are located in `/Tcl`.
  - ✓ `design.tcl`: It is similar to `design_complete.tcl`, but it only runs synthesis. We will use this file in this tutorial.
  - ✓ `/Sources/hdl/top`: `top.vhd, static.vhd, my_genpulse.vhd`: These constitute the static region, i.e., the circuit that does not consider the Reconfigurable Partition (RP). Note that the RP is left as a black box.
  - ✓ `/Sources/hdl/count_up`: `count_rp.vhd, top_count.vhd, my_genpulse.vhd, my2bit_udcount.vhd`: These files constitute a Reconfigurable Module (where `ud` is set to '1' in `count_rp.vhd`), i.e., a variant of the RP.
  - ✓ `/Sources/hdl/count_down`: `count_rp.vhd, top_count.vhd, my_genpulse.vhd, my2bit_udcount.vhd`: These files constitute a Reconfigurable Module (where `ud` is set to '0' in `count_rp.vhd`), i.e., a variant of the RP.
  - ✓ `/Sources/xdc`: `top_io.xdc`: I/Os and clocking constraints (Period: 8 ns). This file is associated with the ZYBO Board.
    - If you have a ZYBO Z7-10 Board, you must use your own `top_io.xdc` file.
- Note: We are using the TCL-based flow (not the Vivado GUI-based flow). So, you have to execute the `design.tcl` script.

SYNTHESIS

- Open the Vivado TCL Shell (Program → Vivado 2019 TCL Shell). Navigate to the `/my_dynled` directory.
- Run the `design.tcl` script by entering: `source design.tcl -notrace`. This will Synthesize the design and create output files in the /Synth folder. The 'top' design will be created with a blank circuit for the Reconfigurable Partition.

ASSEMBLE THE DESIGN

- Open the Vivado IDE (`start_gui`). Go to the TCL console. You can now see the design structure in the Netlist pane.
- Load the design: `open_checkpoint Synth/Static/top_synth.dcp`
  You can see the design structure in the Netlist pane, but a black box exists for the `count_rp` partition. The instantiation name in the VHDL code is `ga`.
- Load the synthesized checkpoints for first Reconfigurable Module (RM) for each Reconfigurable Partition (RP). In our case, we will use the `count_up` as our first RM. We only have one RP (instantiation name: `ga`)
  `read_checkpoint -cell ga Synth/count_up/count_rp_synth.dcp`
  Note that the `count_rp` module has been filled in with logical resources.
- Define each RP as partially reconfigurable:
  `set_property HD.RECONFIGURABLE 1 [get_cells ga]`
- Save the assembled design state for this initial configuration (where RP is `count_rp` with `ud=1`, i.e., `count_up`):
  `write_checkpoint ./Checkpoint/top_link_up.dcp`

BUILD THE DESIGN FLOORPLAN

- Here, you create a floorplan to define the regions that will be partially reconfigured.
  - ✓ Select the `ga` instance in the Netlist pane. Right click and select Floorplanning → Draw Pblock. Draw a rectangular box that fits the resources occupied by the largest RM in that particular RP (instance name `ga`). The Statistics Tab of the Pblock Properties pane shows an estimate of the required resources for your module and the available ones in the box that you just drew. This is useful to optimize the resource count of your RPs.
  - ✓ Run PR Design Rule Checks by selecting Report → Report DRC. Be careful of the warnings, for example:
    HDPR-26 (left/right edge improper termination): The Pblock needs to be moved to a different area.
    HDPR-8 (no resets in the PR): Software controllable reset missing. This is not a problem in this visual example.
  - ✓ Save these Pblock definitions (RP size and location) and its associated properties on a `.xdc` file:
    `write_xdc ./Sources/xdc/fplan.xdc`
- If the Pblock definitions are already available (e.g., you had run this before), you can just read in that `.xdc` file. In this project, the file `pblocks.xdc` already includes the Pblock definitions: `read_xdc ./Sources/xdc/pblocks.xdc`

IMPLEMENT THE FIRST CONFIGURATION (RP: counter up)

- Load the top-level constraint file (to set device pinouts and top-level constraints):
  `read_xdc Sources/xdc/top_io.xdc`
- Optimize, place, and route the design. Notice the Partition Pins (interface points between static and dynamic regions)
  ```
  opt_design
  place_design
  route_design
  ```
- Save the full design checkpoint and create report files:
  ```
  write_checkpoint -force Implement/Config_count_up/top_route_design.dcp
  report_utilization -file Implement/Config_count_up/top_utilization.rpt
  report_timing_summary -file Implement/Config_count_up/top_timing_summary.rpt
  ```

At this point, you can use the static portion of this configuration for all subsequent configurations (variants of the circuit with different RMs for each RP). We need to isolate the static design by removing the Reconfigurable Modules:

- Clear out Reconfigurable Module logic:
  `update_design -cell ga -black_box`
- Lock down all placement and routing. This is an important step to guarantee consistency for different RMs for each RP.
  `lock_design -level routing`
- Write out the remaining static-only checkpoint (this checkpoint will be used for any future configurations).
  `write_checkpoint -force Checkpoint/static_route_design.dcp`

IMPLEMENT THE SECOND CONFIGURATION (RP: counter down)

- With the locked static design open, read in the post-synthesis checkpoint for the other Reconfigurable Module:
  `read_checkpoint -cell ga Synth/count_down/count_rp_synth.dcp`
- Optimize, place, and route the new RM.
  ```
  opt_design
  place_design
  route_design
  ```
- Save the full design checkpoint and report files:
  ```
  write_checkpoint -force Implement/Config_count_down/top_route_design.dcp
  report_utilization -file Implement/Config_count_down/top_utilization.rpt
  report_timing_summary -file Implement/Config_count_down/top_timing_summary.rpt
  ```

- At this point, you have implemented the static design and all Reconfigurable Module variants. This process would be repeated for designs that have more than two Reconfigurable Modules per RP, or more RPs. Close the current design:
  `close_project`

GENERATE BITSTREAMS

- Run the `pr_verify` command from the TCL console. This is to verify compatibility of all configurations.
  `pr_verify Implement/Config_count_up/top_route_design.dcp Implement/Config_count_down/top_route_design.dcp`

- Read the **first** configuration into memory:
  `open_checkpoint Implement/Config_count_up/top_route_design.dcp`
  - ✓ Generate full and partial bitstreams for the first configuration (ensure you are in the right window with the design open):
    `write_bitstream -file Bitstreams/Config_Up.bit`
    - Two bitstreams are created:
      `Config_Up.bit`: Power-up, full design bitstream
      `Config_Up_pblock_ga_partial.bit`: Partial bit file for the `count_rp` module (first RM – counter up)

  `close_project`

- Read the **second** configuration into memory:
  `open_checkpoint Implement/Config_count_down/top_route_design.dcp`
  - ✓ Generate full and partial bitstreams for the second configuration (locate in the right window with the design open):
    `write_bitstream -file Bitstreams/Config_Down.bit`
    - Two bitstreams are created:
      `Config_Down.bit`: Power-up, full design bitstream
      `Config_Down_pblock_ga_partial.bit`: Partial bit file for the `count_rp` module (second RM – counter down)

  `close_project`

- Generate a full bitstream with a **blackbox** for the RP, plus blanking bitstreams for the RMs, these can be used to erase an existing configuration to reduce power consumption:

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -cell ga -buffer_ports
place_design
route_design
write_checkpoint Checkpoint/Config_black_box.dcp
write_bitstream -file Bitstreams/config_black_box.bit
```

  Two bitstreams are created:
  `Config_black_box.bit`: Power-up, full design bitstream (with no logic in the RP)
  `Config_black_box_pblock_ga_partial.bit`: Partial bit file for the `count_rp` module (RM – black box)

  `close_project`

  * The `update_design` command inserts constant drivers (GND) for all outputs so that they don't float.

PARTIAL RECONFIGURATION OF THE FPGA

- From the main Vivado IDE (you might need to do `close_project` again), select Flow → Open Hardware Manager.
- Then `Open a New hardware Target`.
- Select Program Device and pick the XC7Z010 Device. Navigate to the `/Bistreams` folder to select `Config_Up.bit`. Program the device. You will see the 2 LSBs are counting up while the 2 MSBs switch from 01 to 10 (make sure `E=1`).

Partial Reconfiguration

- Select `Program Device`. Navigate to the Bitstreams Folder to select `Config_Down_pblock_ga_partial.bit`. Program the Device. The count on the 2 LSBs will change direction, while the 2 MSBs keep switching from `01` to `10` unaffected by Partial reconfiguration (note the much shorter reconfiguration time).
- Select `Program Device`. Navigate to the Bitstreams Folder to select `Config_black_box_pblock_ga_partial.bit`. Program the Device. The 2 LSBs will be blank, while the 2 MSBs keep switching from `01` to `10` unaffected by Partial reconfiguration (note the much shorter reconfiguration time).

You can repeat this experiment over and over with new partial bitstreams.

IMPORTANT NOTE:

- The steps in this tutorial can be skipped by executing the `design_complete.tcl` script:
  - ✓ Go to the Vivado TCL Shell, and type: `source design_complete.tcl -notrace` ↵
- It will compile the entire design, from RTL to bitstreams (it will also generate more intermediate checkpoints and reports). The script uses the constraint file `top.xdc` that merges `top_io.xdc` (I/O and clocking constraints) and `pblocks.xdc` (RP constraints). Usually, we do not know RP constraints and thus we need to define them manually (as described in this tutorial)
  - ✓ Note that the black box bitstream is not generated. You need to generate this manually.
  - ✓ The available `top.xdc` file is associated with the ZYBO Board. For the ZYBO Z7-10, you must use your own `top.xdc` file.

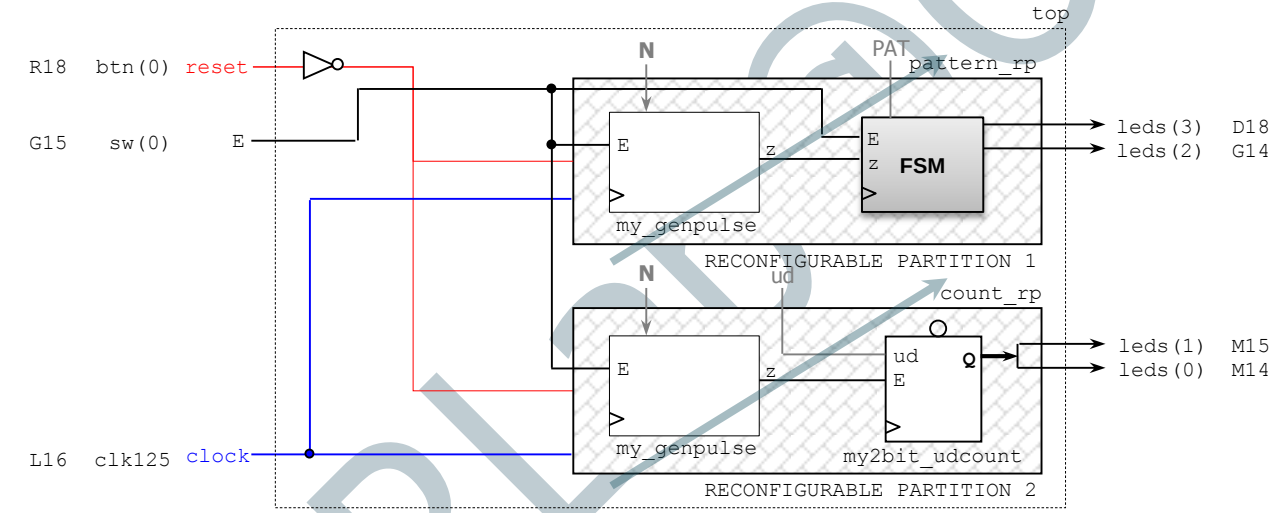
TEST PROJECT – 2 RPS

- **LED pattern control:** The circuit, written in VHDL, controls the pattern on the `leds(3..0)` signal.
  - ✓ `leds(3..2)`: The FSM switches between `10` and `01` if parameter `PAT="FIRST"`, and between `11` and `00` if parameter `PAT="SECOND"`. The rate of change is controlled by `my_genpulse` that issues a one-cycle pulse every N cycles.
  - ✓ `leds(1..0)`: It is controlled by a 2-bit updown counter whose rate of change is controlled by the `my_genpulse` circuit that issues a one-cycle pulse every N cycles (1 cycle = 8 ns in ZYBO Board).

- **RECONFIGURABLE PARTITIONS (RPs)**: There are two dynamic regions:
  - ✓ `pattern_rp`: This RP has two parameters: `N`, `PAT`. By combining their values, we can create a large set of variants (RMs). `pattern_rp.vhd`: Wrapper file where we can modify the RP parameters in order to create different variants (RMs).
  - ✓ `counter_rp`: This RP has two parameters: `N`, `ud`. By combining the parameters' values, we can create a large set of RMs. `count_rp.vhd`: wrapper file where we can modify the RP parameters in order to create different variants (RMs).
- We will modify (at run-time) the RPs by utilizing two variants for each one (the parameter `N` is set to $125 \times 10^6/2$).

Reconfigurable Modules for `pattern_rp` partition	Reconfigurable Modules for `counter_rp` partition
✓ `PAT="FIRST"`: 01-10 pattern	✓ `DIR=UP`: Counter up
✓ `PAT="SECOND"`: 11-00 pattern	✓ `DIR=DOWN`: Counter down

- **Four Configurations**:
  - ✓ `01-10` Pattern and Counter up
  - ✓ `01-10` Pattern and Counter down
  - ✓ `11-00` Pattern and Counter up
  - ✓ `11-00` Pattern and Counter down



SYNTHESIS

- Open the Vivado TCL Shell. Navigate to the `/my_led2RP` directory.
- Run the `design.tcl` script by entering: `source design.tcl -notrace`. This will Synthesize the design and create output files in the `/Synth` folder. The 'top' design will be created with a blank circuit for the Reconfigurable Partitions.

ASSEMBLE THE DESIGN

- Open the Vivado IDE (`start_gui`). Go to the TCL console. You can now see the design structure in the Netlist pane.
- Load the design: `open_checkpoint Synth/Static/top_synth.dcp`
  You can see the design structure in the Netlist pane, but blackboxes exist for the `pattern_rp` and `count_rp` partitions. The instantiation names in the VHDL code is `ga` and `gb`.
- Load the synthesized checkpoints for first Reconfigurable Module (RM) for each Reconfigurable Partition (RP). In our case, we will use `pattern_first` and `count_up` as our first RMs for each RP.
  ```
  read_checkpoint -cell ga Synth/pattern_first/pattern_rp_synth.dcp
  read_checkpoint -cell gb Synth/count_up/count_rp_synth.dcp
  ```
  Note that `pattern_rp` and `count_rp` partitions have been filled in with logical resources.
- Define each RP as partially reconfigurable:
  ```
  set_property HD.RECONFIGURABLE 1 [get_cells ga]
  set_property HD.RECONFIGURABLE 1 [get_cells gb]
  ```
- Save the assembled design state for this initial configuration (where RP1 is `pattern_first` and RP2 is `count_up`)
  ```
  write_checkpoint ./Checkpoint/top_link_first_up.dcp
  ```

BUILD THE DESIGN FLOORPLAN

Here, you create a floorplan to define the regions that will be partially reconfigured.

- Select the `ga` instance in the Netlist pane. Right click and select Floorplanning → Draw Pblock. Draw a rectangular box that fits the resources occupied by the largest RM in that RP (instance name `ga`). Repeat this procedure for the `gb` instance.
- Run PR Design Rule Checks by selecting Report → Report DRC.

- Save these Pblock definitions and its associated properties on a `.xdc` file:
```
write_xdc ./Sources/xdc/fplan.xdc
```

IMPLEMENT THE FIRST CONFIGURATION (RP1: pattern_first, RP2: count_up)

- Load the top-level constraint file (to set device pinouts and top-level constraints): `read_xdc Sources/xdc/top_io.xdc`
- Optimize, place, and route the design. Notice the Partition Pins (interface points between static and dynamic regions)
```
opt_design
place_design
route_design
```
- Save the full design checkpoint and create report files:
```
write_checkpoint -force Implement/Config_pattern_first_count_up/top_route_design.dcp
report_utilization -file Implement/Config_pattern_first_count_up/top_utilization.rpt
report_timing_summary -file Implement/Config_pattern_first_count_up/top_timing_summary.rpt
```

At this point, you can use the static portion of this configuration for all subsequent configurations (variants of the circuit with different RMs for each RP). We need to isolate the static design by removing the Reconfigurable Modules (RMs):

- Clear out Reconfigurable Module logic:
```
update_design -cell ga -black_box
update_design -cell gb -black_box
```
- Lock down all placement and routing. This is an important step to guarantee consistency for different RMs for each RP.
```
lock_design -level routing
```
- Write out the remaining static-only checkpoint (this checkpoint will be used for any future configurations).
```
write_checkpoint -force Checkpoint/static_route_design.dcp
```

IMPLEMENT THE SECOND CONFIGURATION (RP1: pattern_first, RP2: count_down)

- With the locked static design open, read in the post-synthesis checkpoints for the RMs that make up this Configuration:
```
read_checkpoint -cell ga Synth/pattern_first/pattern_rp_synth.dcp
read_checkpoint -cell gb Synth/count_down/count_rp_synth.dcp
```
- Optimize, place, and route the new RMs.
```
opt_design
place_design
route_design
```
- Save the full design checkpoint and report files:
```
write_checkpoint -force Implement/Config_pattern_first_count_down/top_route_design.dcp
report_utilization -file Implement/Config_pattern_first_count_down/top_utilization.rpt
report_timing_summary –file Implement/Config_pattern_first_count_down/top_timing_summary.rpt
```

IMPLEMENT THE THIRD CONFIGURATION (RP1: pattern_second, RP2: count_up)

- Clear out Reconfigurable Module logic:
```
update_design -cell ga -black_box
update_design -cell gb -black_box
```
- With the locked static design open, read in the post-synthesis checkpoints for the RMs that make up this Configuration:
```
read_checkpoint -cell ga Synth/pattern_second/pattern_rp_synth.dcp
read_checkpoint -cell gb Synth/count_up/count_rp_synth.dcp
```
- Optimize, place, and route the new RMs.
```
opt_design
place_design
route_design
```
- Save the full design checkpoint and report files:
```
write_checkpoint -force Implement/Config_pattern_second_count_up/top_route_design.dcp
report_utilization -file Implement/Config_pattern_second_count_up/top_utilization.rpt
report_timing_summary –file Implement/Config_pattern_second_count_up/top_timing_summary.rpt
```

IMPLEMENT THE FOURTH CONFIGURATION (RP1: pattern_second, RP2: count_down)

- Clear out Reconfigurable Module logic:
```
update_design -cell ga -black_box
update_design -cell gb -black_box
```
- With the locked static design open, read in the post-synthesis checkpoints for the RMs that make up this Configuration:
```
read_checkpoint -cell ga Synth/pattern_second/pattern_rp_synth.dcp
read_checkpoint -cell gb Synth/count_down/count_rp_synth.dcp
```
- Optimize, place, and route the new RMs.
```
opt_design
place_design
route_design
```
- Save the full design checkpoint and report files:
```
write_checkpoint -force Implement/Config_pattern_second_count_down/top_route_design.dcp
report_utilization -file Implement/Config_pattern_second_count_down/top_utilization.rpt
report_timing_summary –file Implement/Config_pattern_second_count_down/top_timing_summary.rpt
```
- At this point, you have implemented the static design and all Reconfigurable Module variants. Close the current design:
```
close_project
```

GENERATE BITSTREAMS

- Run the `pr_verify` command from the TCL console. This is to verify compatibility of all configurations.

```
pr_verify -initial Implement/Config_pattern_first_count_up/top_route_design.dcp -additional
{Implement/Config_pattern_first_count_down/top_route_design.dcp
Implement/Config_pattern_second_count_up/top_route_design.dcp
Implement/Config_pattern_second_count_down/top_route_design.dcp}
```

- Read the **first** configuration into memory:

```
open_checkpoint Implement/Config_pattern_first_count_up/top_route_design.dcp
```

  - ✓ Generate full and partial bitstreams for the first configuration

```
write_bitstream -file Bitstreams/Config_First_Up.bit
```

    - Three bitstreams are created:
      `Config_First_Up.bit`: Power-up, full design bitstream
      `Config_First_Up_pblock_ga_partial.bit`: Partial bit file for the `pattern_rp` RP (`pattern_first`)
      `Config_First_Up_pblock_gb_partial.bit`: Partial bit file for the `count_rp` RP (`count_up`)

```
close_project
```

- Read the **second** configuration into memory:

```
open_checkpoint Implement/Config_pattern_first_count_down/top_route_design.dcp
```

  - ✓ Generate full and partial bitstreams for the second configuration

```
write_bitstream -file Bitstreams/Config_First_Down.bit
```

    - Three bitstreams are created:
      `Config_First_Down.bit`: Power-up, full design bitstream
      `Config_First_Down_pblock_ga_partial.bit`: Partial bit file for the `pattern_rp` RP (`pattern_first`)
      `Config_First_Down_pblock_gb_partial.bit`: Partial bit file for the `count_rp` module (`count_down`)

```
close_project
```

- Read the **third** configuration into memory:

```
open_checkpoint Implement/Config_pattern_second_count_up/top_route_design.dcp
```

  - ✓ Generate full and partial bitstreams for the second configuration

```
write_bitstream -file Bitstreams/Config_Second_Up.bit
```

    - Three bitstreams are created:
      `Config_Second_Up.bit`: Power-up, full design bitstream
      `Config_Second_Up_pblock_ga_partial.bit`: Partial bit file for the `pattern_rp` RP (`pattern_second`)
      `Config_Second_Up_pblock_gb_partial.bit`: Partial bit file for the `count_rp` module (`count_up`)

```
close_project
```

- Read the **fourth** configuration into memory:

```
open_checkpoint Implement/Config_pattern_second_count_down/top_route_design.dcp
```

  - ✓ Generate full and partial bitstreams for the second configuration

```
write_bitstream -file Bitstreams/Config_Second_Down.bit
```

    - Three bitstreams are created:
      `Config_Second_Down.bit`: Power-up, full design bitstream
      `Config_Second_Down_pblock_ga_partial.bit`: Partial bit file for the `pattern_rp` RP (`pattern_second`)
      `Config_Second_Down_pblock_gb_partial.bit`: Partial bit file for the `count_rp` module (`count_down`)

```
close_project
```

- Generate a full bitstream with a **blackbox** for the RPs, plus blanking bitstreams for the RMs, these can be used to erase an existing configuration to reduce power consumption:

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -cell ga -buffer_ports
update_design -cell gb -buffer_ports
place_design
route_design
write_checkpoint Checkpoint/Config_black_box.dcp
write_bitstream -file Bitstreams/config_black_box.bit
```

  Three bitstreams are created:
  `Config_black_box.bit`: Power-up, full design bitstream (with no logic in the RPs)
  `Config_black_box_pblock_ga_partial.bit`: Partial bit file for the `pattern_rp` RP (black box)
  `Config_black_box_pblock_gb_partial.bit`: Partial bit file for the `count_rp` module (black box)

```
close_project
```

  * The `update_design` command inserts constant drivers (GND) for all outputs so that they don't float.

IMPORTANT NOTE:

- The steps in this tutorial can be skipped by executing the `design_complete.tcl` script:
  - ✓ Go to the Vivado TCL Shell, and type: `source design_complete.tcl -notrace` ↵