**Project description:**

In their final project the Fellows will be implementing a block which is dedicated for image processing purposes.

The block will support two operations:

⦁ **Threshold** – Convert an image from RGB to binary by turning all pixels below some threshold value to zero and all pixels above that threshold to one.

⦁ **Brighteness** – Add a constant value to each pixel of the image.

The block receives requests from two different external sources, uses a fixed arbitration (slv0 is the highest) if needed and gives service to the granted source according to the request mode, relevant operation value and the data sent from the granted source.

The data is sent from the sources through the data bus (configurable width of 32/64 bits) to the block where every byte represents either a header value or an actual pixel R/G/B value. The block supports processing BMP with pixel format of 24-bit pixel (24bpp).

An operation starts by receiving all the image relevant data from the block, performing the necessary calculations and then outputting the results outside.

If the block is during processing operation then no other requests can be received and thus no grant/rdy should be generated during processing time.

**Getting started: Instructions**

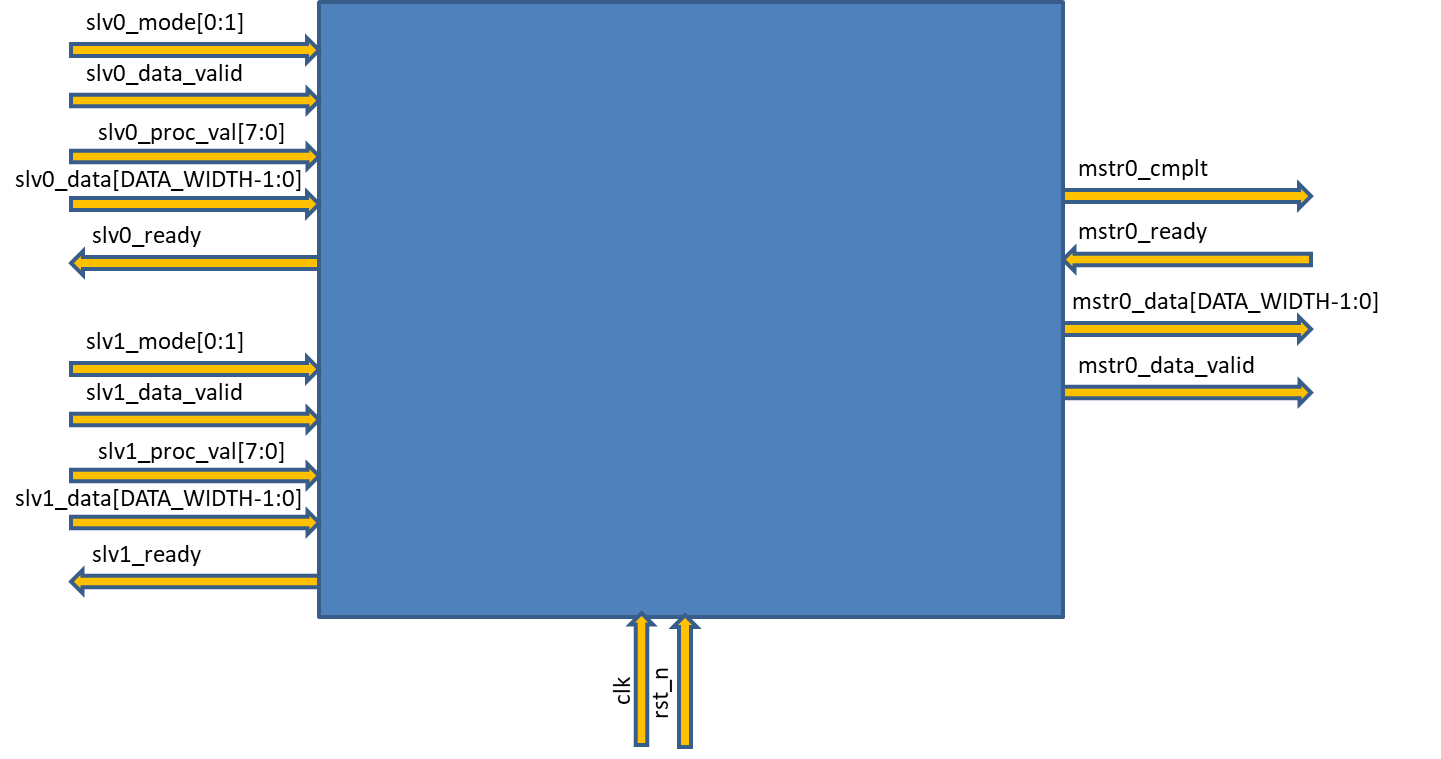
⦁ Load an array of images with multiple images before simulation. The values should be in HEX so each image should be converted from BMP to HEX (using Perl or Python or an online coverter). These images can be used by dedicated drivers to inject the data into the block.

Note that it’s using real images data is not mandatory. You can drive random values for simulation purposes.

⦁ Every time there’s a valid processed data on the bus, the data should be captured and written into an array of images so the images can be automatically or manually checked once the simulation is complete.

⦁ Use your knowledge of the images parameters (address in memory, height, width, etc.) to exercise you driver.

⦁ Reduce the problem by planning and implementing sub-blocks. For example, it’s recommended to implement a module that its only purpose is interacting with the sources requests. Think and plan the modules and how they connect with each other and with the outside world.



Signal definitions

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal Name** | **Width** | **Direction** | **Description** |
| clk |  | Input | Clock signal |
| rst\_n |  | Input | reset signal (active low) |
|  |  |  |  |
| slvX\_mode | 2 bits | Input | 0 = no req, 1 = threshold operation, 2 = brightness operation |
| slvX\_data\_valid | 1 bit | Input | Indicates the image start address in the memoryIndicates the data on the data bus is valid |
| slvX\_proc\_val | 8 bits | Input | Indicates to what address to processed image should be storedThe header/image data (header comes first and then row by row starting from the top left pixel) |
| slvX\_data | 32/64 bits | Input | Only relevant for brightness and threshold operations |
| slvX\_ready | bit | Output | Indicates req is granted or not |
|  |  |  |  |
| mstr0\_cmplt | 1 bit | Output | Indicates processing is completed |
| mstr0\_ready | 1 bit | InputOutput | Source X is ready to accept the processed dataError indication |
| mstr0\_data | 32/64 bits | Output | The processed data |
| mstr0\_data\_valid | 1 bit | OutputOutput | Indicates the data on the data bus is validMemory access address |
| mem\_rd |  | Output | Indicates memory read access |
| mem\_wr |  | Output | Indicates memory write access |
| mem\_wdata |  | Output | Write access data |
| mem\_rdata |  | Input | Read response data |
| mem\_rdata\_valid |  | Input | Read data is ready and valid |

Getting started: Instructions

* Load thean array of images memory with multiple images before simulation. The values should be in HEX so each image should be converted from BMP to HEX (using Perl or Python or an online coverter). These images can be used by dedicated drivers to inject the data into the block.

Note that it’s using real images data is not mandatory. You can drive random values for simulation purposes.

* Every time there’s a valid processed data on the bus, the data should be captured and written into an array of images so the images can be automatically or manually checked once the simulation is complete.
* Once the simulation is completed all the images stored in the memory should be converted back to BMP for visual or automated checking (using Perl or Python).
* Use your knowledge of the images parameters (address in memory, height, width, etc.) to exercise you driver.
* Reduce the problem by planning and implementing sub-blocks. For example, it’s recommended to implement a module that its only purpose is interacting with the memorysources requests. Think and plan the modules and how they connect with each other and with the outside world.
* Same goes for the external sources. They should all function as drivers and thus no Verilog implementation is needed.