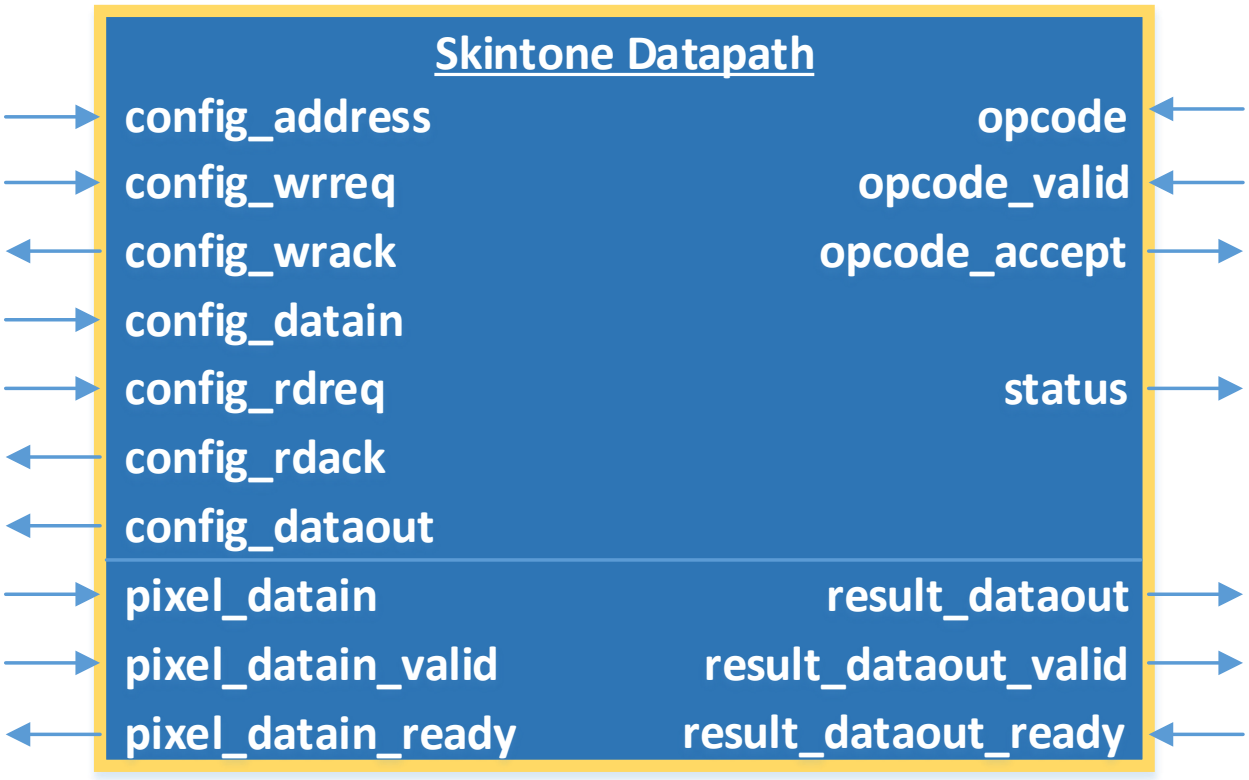

Figure 1) Block diagram of the skintone datapath module.

Overview

In this assignment you will verify the color-space conversion function of the skintone accelerator. An incremental version of the skintone accelerator that only instantiates the color-space converter is provided. A rudimentary SystemVerilog verification environment is also provided. This verification environment performs the essential tasks of configuring and streaming data in and extracting data out of the accelerator. The testbench is not comprehensive by any means and you will need to extend its functionality to find any potential bugs.

Step 1. Review the specification of the accelerator to become familiar with its theory of operation.

Step 2. Run the provided testbench to evaluate the design and to become familiar with the test environment. Review the SystemVerilog code to become intimately familiar with the testbench structure and behavior.

Step 3. Modify the testbench to generate more comprehensive test scenarios to uncover corner case bugs. If bugs are identified, implement an appropriate fix and rerun your verification suite to ensure you indeed fixed the bug(s).

Specification

The color-space converter transforms a stream of RGB pixels into a stream of YCbCr pixels. The pixel-wise transformation is performed by evaluating the following functions:

$$
\begin{aligned}
Y &= Offset_Y + (C_0 \times R)/256 + (C_1 \times G)/256 + (C_2 \times B)/256 \\
Cb &= Offset_{Cb} - (C_3 \times R)/256 - (C_4 \times G)/256 + (C_5 \times B)/256 \\
Cr &= Offset_{Cr} + (C_6 \times R)/256 - (C_7 \times G)/256 - (C_8 \times B)/256
\end{aligned}
$$

Equation 1 RGB to YCbCr Conversion functions

Here R, G, and B are the Red, Green, and Blue components of the pixel respectively. $Offset_Y$, $Offset_{Cb}$, $Offset_{Cr}$, and $C_{0\text{-}8}$ are constants that tune the conversion behavior. Although we could provide these constants at design time, a feature of our accelerator is to allow the user to configure these parameters at runtime through the software configuration API. To support this feature, the accelerator holds these constants in memory mapped "Control Registers" that are written via the configuration interface. Refer to Table 1 for the addresses of the individual Control Registers.

Table 1 Control Registers address map

Address [35:0]	Control Register [31:0]
0x000000000	C_0
0x000000010	C_1
0x000000020	C_2
0x000000030	C_3
0x000000040	C_4
0x000000050	C_5
0x000000060	C_6
0x000000070	C_7
0x000000080	C_8
0x000000090	$Offset_Y$
0x0000000A0	$Offset_{Cb}$
0x0000000B0	$Offset_{Cr}$

At the start of simulation, the design is configured through a series of configuration writes targeting the Control Registers. Once configuration is complete, the testbench streams data into the DUT, collects the output stream, and performs automatic checking.

Interfaces

The following tables and diagrams specify the DUT's external signals and their corresponding handshake protocols.

Clocking

Table 2 Clocking signal definitions

Signal Name	Direction	Description
clk	Input	Positive edge active clock. All signals are synchronous to this clock
rst	Input	Synchronous active high reset

Data Stream Interface

Table 3 Data Stream Interface signal definitions

Signal Name	Direction	Description
pixel_datain[127:0]	Input	Pixel bus carrying four ARGB pixels. The ordering is as follows $A_{i+3},R_{i+3},G_{i+3},B_{i+3},A_{i+2},R_{i+2},G_{i+2},B_{i+2}$, $A_{i+1},R_{i+1},G_{i+1},B_{i+1}$, A_i,R_i,G_i,B_i
pixel_datain_valid	Input	Qualifier signal signifying that the data on the pixel_datain bus is valid
pixel_datain_ready	Output	Qualifier signal signifying that the accelerator is capable of accepting data on the pixel_datain bus.
result_dataout[127:0]	Output	Result bus carrying four AYCbCr pixels. The ordering is as follows $A_{i+3},Y_{i+3},Cb_{i+3},Cr_{i+3},A_{i+2},Y_{i+2},Cb_{i+2},Cr_{i+2}$, $A_{i+1},Y_{i+1},Cb_{i+1},Cr_{i+1}$, A_i,Y_i,Cb_i,Cr_i
result_dataout_valid	Output	Qualifier signal signifying that the data on the result_dataout bus is valid
result_dataout_ready	Input	Qualifier signal signifying that the downstream component is capable of accepting data on the result_dataout bus.

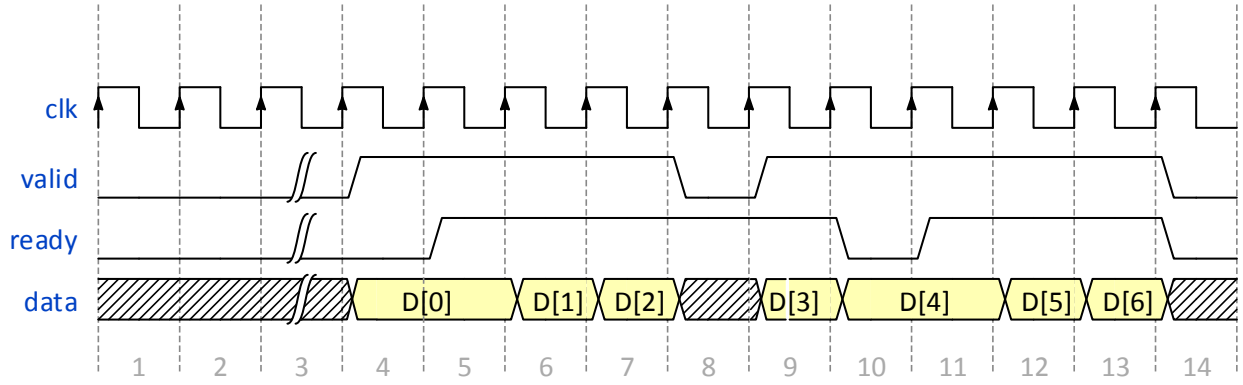

Figure 2 Data Stream Interface timing diagram

Configuration Interface

Table 4 Configuration Interface signal definitions

Signal Name	Direction	Description
config_address[35:0]	Input	36-bit configuration address
config_datain[127:0]	Input	128-bit configuration datain bus for writes into the accelerator configuration space
config_wrreq	Input	Qualifier signal signifying a write request to the accelerator configuration space
config_rdreq	Input	Qualifier signal signifying a read request to the accelerator configuration space

config_wrack	Output	Qualifier signal asserted by the accelerator to accept data during a write request (config_wrreq == 1)
config_rdack	Output	Qualifier signal asserted by the accelerator to signify valid data on the config_dataout bus in response to a read request (config_rdreq == 1)
config_dataout[127:0]	Output	128-bit configuration dataout bus for reads from the accelerator configuration space

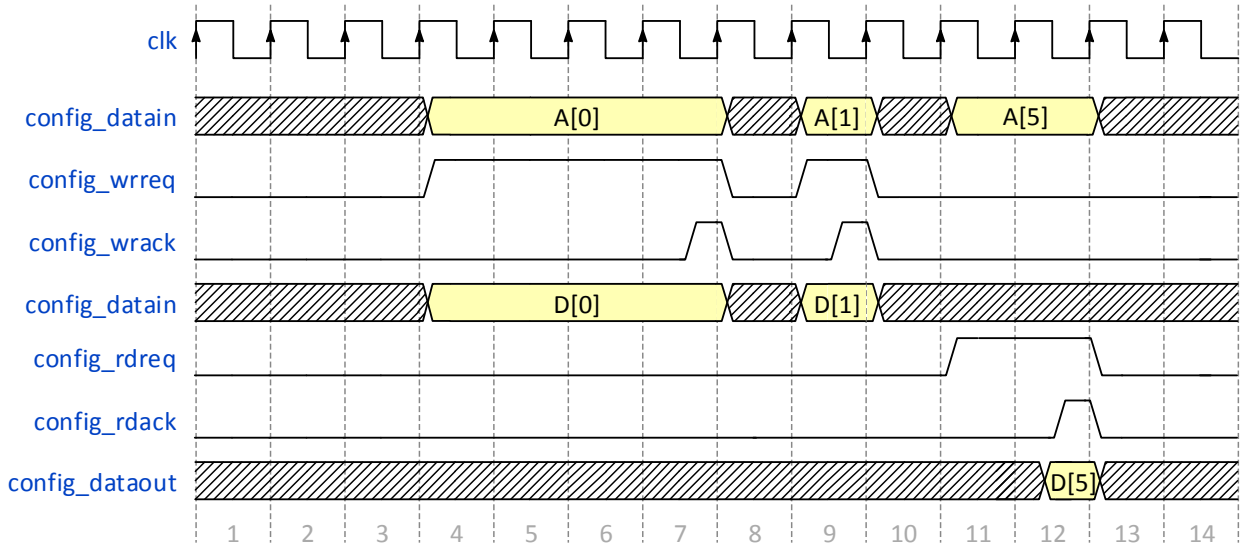

Figure 3 Configuration Interface timing diagram

Control Interface

Table 5 Control Interface signal definitions

Signal Name	Direction	Description
opcode[15:0]	Input	Opcode bus for passing accelerator specific runtime operation designation codes
opcode_valid	Input	Qualifier signal signifying that the data on the opcode bus is valid
opcode_accept	Output	Qualifier signal signifying that the accelerator has accepted the current data residing on the opcode bus

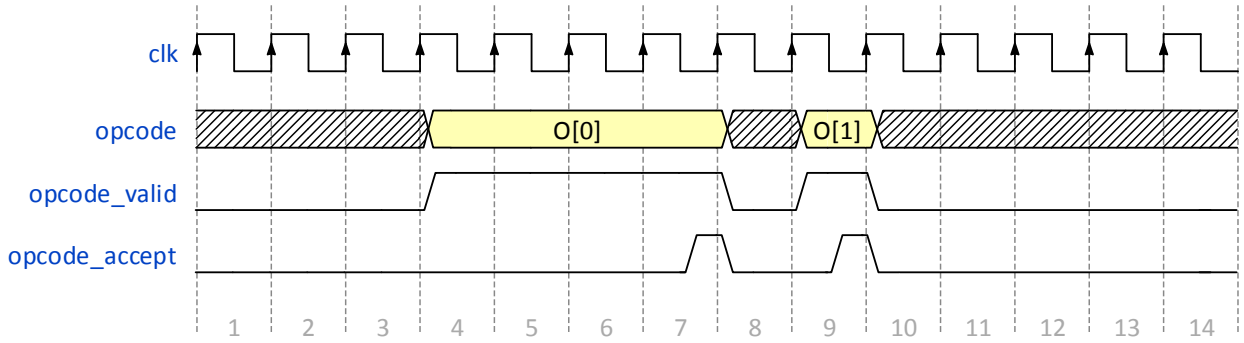

Figure 4 Control Interface timing diagram

Status Interface

Table 6 Status Interface signal definitions

Signal Name	Direction	Description
status[35:0]	Output	Accelerator specific bus for passing status information to external components (i.e. control unit)

Getting Started

To execute the simulation, do the following:

1. Start ModelSim simulator
2. At the ModelSim command prompt, change the "Current Directory" to the "simulation" folder which contains the script file **"startsim.do"**: cd <path to folder>
3. Compile and start the simulation by typing the following at the prompt: do startsim.do
4. Add the necessary signals into the waveform viewer
5. Run the simulation

Deliverables

Please submit the entire project directory as a zip archive. Your submitted directory should only have Verilog, SystemVerilog, and script files (*.do). Remove any simulation generated directories and files.