Hardware Verification – Main Coursework

Specification for AHB Peripherals – Version 1.0

Revision History

Revision	Issue Date	
1.0	27th October 2022	Initial version

Introduction

This is the specification for the two AHB peripherals, the verification of which will be the main objective of the coursework that you will submit for assessment.

See the "Hardware Verification - Introduction to Main Coursework" and "AHB Peripherals" presentations for further details.

The two peripherals are:

- GPIO (General Purpose Input/Output) Peripheral
- VGA Peripheral

The baseline RTL for these two peripherals will be provided; some changes will be needed to the GPIO in order to meet the specification and you will need to add additional RTL to implement a dual lock-step configuration for the VGA Peripheral.

AHB-Lite Interface (this is common to both peripherals)

The GPIO and VGA Peripherals are connected to the Cortex-M0 CPU using an AHB-Lite bus. The peripherals must adhere to the AHB-Lite bus protocol.

You can download the AHB-Lite Specification from this link:
https://developer.arm.com/docs/ihi0033/a

(You can ignore the other AHB-Lite protocol signals that are not listed here. They are either not needed by the GPIO or VGA Peripherals or can be tied off at the integration level.)

Signal Description

AHBLITE INTERFACE

Input signal name	Purpose
HSEL	Slave select signal
HCLK	AHB-Lite clock
HRESETn	AHB-Lite reset (active low)
HREADY	AHB-Lite Transfer Complete
HADDR [31:0]	AHB-Lite Address
HTRANS [1:0]	AHB-Lite Transfer Type
HWRITE	AHB-Lite Transfer Direction
HSIZE [2:0]	AHB-Lite Transfer Size
HWDATA [31:0]	AHB-Lite Write Data Bus

Output signal name	Purpose
HREADYOUT	Slave Transfer Complete
HRDATA [31:0]	AHB-Lite Read Data Bus

Output data to the peripherals will be transferred on the HWDATA bus and data from the peripherals will be transferred on the HRDATA bus. You will see that in some cases only the lower 8 bits or the lower 16 bits of the HWDTA and HRDATA buses are used.

NOTE: This interface is already implemented in the baseline RTL provided for each of the peripherals. However, you will need to ensure that the AHB-Lite bus protocol is conformed to in your unit-level verification. (Hint: Your testbench will need to drive the AHB-Lite bus according to the AHB-Lite specification; plus you could write some properties to check for correct AHB-Lite behaviour).

1. GPIO (General Purpose Input/Output) Specification

The GPIO block is a simple peripheral that enables data to be output to an external device and data to be read from an external device.

The GPIO shall implement the following registers:

- Data registers
    - Input data: the data read from external devices
    - Output data: the data sent to external devices
- Direction register
    - Controls whether it is a read or write operation

Register	Address	Size
GPIO Data	0x5300_0000	16 bits
GPIO Direction	0x5300_0004	16 bits

The direction of input/output is controlled by the direction register (gpio_dir [15:0]). In the baseline RTL provided, both the input and output buses are 16 bits wide. Although the direction register is also 16 bits wide, in this implementation only two values are used. gpio_dir[15:0] == 16’h0000 for an input operation, gpio_dir[15:0] == 16’h0001 for an output operation.

The basic operation is as follows:

Write the required direction to the direction register using an AHB write cycle with the AHB address set to 0x5300_0004

If the direction is output, then the value of GPIOUT[15:0] will be set on an AHB write cycle with the AHB address set to 0x5300_0000

An internal register (gpio_datain[15:0]) shall capture GPIOIN[15:0] on every clock edge when the direction is set to input and the value of GPIOOUT[15:0] when the direction is set to output and this internal register shall drive HRDATA[15:0].

Note: Parity generation and checking are not implemented in the RTL provided and need to be added

Parity Generation

When an entry is written into the GPIO output register (gpio_dataout), an additional parity bit shall be computed and stored alongside the 16-bit data entry. This parity bit should also be connected to an output pin, so you will need to increase the width of GPIOOUT to 17 bits.

The parity shall be pin-configurable as odd or even parity using the PARITYSEL input pin.

Parity Checking

The parity of the GPIO input (GPIOIN) should be checked when the data is registered on the input of the GPIO – i.e. when gpio_datain is updated from the GPIOIN input bus. Note that the width of GPIOIN will need to be increased to 17 bits to accommodate the extra parity bit. If there is a parity error, the PARITYERR signal should be asserted. For simplicity, the transfer should still proceed even if a parity error has occurred. (In a real system you could for instance connect the PARITYERR signal to the Cortex-M0 interrupt input to signal the error but this is beyond the scope of this assignment).

Fault injection

In order to test the parity generation and checking logic, you should implement a way of injecting a fault into either the data or the parity bit itself so that a PARITYERR is generated. It is up to you how you implement this (it could be via an extra input pin).

GPIO-specific Signal Description

GPIO Input/Output INTERFACE

Input signal name	Purpose
GPIOIN[15:0]	GPIO input port

Output signal name	Purpose
GPIOOUT[15:0]	GPIO output port

PARITY ERROR INTERFACE

Output signal name	Purpose
PARITYERR	Set to a '1' every time a parity error is detected

CONFIGURATION SIGNAL

Input signal name	Purpose

PARITYSEL	Parity selection input pin Odd parity if PARITYSEL=1 Even parity if PARITYSEL =0

2. VGA Peripheral Specification

The VGA peripheral is designed to display text and images on a monitor according to the VGA specification. Text and images to be displayed are written to the VGA peripheral using AHB-Lite write operations. See 'AHB Peripherals Presentation' for a more detailed description of the operation of the VGA peripheral. For the purposes of this coursework, you can consider the VGA peripheral as a design that conforms to the VGA specification.

NOTE: Your verification should focus on the operation of the text display function, not on the image display.

The VGA peripheral shall implement the following register map:

- Console text: 1 word (4 bytes) register to contain the character to be displayed
- Image buffer: The remainder of the 16Mbyte memory map (16M-4 bytes) is used to store pixels

Register	**Base Address**	**End Address**	**Size**
Console text	0x5000_0000	0x5000_0000	4 bytes
Image Buffer	0x5000_0004	0x50FF_FFFF	(16M-4) bytes

VGA-specific Signal Description

VGA output signal names	**Purpose**
RGB[7:0]	Pixel colour (3-bit red, 3-bit green, 2-bit blue)
HSYNC	Horizontal synchronisation signal; one pulse indicates the start of the next line
VSYNC	Vertical synchronisation signal; one pulse indicates the start of the next frame

The VGA specification that the VGA peripheral should conform to is for a 640x480 display at 60Hz, for which the industry standard timing is as follows:

Screen refresh rate: 60Hz

Vertical refresh: 31.46875kHz

Pixel frequency: 25.175MHz

The horizontal (line) timing is as follows:

Scanline part	Number of Pixels	Time (us)
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.813306852035
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

The vertical (frame) timing is as follows:

Frame part	Number of Lines	Time (ms)
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

As you will not be driving an actual monitor, for the purposes of simulation you could use a clock frequency of 25MHz to simplify things.

For the purposes of this coursework, you should at least verify that text characters are displayed in the correct part of the display, in other words in the "Visible area" as defined in the tables above and as show in the diagrams in the 'AHB Peripherals' Presentation slides.

Text display operation

The VGA peripheral contains logic that drives the RGB[7:0], HSYNC and VSYNC signals at the correct times in order to display the characters at the correct place on the display. Each character is converted into a matrix of pixels and each row of that matrix is driven line by line to make up the complete character. If you look at the file font_rom.v, you will see how the matrix of pixels is derived for each ASCII character. For example, the ASCII code for the digit 1 is (in hex) 0x31 and the pixel matrix for this is as follows (there are blank lines above and below the character):

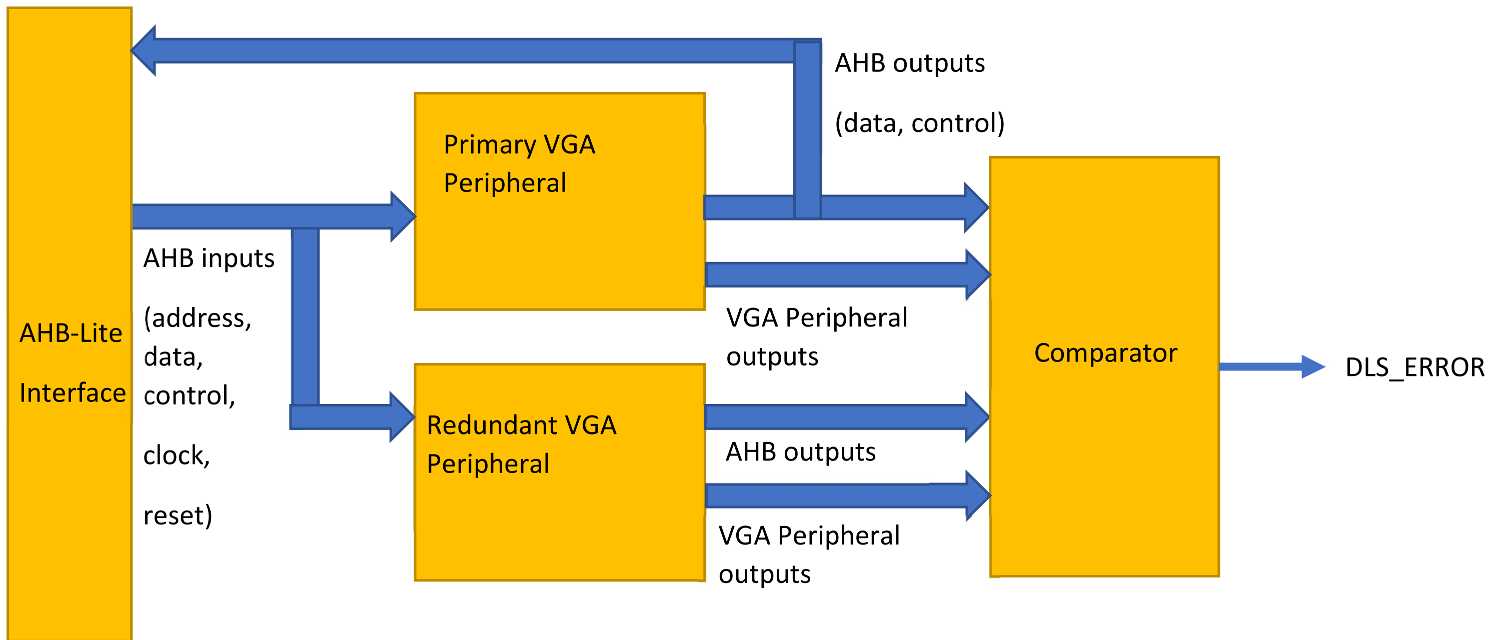
```
//code x31
          11'h310: data = 8'b00000000; //
          11'h311: data = 8'b00000000; //
          11'h312: data = 8'b00011000; //
          11'h313: data = 8'b00111000; //
          11'h314: data = 8'b01111000; //    **
          11'h315: data = 8'b00011000; //   ***
          11'h316: data = 8'b00011000; //  ****
          11'h317: data = 8'b00011000; //    **
          11'h318: data = 8'b00011000; //    **
          11'h319: data = 8'b00011000; //    **
          11'h31a: data = 8'b00011000; //    **
          11'h31b: data = 8'b01111110; //    **
          11'h31c: data = 8'b00000000; //    **
          11'h31d: data = 8'b00000000; //  ******
          11'h31e: data = 8'b00000000; //
          11'h31f: data = 8'b00000000; //
```

And so if you write the character '1' to the VGA peripheral via the AHB bus, this matrix of pixels should appear in the visible display area of the VGA monitor. See if you can develop a checker that verifies this. You can do this for just one or two characters, I don't expect you to verify the complete ASCII character set can be displayed! By using the 'Number of Pixels' and 'Number of Lines' information above and using the HSYNC and VSYNC signals as reference points, by counting clocks you should be able to work out when the characters will be displayed and then for a few lines record which pixels are being driven (RGB[7:0] is non-zero). You don't need to verify the colour information.

(For simplicity this VGA Peripheral is hard-wired to display all characters in the text display in green.)

Dual Lock-Step Configuration of VGA Peripheral

The RTL provided for the VGA Peripheral can be used unmodified for this coursework (unless you find a bug in it!) However, you are required to implement a Dual Lock-Step configuration of the VGA Peripheral so as to be able to detect any hardware faults that may occur. To do this, you need to instantiate a second identical copy of the VGA Peripheral and connect it to the same input signals as the first copy. For this specification we will refer to the original VGA Peripheral as the 'Primary' and the second redundant copy as the 'Redundant' VGA Peripheral. Only the Primary block should have its AHB outputs connected to the rest of the SOC. To be able to detect a fault, the outputs from the Primary and Redundant blocks shall be connected to a comparator that compares the outputs from the two blocks on every clock cycle (i.e. they run in lock-step). If any difference in the outputs is detected, the comparator shall assert the signal **DLS_ERROR**



DLS Comparator Output Signal Description

Output signal name	Purpose
DLS_ERROR	Dual Lock-Step Error

Note that to verify the operation of the comparator, you will need to implement some way of injecting a fault in the Primary or Redundant blocks (or in the comparator itself).