Hardware Verification – Introduction to Main Coursework

ELEC70056

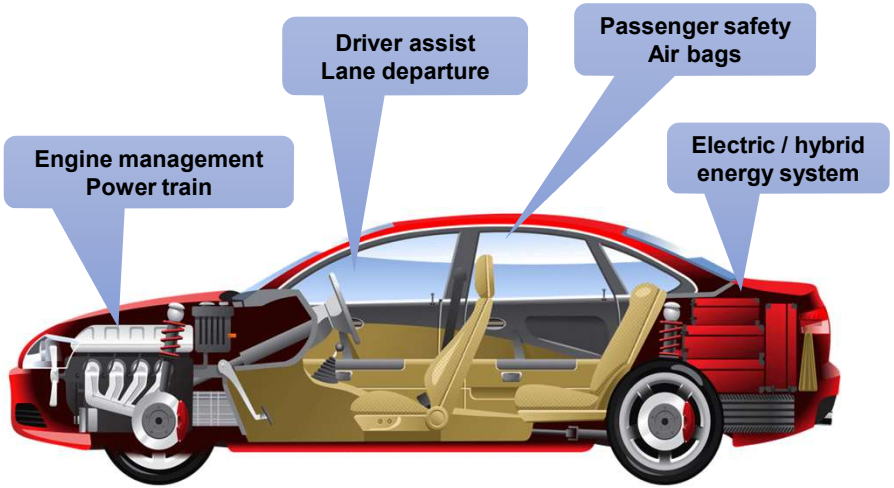
Autumn term 2022

Version 2 – October 27th 2022

Pete Harrod

Functional safety – detecting faults

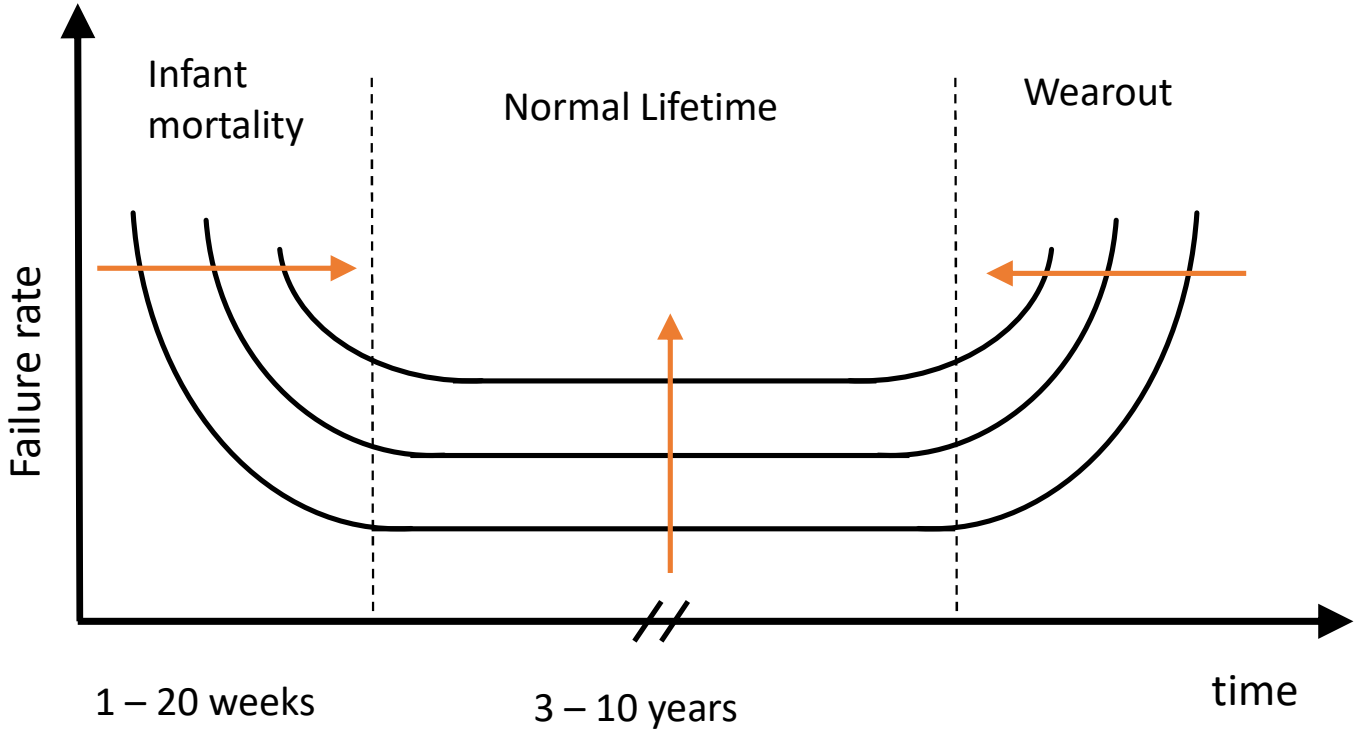
- Functional safety is required for many automotive systems
- Detecting and managing faults is required
- Example: ABS brakes
  - If a fault is detected, then a typical strategy is to do a h/w reboot
  - If the fault is transient, then system will resume normally
  - If the fault is permanent, then the function can be disabled
- The coursework will involve
  - adding mechanisms to detect faults to the RTL design that is provided
  - ensuring the thorough verification of these features, in addition to the verification of the functionality of the blocks



Types of Faults

- Random hardware failures
  - Permanent faults
    - Manufacturing faults (stuck-at, bridging)
    - Design bug
    - Critical path due to slow process
    - Latch-up due to alpha-particle strike
  - Transient faults
    - Bit flip in memory due to alpha-particle
- Common-mode failures
  - Clock
  - Reset
  - EMC

The Bathtub Curve

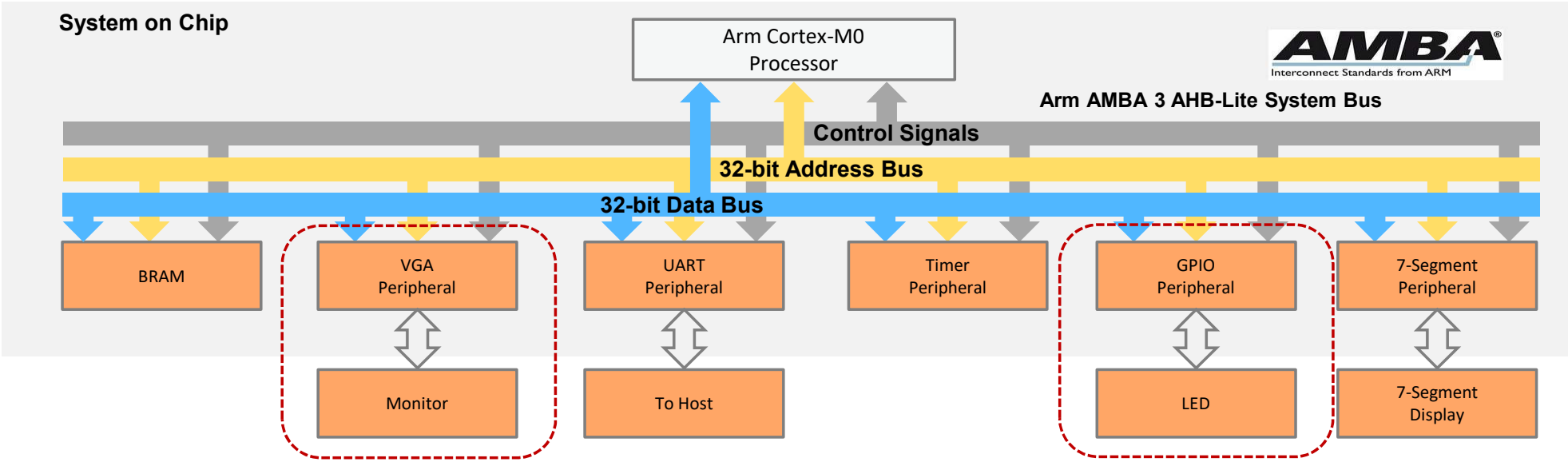


- Infant mortality: *Increasing manufacturing defects*
- Normal lifetime: *Increasing transient errors*
- Wearout: *Acceleration of aging phenomena*

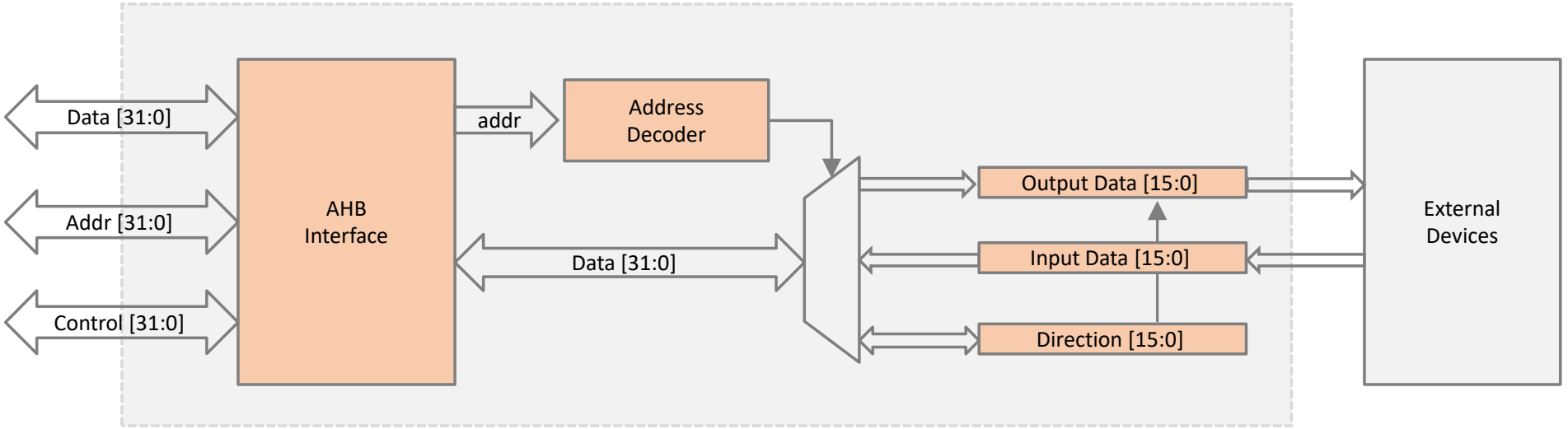
Coursework objectives

- To add error-detection features to and fully verify AHB GPIO and VGA peripherals, integrated into an example Cortex-M0 SOC
- The basic RTL of the GPIO and VGA peripherals is provided
- You will need to:
  - Modify the GPIO RTL to add parity generation/checking
  - Instantiate a redundant VGA peripheral block plus add RTL for a comparator
  - Create unit-level constrained random testbenches written in SystemVerilog for the GPIO and VGA peripherals
  - Write appropriate SystemVerilog assertions (SVA)
    - To cover designer intent
    - To describe behaviour of the interface(s)
  - Attempt to prove some properties using Formal verification
  - Develop checkers for the GPIO and VGA peripheral behaviour
  - Write functional coverage and demonstrate coverage has been achieved
  - Demonstrate integration and verification at the Cortex-M0 SOC level

Example SOC


System on Chip
Arm Cortex-M0 Processor
AMBA
Interconnect Standards from ARM
Arm AMBA 3 AHB-Lite System Bus
Control Signals
32-bit Address Bus
32-bit Data Bus
BRAM
VGA Peripheral
UART Peripheral
Timer Peripheral
GPIO Peripheral
7-Segment Peripheral
Monitor
To Host
LED
7-Segment Display

GPIO – the basics



GPIO Design

- GPIO – General Purpose Input Output
- Direction can be controlled
- Modifications required:
  - Add pin to configure odd or even parity
  - Calculate parity on write data word and output parity bit along with the data; check parity on read data and signal if parity error

GPIO Design modifications – more details

- Add parity generation on GPIO output and parity checking on GPIO input
- 1 parity bit per 16-bit data word
- Pin configurable odd/even parity

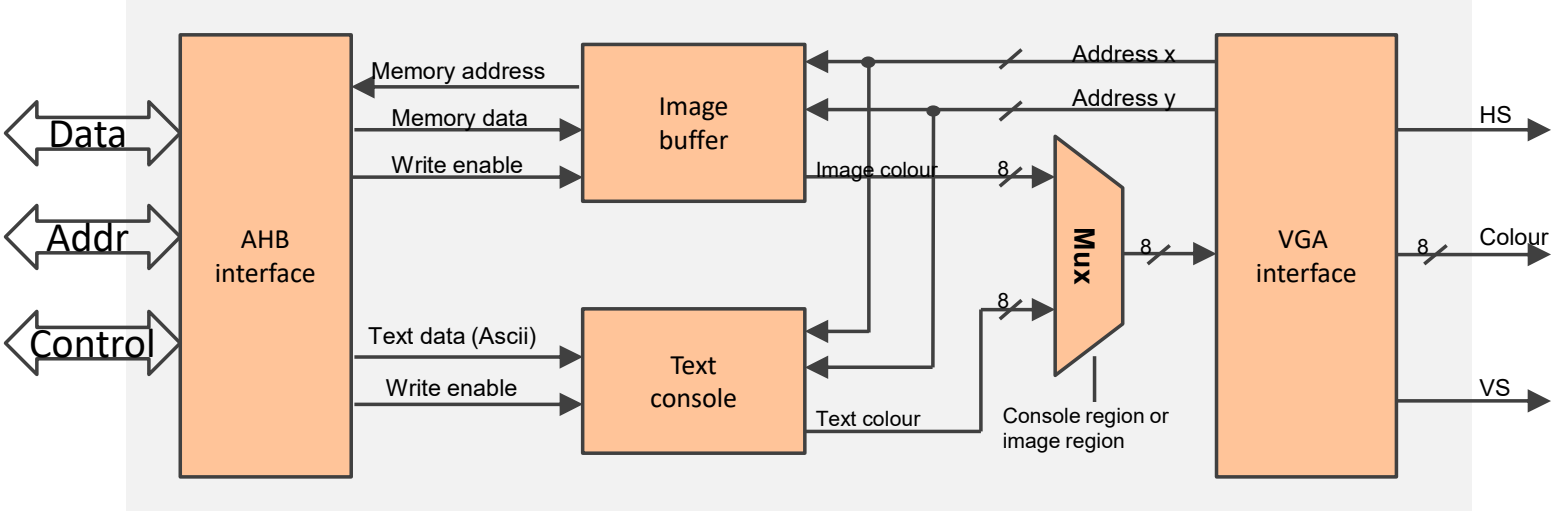
GPIO – verification at the unit-level

AHB-Lite interface

AHB VGA Peripheral Hardware Architecture

The VGA peripheral can display texts and images on a monitor through a VGA cable.

The VGA peripheral consists of 5 components: , an AHB interface, a VGA interface, an image buffer for displaying images, a text console module for displaying texts, and a multiplexer.



arm

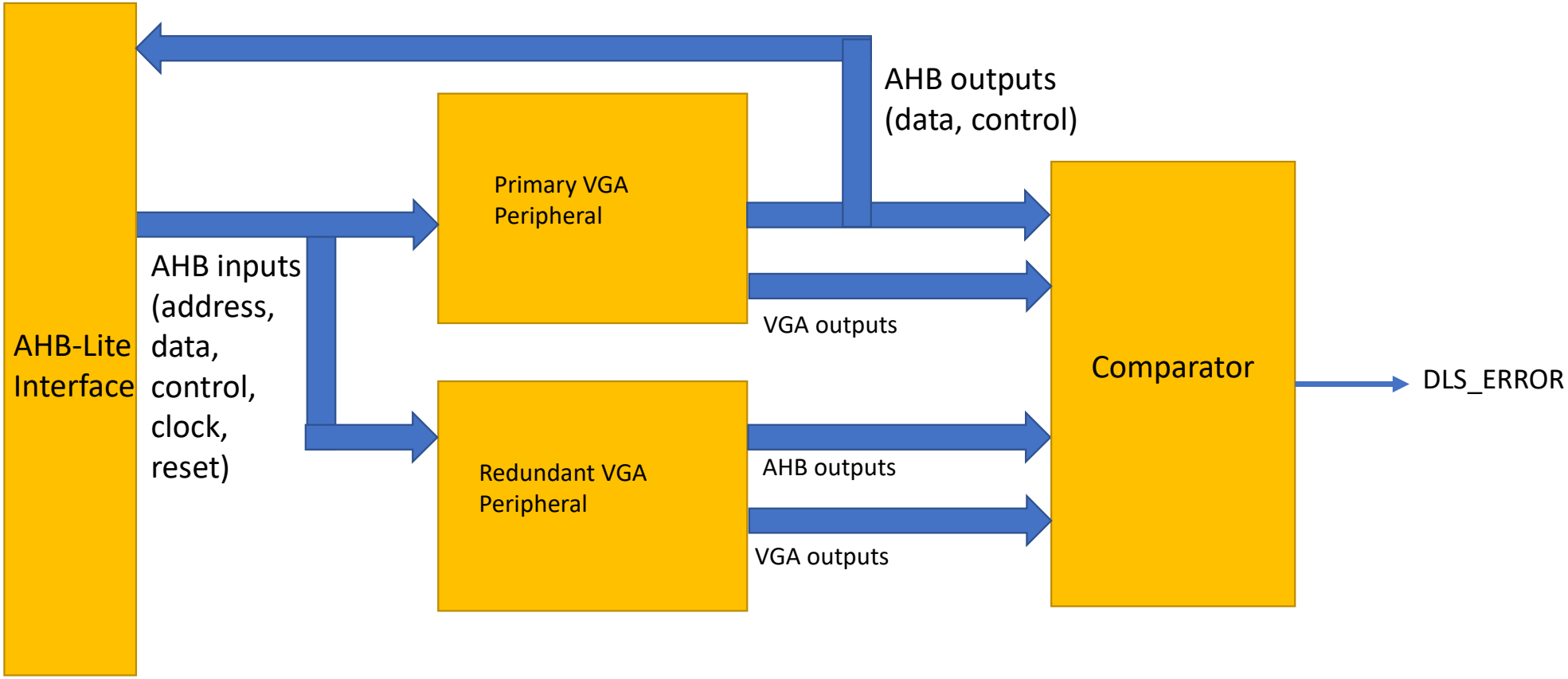
VGA Peripheral – verification at the unit-level

AHB-Lite interface

VGA Peripheral – Dual Lock-Step Configuration

- VGA Peripheral RTL is provided
- Create a Dual Lock-Step (DLS) configuration for the VGA Peripheral, so as to be able to detect faults
- Modifications required:
  - Instantiate a second redundant copy of the VGA Peripheral
  - Write the RTL for a comparator block, that compares the outputs from the Primary block with the Redundant block and signals an error if there is a mismatch
  - Provide a way of injecting a fault into the Primary block, Redundant block or comparator, to test the operation of the DLS

Dual Lock-Step Configuration



Verification Requirements

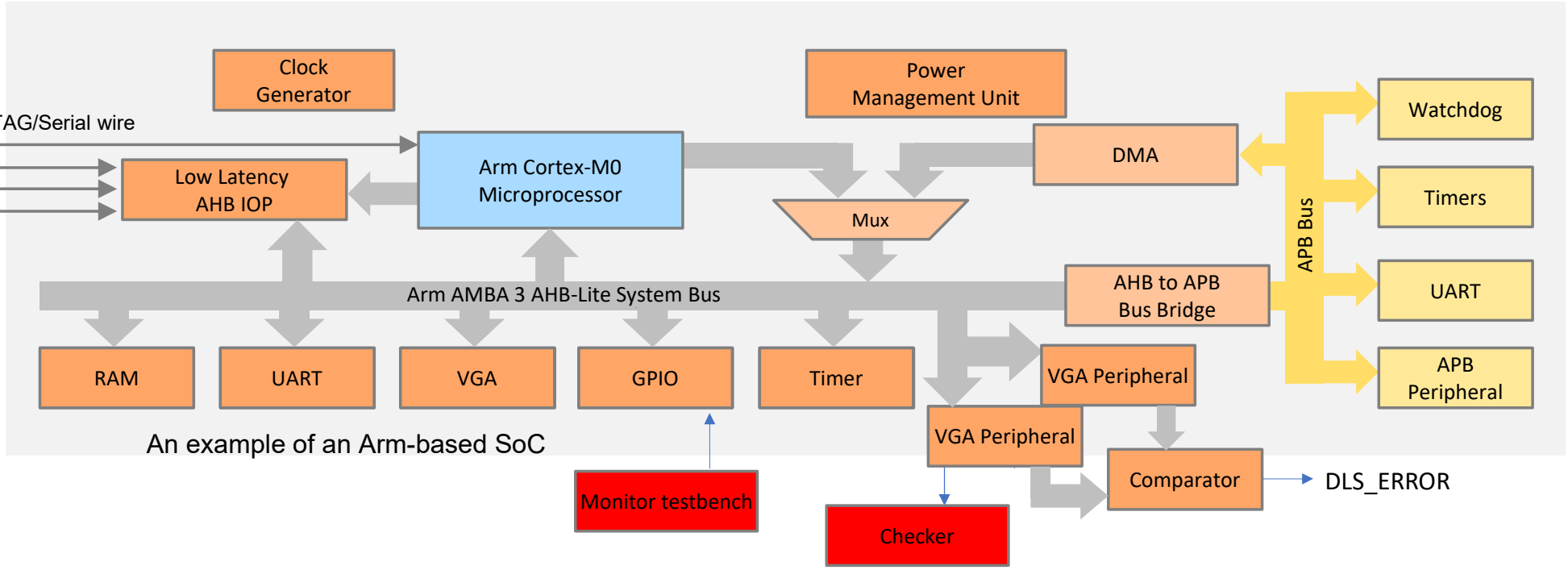
- Write a basic Verification Plan
- Create unit-level constrained random testbench(es) for the two blocks (GPIO and VGA Peripheral)
- Write SVAs
- Use SV constructs included in lectures in your testbench code
  - Interfaces
  - Clocking blocks
  - Class – e.g. for randomised variables
- Include a way of injecting faults into data and parity bit, to cause parity error
- Include a way of injecting faults in dual lock-step VGA Peripheral configuration, to cause comparator errors

More Verification Requirements

- Attempt to prove some properties with JasperGold formal tool
- Provide some way of checking the behaviour of the GPIO and VGA Peripherals (limited checking for the VGA)
- Write functional coverage points and demonstrate coverage
- Measure code coverage
- Write directed tests in Assembler for integration testing

Final objective

- Verify GPIO and VGA Peripheral operating in example Cortex-M0 SOC
  - Write assembler code to provide directed testing of both peripherals



Cortex-M0 and SOC code

- RTL for GPIO and VGA Peripheral blocks will be provided
- Example Assembler code for driving existing blocks will also be given
- You will need to download the Cortex-M0 DesignStart model from Arm (instructions will be provided)
  - DesignStart model is a netlist model
  - You will need to agree to the license terms
- Suggest trying to simulate Cortex-M0 SOC with existing blocks before making modifications to the GPIO and adding the redundant VGA Peripheral block

Deliverables

- Brief ReadMe, with any information that you think will be useful to me when assessing your work, e.g.
  - Location of files (and what they are for if not obvious)
  - Any important points about your verification strategy and any difficulties that you encountered
  - Brief summary of your verification results
- Verification Plan (a list of what requires testing and how it will be tested)
- Modified RTL code for the GPIO and the dual lock-step configuration of the VGA Peripheral
- SystemVerilog files of your testbench(es)
- Your SystemVerilog assertions and evidence of trying to prove some of these with formal verification (a screenshot from Jaspergold or a log file)
- Checkers for both the GPIO and VGA (see the 'AHB Peripherals Specification' for what should be checked for the VGA)
- Evidence of running the unit-level testbench simulation (log file or screenshot)
- Assembler (or C) code for your top-level integration testing
- If possible, a log file or screenshot showing the result of running your top-level test
- Functional coverage points (you can include these with your testbench code)
- Code coverage and functional coverage reports (in text or HTML format)

Some hints for verification of the GPIO

- You could choose to model the GPIO in SystemVerilog as a way of checking the functionality of the Verilog RTL
- You might choose to do end-to-end checking for the GPIO
  - Write a data stream, then read the data back through the GPIO and check you receive what was sent
- The “monitor” BFM can be used for unit-level and top-level verification
- Get your modified design working at unit-level before integrating it into the SOC
- Use properties written as SVAs to help with bring-up (e.g. for parity logic)
  - Can try and prove these with Formal tool rather than simulation