Image Edge Detection with RC4 cipher using AHB Lite-Based SoC Verification Plan

Team members: Dimcho Karakashev, Ruchir Aggarwal, Gautam Rangarajan, Ribhav Agarwal

Architecture

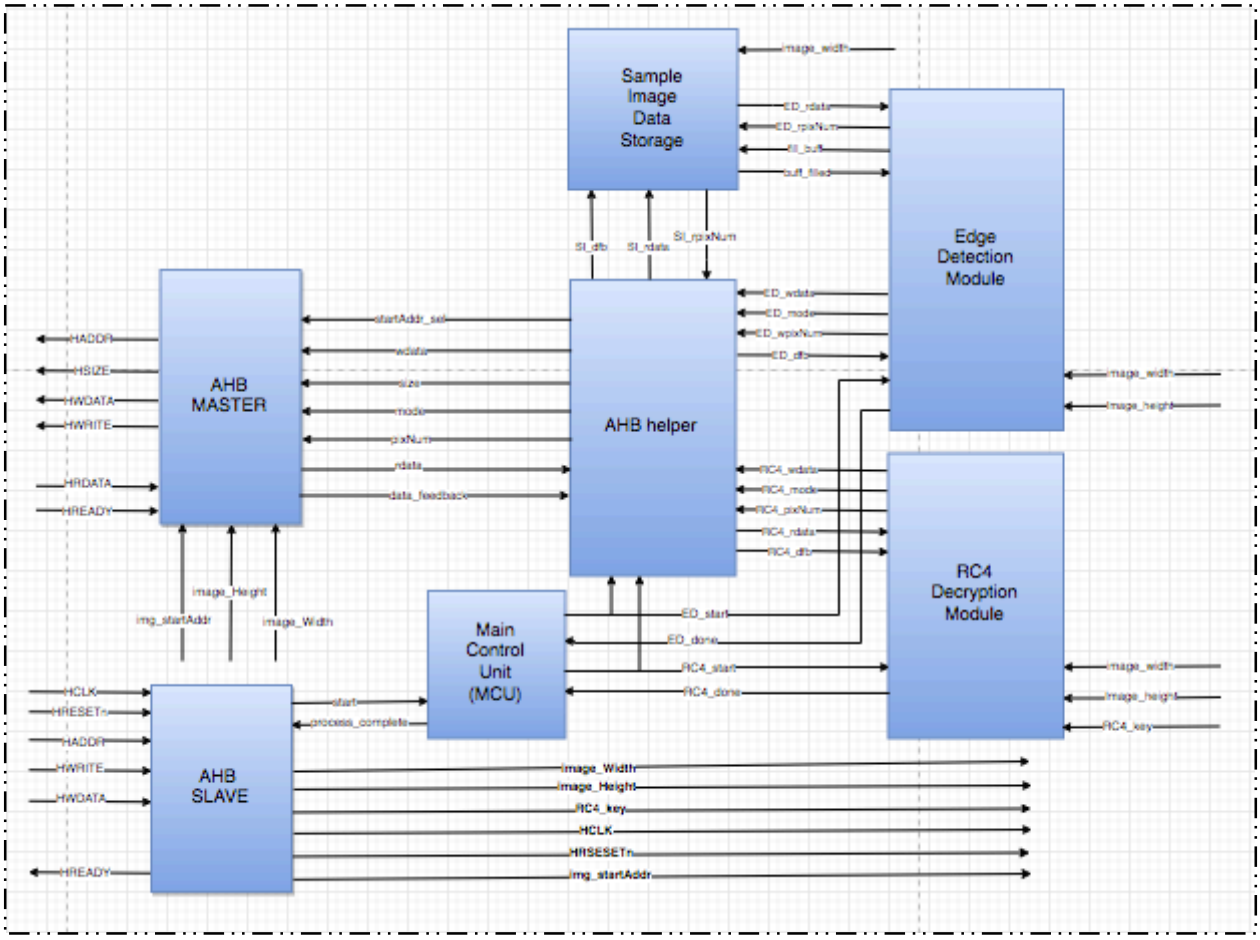

Figure1: Design Architecture

- **AHB-Lite Slave Interface Module:** Top-level block that handles data sending and reception via the AHB-Lite Bus between CPU and our design. Has a set of memory mapped registers that hold information about image like image size and image starting address, and decryption key supplied by the user for RC4 decryption. All of this information is supplied via writing 32-bit value to a designated address for that information.
- **AHB-Lite Master Interface Module:** Top-level block that handles data sending and reception via the AHB-Lite Bus between our design and SRAM.

- AHB-Helper Module: Top-level block that handles setting of control signals and relaying of data values between AHB-master and RC4 Decryption module or AHB-master and Edge-Detection module.
- Main Control Unit Module: Top-level block that handles the order and transition between edge-detection and RC4 decryption. It also responds when the entire processing on an image is done.
- RC4 Decryption Module: Top-Level block that handles all of the steps for decrypting encrypted data received by the design using the 32-bit key supplied by the user.
- Sample Image Data Storage Module: Top-level block that handles getting the correct data from the image so that edge-detection can process the right pixels.
- Edge Detection module: Top-level block that handles all of the steps for Sobel edge-detecting image data received from Sample Image Data Storage Module and sends the processed data to AHB-master for writing.

Fixed Success Criteria

1. Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria. (2 pts)
2. Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings (4 pts)
3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero (2 pts)
4. A complete IC layout is produced that passes all geometry and connectivity checks (2 pts)
5. The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0. (2 pts)
    a. Area: 5mm x 5mm
    b. Pin Count: 3 (power, ground, clock)
    c. Clock Period: 28.5 MHz

Design Specific Criteria

1. Demonstrate by simulation of a Verilog test bench that decryption and edge-detection do not begin until all user inputs have been received (1pt)
2. Demonstrate by simulation of a Verilog test bench that our module works correctly for variable image sizes (1pt)
3. Demonstrate by simulation of a Verilog test bench that image obtained after RC4 decryption matches the expected output (2pt)
4. Demonstrate by simulation of a Verilog test bench that Sobel Edge detection gives a visible and convincing image (2pts)
5. Demonstrate by simulation of a Verilog test bench that data transmitted serially on the AHB-LITE bus is reading, writing and waiting correctly (2pts)

Verification Plan Summary

What to Verify	Design Module(s) involved	Verification Procedure Summary	DSSC(s) Proved	Use in Final Demo	Comments
Chip holds processes until user inputs have been received	Top Level	Test bench to monitor AHB-slave MM-registers and MCU outputs	DSSC 1	Yes	Use temporary registers in test bench to grab user inputs and check MCU outputs
Correctness of Edge-detection on variable image sizes	Top Level	Compare results to an alternate implementation	DSSC 2	Yes	Test images are of different sizes. Should be able to reuse most of the test benches
Correctness of RC4 Decryption	Top Level	Compare decrypted results to original image	DSSC 3	Yes	Image is sent to python implementation of RC4 encryption whose output is used as input to our RC4 decryption. The output is compared to original image.
Correctness of Sobel Edge-Detection	Top Level	Compare edge detection results to an alternate implementation	DSSC 4	Yes	Sobel Edge-detected image is compared with the output of python implementation of Sobel edge-detection
AMBA AHB-Lite Protocol interfacing	Top Level	Test Bench provides samples of each type of bus transaction	DSSC 5	Yes	Should be able to reuse test bench code from AMBA bus interface controller
AMBA AHB-Lite Protocol interfacing	AHB-Lite Slave, AHB-Lite Master	Test Bench provides samples of each type of bus transaction	DSSC 1	Only if can't show using top level	Should be able to reuse test bench code from AMBA bus interface controller
Correct functioning of AHB-Helper	AHB-Helper	Test bench to simulate rc4 r/w and edge-detection r/w	DSSC 1	Only if can't show using top level	Verify that the right control signals are set by AHB-Master for different instructions

Chip holds processes until user inputs have been received	AHB-Lite Slave, MCU	Test bench to monitor AHB-slave MM-registers and MCU outputs	DSSC 2	Only if can't show using top level	Use temporary registers in test bench to grab user inputs and check MCU outputs
Correctness of RC4 Decryption	RC4 decryption module	Compare decrypted results to original image	DSSC 3	Only if can't show using top level	Verify that RC4 decryption works correctly for one byte of data
Correctness of Sobel Edge-Detection	Edge Detection module	Compare edge-detection results with an alternate implementation	DSSC 4	Only if can't show using top level	Verify that Sobel edge-detection algorithm works correctly on one set of buffered data
Correctness of sample buffer data	Sample Image Data storage	Compare sample buffer data with an alternate implementation	DSSC 4	Only if can't show using top level	Verify that the data required to calculate results for two pixels is grabbed correctly.
Correctness of decryption/edge-detection on variable image sizes	RC4 Decryption module, Edge-detection module	Compare results to an alternate implementation	DSSC 5	Only if can't show using top level	Compare results of each block with expected outputs for variable image sizes

Detailed verification Test Breakouts

Demo Tests

Chip holds processes until user inputs have been received

- Shown in Demo: Yes
- DSSC(s) Proved: 1
- Highest Level of Design module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Have test vector of samples of each type of bus transaction and correct chip response information
    - Chip response should be verified by checking against standards
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
    - Write image specifications + decryption key onto AHB slave
    - Check AHB response to MCU.
    - Grab MCU response to RC4/Edge-detection to check if processing started.

Correctness of Edge-detection on variable image sizes

- Shown in Demo: Yes
- DSSC(s) Proved: 2
- Highest Level of Design module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Extract encrypted image data from file (produced by python code), and store in temporary registers (these simulate the SRAM).
    - Have temporary registers to capture decrypted results. This data is sent as the inputs to the edge detection module.
    - Have temporary registers to capture edge detected results, that will later be saved in an external file. This file is compared with the output of python implementation of edge detection to ensure the correctness of our edge detection module.
    - Emulate the initiation of image processing by inputting image specifications + RC4 decryption key.
- No external or premade references are needed
- Pre-processing is needed
    - A python code that generates encrypted image will be run, and the output of this will be stored in a file. This will be used by the test bench as the input to the chip. This is repeated for images of 3 different sizes.
- Main verification Test Steps:
    - All steps from the testing of DSSC 5 are repeated for images of 3 different sizes.

Correctness of RC4 Decryption

- Shown in Demo: Yes
- DSSC(s) Proved: 3
- Highest Level of Design module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Extract encrypted image data from file (produced by python code), and store in temporary registers (these simulate the SRAM).
    - Have temporary registers to capture decryption results, that will later be saved in an external file. This file is compared with the original image to ensure the correctness of RC4 decryption. The data on these temp registers is also used as the inputs to the edge detection module.
    - Emulate the initiation of image processing by inputting image specifications + RC4 decryption key.
- No external or premade references are needed
- Pre-processing is needed
    - A python code that generates encrypted image will be run, and the output of this will be stored in a file. This will be used by the test bench as the input to the chip.
- Post-processing is required
    - A python code that generates the edge detected image will be run, and the output of this will be stored in a file. This will be used to determine the correctness of the output of our chip. This is repeated for images of 3 different sizes.
- Main verification Test Steps:
    - Image processing steps are initiated by inputting image specifications + RC4 decryption key data into chip.
    - Send encrypted image data as input during RC4 decryption stage.
    - Store decrypted data on temp registers.
    - After all of the encrypted data has been decrypted, write decrypted data onto an external file. This file is compared with the original image to ensure the correctness of RC4 decryption.

Correctness of Sobel Edge-Detection

- Shown in Demo: Yes
- DSSC(s) Proved: 4
- Highest Level of Design module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Extract encrypted image data from file (produced by python code), and store in temporary registers (these simulate the SRAM).

    - Have temporary registers to capture edge detected results, that will later be saved in an external file. This file is compared with the output of python implementation of edge detection to ensure the correctness of our edge detection module.
    - Emulate the initiation of image processing by inputting image specifications + RC4 decryption key.
- No external or premade references are needed
- Pre-processing is needed
    - A python code that generates encrypted image will be run, and the output of this will be stored in a file. This will be used by the test bench as the input to the chip.
- Post-processing is required
    - A python code that generates the edge detected image will be run, and the output of this will be stored in a file. This will be used to determine the correctness of the output of our chip.
- Main verification Test Steps:
    - Image processing steps are initiated by inputting image specifications + RC4 decryption key data into chip.
    - Once RC4 decryption is complete (continuation from previous test), the decrypted image data is looped back and sent as an input to the edge detection module.
    - Store edge detected pixel data on temp registers.
    - After all the pixels have been edge detected, the edge detected image data is stored in an external file. This file is compared with the python-edge detected image to check for correctness of the edge detection module.

AMBA AHB-Lite Protocol interfacing

- Shown in Demo: Yes
- DSSC(s) Proved: 5
- Highest Level of Design module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Have test vector of samples of each type of bus transaction and correct chip response information
    - Chip response should be verified by checking against standards
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
    - Write image specifications + decryption key onto AHB slave
    - Ensure that these signals are held constant until ‘process complete’ signal is received
    - Emulate bus transaction sample from test vector
    - Check chip response against the correct response values in the test vector
    - Repeat steps 3-4 for all entries in test vector

Backup and Sub-Module tests

Chip holds processes until user inputs have been received (AHB-Lite Slave, MCU)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 1
- Highest Level of Design module(s) involved:
    - AHB-Lite Slave
    - MCU
- Test bench Expectations/Requirements:
    - Have test vector of samples of write instruction and correct response information
    - Use different dummy data for transaction/payloads to check that intended values are actually getting received
    - Response information should be verified by checking against standards
    - Response from AHB-slave to MCU and MCU to RC4 decryption/Edge-detection module should be grabbed to check if design started decrypting the image
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
    - Emulate bus transaction sample from test vector containing image description + RC4 decryption key to the be stored in MM-registers in AHB-slave
    - Check block response against the correct response values in the test vector
    - Repeat steps 1-2 for all the entries in the test vector.
    - Check AHB-Slave to MCU response against the expected value.
    - Check the MCU to RC4 decryption/Edge-detection module against the expected value.

Correctness of RC4 Decryption on variable image sizes (RC4 Decryption module)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 2
- Highest Level of Design module(s) involved:
    - RC4 Decryption module
- Test bench Expectations/Requirements:
    - Extract encrypted image data from file (produced by python code), and store in temporary registers (these simulate the SRAM).
    - Have temporary registers to capture decrypted results, that will later be saved in an external file. This file is compared with the output of python implementation of RC4 decryption to ensure the correctness of our RC4 decryption module.
    - Emulate the initiation of image processing by inputting image specifications + RC4 decryption key.
- No external or premade references are needed
- Pre-processing is needed

    - A python code that generates encrypted image will be run, and the output of this will be stored in a file. This will be used by the test bench as the input to the chip. This is repeated for images of 3 different sizes.
- Post-processing is required
    - A python code that generates the decrypted image will be run, and the output of this will be stored in a file. This will be used to determine the correctness of the output of RC4 decryption block. This is repeated for images of 3 different sizes.
- Main verification Test Steps:
    - All steps for the testing of RC4 decryption are repeated for images of 3 different sizes.

Correctness of Edge-detection on variable image sizes (Edge-detection module)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 2
- Highest Level of Design module(s) involved:
    - Edge-detection module
- Test bench Expectations/Requirements:
    - Original images are sent as the inputs to the edge detection module.
    - Have temporary registers to capture edge detected results, that will later be saved in an external file. This file is compared with the output of python implementation of edge detection to ensure the correctness of our edge detection module.
    - Emulate the initiation of image processing by inputting image specifications + RC4 decryption key.
    -
- No external or premade references are needed
- Pre-processing is not needed
- Post-processing is required
    - A python code that generates the edge detected image will be run, and the output of this will be stored in a file. This will be used to determine the correctness of the output of Edge-Detection block. This is repeated for images of 3 different sizes.
- Main verification Test Steps:
    - All steps for the testing of Edge-Detection (under sub-module section) are repeated for images of 3 different sizes.

Correctness of RC4 Decryption (RC4 Decryption Block)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 3
- Highest Level of Design module(s) involved:
    - RC4 Decryption Block
- Test bench Expectations/Requirements:
    - Test bench 1:

        - For RC4 Core supply an expected value generated by RC4 Generate Random Module (4 bytes for each byte supplied to the module)
        - Performs the decryption of the 4 bytes of encrypted information supplied to the module
    - Test bench 2:
        - For RC4 Generate Random module, supply rc4 key and run through just the initialization of the state array.
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
    - Test bench 1:
        - For sample image, using python code, save all values for each byte that are used for encryption in an external file.
        - Through the same code save the encrypted image in an external file
        - Use the first 4 bytes generated for encryption and the first 4 encrypted bytes and supply them to the test bench for RC4 core.
        - Compare the output of the test bench with the original 4 bytes.
    - Test bench 2:
        - Create the initialized state array through python code and separately through the rc4 module.
        - Compare those two arrays if they match

Correctness of Sobel Edge-Detection (Edge-Detection Block)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 4
- Highest Level of Design module(s) involved:
    - Edge-Detection Block
- Test bench Expectations/Requirements:
    - 12 bytes of pixel data stored in temp registers as a 3X4 matrix. The edge detection module performs edge detection on the two center pixels.
    - Temp registers to store the edge detected data of the two pixels. This is compared with the output of the python file to check for correctness.
- No external or premade references are needed
- No pre-processing is needed
- Post-processing is required
    - A python code that generates the edge detected image will be run, and the output of this will be stored in a file. This will be used to determine the correctness of the output of our Edge-detection Block.
- Main verification Test Steps:
    - Get first 12 bytes of data from the image.
    - Feed raw data into Edge-detection as if it came from Sample image data storage block.

- Check edge-detected data for the two-pixels with the output of python file.

Correctness of Sample Buffer Data (Sample Image Data Storage Block)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 4
- Highest Level of Design module(s) involved:
  - Sample Image Data Storage Block
- Test bench Expectations/Requirements:
  - Temp registers with pixel data are stored in test bench. This is used as the input to the sample image data storage module.
  - Test vectors to test the extraction of different pixel sets, along with expected pixel data for that test vector is stored in the test bench.
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
  - The sample image data storage module is supplied with one of the input test vectors.
  - The corresponding data that is read by the sample storage module is checked against the expected data for that test vector.
  - Steps 1-2 are repeated for remaining test vectors.

Correctness of AHB-Helper (AHB-Helper Block)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 5
- Highest Level of Design module(s) involved:
  - AHB-Helper Block
- Test bench Expectations/Requirements:
  - Have test vector of samples for RC4 read/write as well as Edge detection read/write, along with their expected outputs.
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
  - Emulate RC4 r/w.
  - Compare output lines with expected output data for current test vector.
  - Emulate Edge detection r/w.
  - Compare output lines with expected output data for current test vector.
  - Repeat steps 1-4 for all entries in test vector.

AMBA AHB-Lite Protocol interfacing (AHB-Master block, AHB-Slave block)

- Shown in Demo: Only if can't show using top level
- DSSC(s) Proved: 5
- Highest Level of Design module(s) involved:

    - AHB-Lite Slave Block
    - AHB-Lite Master Block
- Test bench Expectations/Requirements:
    - Have test vector of samples of each type of bus transaction and correct response information
    - AHB response should be verified by checking against standards
- No external or premade references are needed
- No pre/post processing is needed
- Main verification Test Steps:
    - Write image specifications + decryption key onto AHB slave
    - Ensure that these signals are held constant until ‘process complete’ signal is received.
    - Emulate read and write transactions by sending inputs directly into AHB master.
    - Check whether output lines match expected values.
    - Repeat steps 3-4 for all entries in test vector.