Lab 5

Congratulations! You are newly hired Embedded Engineers working for Karle Industries. Your project manager has tasked your group with creating an application which must meet the following requirements (they are described in more detail later):

1. Your application will read random numbers from a register interface that you create on the FPGA.
2. Your application will then compute a SHA256 checksum over the random data.
3. Your application will then transmit the SHA256 checksum to a server where the checksum will be verified for correctness (you must successfully compute and transfer checksums to cover 512MB of data).
4. Your application will maintain a heartbeat LED to verify that the unit is operational.
5. Your application will accept interrupts from the FPGA.
6. Because of performance and resource limitations, your application must use FreeRTOS & lwIP.

Your project team has been provided with a port of FreeRTOS for the TLL6219 Mezzanine board. A copy of this source code has been included in the lab bundle. You will need to extract and modify this source under the Linux environment. Your project team has also been provided with C source code for the SHA256 algorithm, which will need to be incorporated into your project. Lastly, your project team has been provided with a target server binary.

A little more detail…

1. Your application sends a “*Request Seed*” packet to the server1.
2. The server1 returns a packet to your application containing the seed value.
3. You seed your random number generator2 with the seed value.
4. You fill a buffer with random data as read from your random number generator2.
5. You compute a SHA256 checksum3 over the random data in the buffer.
6. You transmit only the SHA256 checksum3 to the server1 using a “*Checksum*” packet.
7. If the checksum is correct, the server1 will register the data as received.
8. Repeat from step 4.

1 Provided as lab\_server.exe. Default port is 2000

2 You will implement in your FPGA.

3 Provided in sha256sum.[ch]

### Requirement 1

Implement the follow Linear Feedback Shift Register as pseudo-random number generator. You will create a 32-bit register interface on the FPGA that provides random numbers when read. Your register should produce random numbers in accordance with the following LFSR ~~circuit~~ equation:

**random = (random >> 1) ^ (-(random & 1) & 0x80200003);**

With a seed value of 0xAAAAAAAA, your random number generator will produce the following stream of 32-bit values when read repeatedly:

0x55555555

0xAA8AAAA9

0xD5655557

...

### Requirement 2

Fill a buffer of random data. For example, if you filled a 10-byte buffer with data, it would look like the following (with seed value of 0xAAAAAAAA):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0x00000000** | 0x55 | 0x55 | 0x55 | 0x55 |
| **0x00000004** | 0xAA | 0x8A | 0xAA | 0xA9 |
| **0x00000008** | 0xD5 | 0x65 |  |  |

Note that words are stored in big-endian format. After the buffer has been filled, use the routines provided in sha256sum.c file to calculate a SHA256 checksum for the data buffer.

### Requirement 3

This requirement will be the most complicated to implement, since you are communicating over the network with a server. At a high level, this is the communication between your application and the server:

1. Your application will send a packet to the server requesting a seed value. In this packet, your application will also inform the server the how much data each checksum will cover. See figure 1.
2. The server will then return a packet containing the seed value for the client to use. See figure 2. This value is used to seed your random number generator.
3. Your application then calculates SHA256 checksums over blocks of random data with the size as specified in 1. Your application will then send only the checksum to the server. See figure 3.

Your application will communicate with the server over the UDP protocol (it is okay to hardcode the IP address and port of the server in your application). You will be using lwIP **version 1.3** as your network stack. You can view some sample applications that use lwIP at <http://cvs.savannah.gnu.org/viewvc/contrib/apps/?root=lwip>.

The following figures describe the packets that are transmitted between your application and the server. **All fields are 32-bits and in network byte order.**

#### Request Seed

This packed is transmitted from your application to the server. This causes the server to reset all the statistics for the source IP address. The server will also respond with the packet described in figure 2.

|  |  |
| --- | --- |
| **Offset 0** | Command value = 1 |
| **Offset 4** | Block size = 0 – 1023 |

Figure

The block size parameter is in units of 8 Kilobytes and is one less than the requested size. For example:

|  |  |
| --- | --- |
| **Block size parameter** | **Block size used** |
| 0 | 8KB |
| 1 | 16KB |
| … | … |
| 1023 | 8MB |

#### Seed Packet

This packet is transmitted from the server to the client in response to a “Request Seed” packet. The server transmits this packet to the source port used by the client with the “Request Seed” packet.

|  |  |
| --- | --- |
| **Offset 0** | Command value = 1 |
| **Offset 4** | Seed value |

Figure

#### Checksum Packet

This packet is transmitted from your application to the server.

|  |  |
| --- | --- |
| **Offset 0** | Command value = 3 |
| **Offset 4** | Count ID |
| **Offset 8** | SHA256 |
| **Offset 12** | SHA256 |
| **Offset 16** | SHA256 |
| **Offset 20** | SHA256 |
| **Offset 24** | SHA256 |
| **Offset 28** | SHA256 |
| **Offset 32** | SHA256 |
| **Offset 36** | SHA256 |

Figure

The “Count ID” field should start at 0 for the first checksum packet sent and increment for each subsequent packet. The SHA256 checksum will be the calculated checksum for the buffer.

#### Requirement 3 – Recap

1. A request seed packet is sent to the server, which also denotes the block size to use.
2. Server responds with a seed value packet.
3. Random number generator is seeded with value.
4. A count ID variable is set to zero.
5. A buffer of block size is filled with random data.
6. A SHA256 checksum is calculated over the random data.
7. The count ID & checksum is transmitted to the server.
8. The count ID variable is incremented.
9. Go to step 5.

### Requirement 4

Your application must maintain a visual pattern on at least one LED. This could be as simple as blinking an LED once a second. You will probably need to add LED access to your register interface.

### Requirement 5

This requirement is the second most complicated to implement. The FPGA must be programmed to interrupt the CPU when a push button switch is pressed. On receiving an interrupt, your application must print a list of all tasks running.

Implementing this requirement will require the reading of schematics and datasheets. See “*tll6219\_v3-0\_sch\_final.pdf*” net CPLD\_INT on page 5 to start your journey.

### Requirement 6

All the files in the FreeRTOS tarball are fair game for changes, although the files in “board/tll6219/” and “kernel/portable/GCC/ARM5\_MC9328MX21/” may be of particular interest. It is also recommended to look closely at board.c & main.c.

### Appendix

This is a list of functions that **may** be useful.

xTaskCreate

xSemaphoreCreateCounting

xSemaphoreGiveFromISR

vTaskSwitchContext

vTaskDelay

xSemaphoreTake

xSemaphoreGive

vTaskList

netconn\_new

netbuf\_new

netconn\_bind

netbuf\_alloc

netbuf\_ref

htonl

ntohl

IP4\_ADDR

netconn\_sendto

netbuf\_delete

netconn\_recv

netbuf\_copy

ethIsUp