Final assignment Embedded Systems

By Miroslav Gechev and Stef Otten

Date: 31-1-2019

# Introduction:

For Embedded Systems we were tasked with coming up with a good assignment that would illustrate our skills with programming FPGA’s and would highlight a good feature of FPGA’s. Because you can build your own system with handpicked components FPGA’s have the capabilities to be much faster then regular microcontrollers at big tasks. For our assignment we decided to implement a simple encryption tool that would apply the *Advanced Encryption Standard (AES)* to a binary input given by the switches on our Nios DE2-115 board. Afterwards the encrypted input will be displayed on the LCD. We also implemented the code on pc and Raspberry Pi to compare the encrypting times.

We’re under the impression that this assignment is a good way of showing of the functionality of our board because it makes great uses of its in- and outputs. The FPGA is also perfectly suited for this assignment as it takes a lot of time to run the input through all the steps necessary to encrypt. FPGA’s are also notoriously hard to hack, because of the limited connections to the outside world the programmer can choose to include in his system.

# In- and output:

When the program starts the user gets to input a binary value using the 16 switches on the board, this input also gets displayed on the LCD. This value gets converted to a hex value so our encryption method accepts it better. The program then goes through the steps to convert the input. It displays the time it took to convert and the encrypted value on the console and then start a decrypting process. Finally it shows the encrypted value, the time it took to encrypt, and the decrypted value and the time it took to decrypt on the console.

# Qsys:

Using the Qsys platform designer we build the system to our spec. For our program we’ll need:

* the processor to run our program,
* some on-chip memory to store our program,
* two P-I/O interfaces for user interaction,
* and a JTAG UART interface for communication with the host computer.

After picking the components the equivalent system of figure 1 will be realised.

Afbeelding met schermafbeelding

Automatisch gegenereerde beschrijvingThe Clock will be the 50Mhz clock that the DE2-115 board provides for us, and the NiosII processor is connected to the I/O and memory through the Avalon Switch Fabric which is an interconnection network automatically generated by Qsys.

Figure 1: System overview

Picking the components is an easy process, where most values can be left default. We set the width of the on-chip memory at 32 bits, and the width of the I/O responsible for the switches at 16 (we’re making use of 16 switches). Connections where made until all the error messages where resolved, and the base-addresses we had automatically generated. To avoid the tedious job of having to manually enter all the addresses in the pin planner we’ve included a completed pin planner file in our project directory.

# Eclipse:

Because we were determined to write our project in C opposed to VHDL or Verilog we needed to make a board support package that included a C-library. For this we followed chapter 4.1 and chapter 4.2 of the NiosII with leds tutorial provided by C.Slot and J.Stokkink.

# The Software:

The software of our project is based on the tiny-aes library, which is the smallest implementation of AES in C we could find.

When you start the program we run the initialise functions to init the LCD and the encryption and decryption functions. If everything is fine we proceed in the infinite loop where the input from the switches is read. The input is read and stored in the global plaintext[0]variable. The current process and the stored variable are presented on the LCD. We start the clock to measure the program run time, and call the test\_encrypt\_ecb\_verbose() function. The variable is converted and printed hexadecimally on the console before it gets taken through the 4 encryption steps using the hardcoded key. After the encryption process the encrypted value is stored back in the plaintext[0]variable and printed on the console together with the amount of ticks it took to run the process.

Then the entire program is run inverted, so the cipher is decrypted using the key. Finally also the amount ticks it took to decrypt is printed together with the original value in hex.

**Port to Raspberry pi**

To port the software to Raspberry Pi we only had to exclude some defines and contains we had to put in to make the program comply with Quartus. Instead of taking input from switches it just takes input from command line using scanf().

# Conclusion:

In conclusion we implemented the AES for the FPGA, Raspberry Pi and the pc. We measured the resulting encrypting and decrypting time for each in clock-ticks and these where the results :

Table 1: FPGA test results

|  |  |
| --- | --- |
| Clock ticks | |
| Encrypt | Decrypt | Input |
| 262 | 159 | 16 |
| 165 | 161 | 16 |
| 171 | 167 | 16 |
| 173 | 159 | 16 |
| 166 | 160 | 16520 |
| 165 | 161 | 16520 |
| 156 | 168 | 16520 |
| 156 | 162 | 16520 |
| 165 | 161 | 213640 |
| 153 | 165 | 213640 |
| 170 | 160 | 213640 |
| 159 | 163 | 213640 |

Average encrypt-time: 171.75

Average decrypt-time: 162.16

Table 2 Pi test results

|  |  |
| --- | --- |
| Clock ticks | |
| Encrypt | Decrypt | Input |
| 468 | 749 | 16 |
| 473 | 822 | 16 |
| 477 | 739 | 16 |
| 477 | 741 | 16 |
| 467 | 814 | 16520 |
| 533 | 773 | 16520 |
| 471 | 738 | 16520 |
| 461 | 773 | 16520 |
| 520 | 742 | 213640 |
| 510 | 795 | 213640 |
| 494 | 859 | 213640 |
| 474 | 784 | 213640 |

Average encrypt-time: 485.42

Average decrypt-time: 777.42

Table 3 Pc test results

|  |  |
| --- | --- |
| Clock ticks | |
| Encrypt | Decrypt | Input |
| 14 | 34 | 16 |
| 19 | 48 | 16 |
| 14 | 52 | 16 |
| 18 | 21 | 16 |
| 16 | 57 | 16520 |
| 17 | 59 | 16520 |
| 17 | 47 | 16520 |
| 14 | 57 | 16520 |
| 15 | 21 | 213640 |
| 14 | 20 | 213640 |
| 19 | 49 | 213640 |
| 19 | 49 | 213640 |

Average encrypt-time: 16.33

Average decrypt-time: 42.83

From these tables it becomes apparent that the pc uses the least amount of clock ticks. We expected the FPGA to be the faster because we designed the system for our program. The pc turned out to be faster because it uses a modern Intel Core i7 processor and is more efficient. It can process a heavier workload in every clock tick. The program on the pc has a lot more cache memory to work with.

The FPGA did turn out to be faster then the Raspberry Pi, confirming our suspicion that a full dedicated system would be faster than a not-so powerful solution like a Pi or microcontroller.

# Recommendations:

In this chapter we describe a few things we would have liked to do if the opportunity was there.

* Print the encrypted value on the LCD; in the end we didn’t get around to implementing this because once a value is encrypted it won’t fit on a single row of the LCD, and splitting the value evenly into two rows turned out to be problematic.
* Enter keys and values through keyboard; We couldn’t get around to including a keyboard with our set up because the board simply wasn’t available enough to do so. We had to share the board with two other people in the class and it proved very difficult to reserve the board for programming.
* Use IP blocks; Using IP blocks we could have made the program a lot faster, and it would have saved us a lot of work. Sadly we couldn’t find any source material explaining how this was done, and didn’t have the time to figure it out ourselves. We instead prioritized cleaning up our code.

# Discussion:

Throughout the project we had a lot of trouble working in the Quartus environment, getting error messages that seemed almost random and gave us no relevant information. One of the biggest breakthroughs was turning off the previously recommended smart-compilation setting. Uploading the board support package also proved to be difficult. The standard command used in our source material had a “2” as specifier, because in the example the fpga was the second device in the chain, but in our chain it was the first. After realising this we could start C-programming. Implementing the tiny-aes library proved to be confusing, as we started to work with a lot of conversions (from char to int, decimal to hex, binary to decimal etc.)

The Altera DE-115 also does not inherently support floating point values, making it hard for us to time the process needed for the encryption (which is less then a millisecond).

# Sources:

Slot, C. & Stokkink, J. (2018, April). *Embedded tutorial – NiosII with leds*[Tutorial]. Used on 20th of December 2018, from https://leren.saxion.nl/bbcswebdav/pid-2112388-dt-content-rid-36420982\_4/institution/LED/LED\_Opleiding\_ELT/3e\_jaars\_modulen\_ELT/Embedded%20Systems/Tutorial-Emb-Nios2leds-v22.pdf

Jensen, M (2019, January). *tiny-AES-c* [Repository] Used throughout the project, from https://github.com/kokke/tiny-AES-c