Available online at www.sciencedirect.com

ScienceDirect

Procedia - Social and Behavioral Sciences 102 (2013) 464 – 471

Procedia
Social and Behavioral Sciences

6th International Forum on Engineering Education (IFEE 2012)

Teaching Creative Digital Hardware Design

Norhayati Binti Mohd Zainee [a,*], James M Noras [b]

[a] *Faculty of Engineering and IT, INTI International University, Putra Nilai, Malaysia*
[b] *School of Enginerring Design and Technology, University of Bradford, Richmond Road, Bradford BD7 1DP, UK*

Abstract

Engineering undergraduates not only need to learn facts, but also how to be creative in the open-ended situations they will encounter in their professional careers. Our final year Honours module gives students a grounding in digital systems design, mainly using VLSI for design entry and simulation. The second half of our module is a design exercise, which has straightforward aspects, but which allows motivated students to undertake progressively open-ended investigations. Our educational framework is guided by recommendations of professional bodies promoting excellence and encouragement of creativity in engineering development.

Keywords: Digital electronics; design; VHDL; university education; competence; creativity;

1. Introduction

Since January 2005, students have been studying and graduating from an Honours BEng course in Electrical and Electronic Engineering run jointly by INTI and the University of Bradford (UoB), at the campus which is now the home of INTI International University (IIU). Originating in Bradford, this course has been accredited by the IEE and now the IET continuously for over 40 years, and the recommendations of the Quality Assurance Agency for Higher Education are used to define and control the standards, methods and outcomes of the modules studied [1].

In IIU, the same principles are used to ensure that the required professional level of graduates is achieved, in particular that graduates have “a systematic understanding of key aspects of their field of study, including acquisition of coherent and detailed knowledge” and can “apply the methods and techniques that they have

* Corresponding author. Tel.: +06-7982000-2502
E-mail address: norhayati.mzainee@newinti.edu.my

1877-0428 © 2013 The Authors. Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
Selection and/or peer-review under responsibility of Professor Dr Mohd. Zaidi Omar, Associate Professor Dr Ruhizan Mohammad Yasin, Dr Roszilah Hamid, Dr Norngainy Mohd. Tawil, Associate Professor Dr Wan Kamal Mujani, Associate Professor Dr Effandi Zakaria.
doi:10.1016/j.sbspro.2013.10.762

learned to review, consolidate, extend and apply their knowledge and understanding, and to initiate and carry out projects" [1].

These two outcomes might be described respectively as the possession of knowledge, and the ability to apply that knowledge, so that students can not only pass examinations, but can use subject-specific knowledge in a creative and original way. These aspects are emphasised by the Engineering benchmark statement [2], which says that graduates will "have strategies for being creative, innovative and overcoming difficulties by employing their knowledge in a flexible manner", and by the UK-SPEC of the Engineering Council [3] which includes as part of the standard for BEng degrees that graduates should be able to "apply appropriate theoretical and practical methods to the analysis and solution of engineering problems" and to "implement design solutions, and evaluate their effectiveness".

The degree programme is designed as a whole to equip graduates with knowledge and skills, and to prepare them to continue to update and improve their professional abilities and understanding. This paper briefly sets out how, as part of the digital electronics strand of the course, students are encouraged to achieve these outcomes in the context of learning the VHDL design system and how to apply it.

In the next section we review the digital provision of the course, in particular the module dealing with VHDL: this is the VHSIC Hardware Description Language, where VHSIC means Very-High-Speed Integrated Circuit, which is in widespread use as a design and simulation tool, accepted as an industry standard [4]

Then we discuss how students master the twin requirements, as set out above, of acquiring knowledge of VHDL and then of demonstrating their ability to use it. Finally we explain our assessment strategy and draw some brief conclusions.

2. University-level digital electronics modules

In recognition of the central position nowadays of digital electronics in its many aspects, the BEng course contains several modules which focus on aspects of digital technology. The most significant are shown in table 1.

Table 1. Modules with Significant Digital Electronics Content

Stage or Year	Module Title	Credits
1	Robotics	20
2	Digital Electronics Fundamentals*	10
2	Digital Electronics Design*	10
2	Embedded Systems	10
2	Group Design Project	20
3	Digital Signal Processing	10
3	Project	30
3	Digital Design Using HDL*	10

These 8 modules, together with another 24, make up the 360 credit programme. While some relate to microprocessors or digital applications, those marked with an asterisk provide the main exposure to digital electronics at the gate level, including theory, laboratory and simulation work with discrete gates and programmable chips in the form of FPGAs.

The first courses cover gate-level aspects and computer hardware, with Boolean logic, state machine design and general treatments of technologies such as TTL and CMOS. Simulation classes cover the Altera Hardware Design Language (AHDL) used with Quartus software for design capture [5], and various PIC microcontrollers are used in robotics and general interfacing. Project work allows students to explore more open-ended applications of these technologies, but generally in the first two years, work is relatively pre-defined, with limited scope for design exploration and creativity.

The stage 3 individual project, required by the IET for accrediting purposes, needs 300 hours of individual work at Honours level, permitting original work at that level, and allowing students to demonstrate their potential as innovators and designers. Also in the final year, students can learn other languages for digital design capture and for design simulation, validation and implementation. AHDL, Xilinx and Verilog are discussed, but the main work, described next, is on VHDL and its applications.

3. Learning VHDL and how to use it

The third year course, Digital Design Using HDL, has the main aims of first getting the students familiar with the syntax of VHDL through demonstrations and laboratory tasks, clearly defined to introduce the methods of circuit description and simulation, which we describe here. Secondly we present the students with their first serious exposure to digital design using VHDL, synthesis, testing and performance evaluation, as explained in the next section. These cover the twin aspects of the Engineering standards mentioned above.

3.1. Learning the syntax of VHDL – Entities and Architectures

VHDL files are composed of Entity-Architecture pairs. The Entity is analogous to a symbol for the design, holding all of the external connections of a logic block. The Architecture describes the connectivity, function and implementation of the design.

An example of an entity declaration is given below:

```
entity   fulladder  is
         port (X: in bit; Y: in bit; Cin: in bit; Cout: out bit; Sum: out bit);
end      fulladder;
```

VHDL allows the creation of multiple architectures for each entity. This feature is useful for simulation and for project team environments in which high-level designs of the system interfaces and lower-level architectural descriptions are produced by different engineers, or when designers want to experiment with different methods.

A simple architecture may contain declarations (for example signals, components, local functions and constants) followed by statements which describe a logic block’s structure or functionality, as illustrated by the following example:

```
architecture concurrent of fulladder is
         begin
         Sum <= X xor Y xor Cin;
         Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end concurrent;
```

3.2. Basics of circuit construction with VHDL

The students complete 10 lab experiments before starting on their project work, whereby they learn the basics of VHDL coding step by step, writing code and simulating to check design correctness. The students start with simple gates, then combine these to form logic circuits such as multiplexers, adders, decoders and comparators.

Next the students move on to the basic sequential blocks, flip-flops and latches, using these to build and test registers and counters.

4. Student design tasks

The task for the design project must have the potential to stretch each student, but must not require extensive background study. It must therefore have fairly straightforward aspects, but be reasonably complex, and must be open-ended, such as evaluation of different hardware implementations and of performance measured by maximum clock speed and data throughput.

One category of application that meets these criteria is the implementation of cryptographic algorithms. There are many of these, and a good detailed source of several dozens of these is provided by Schneier [6-7]). We find that algorithms with the Feistel structure, described below, are particularly easy to understand, but set appropriate and varied challenges for implementation.

4.1. Feistel systems in general

Many ciphering algorithms use a particular structure, called a Feistel structure. The nature of this structure is that a block of data to be encrypted is divided into a left and a right half, for example say 32 bits each. We provide a brief tutorial example.

4.1.1. A 2-round Feistel cipher

Suppose (L0,R0) is the message to be encrypted. and (L2,R2) is the resulting ciphertext. The following pseudo-code explains the routine. Here the symbol "^" means the bit-wise XOR operation.

Table 2. The Feistel structure for encryption

Stage	Pseudo-code	Description
first round of encryption	$L_1 = R_0$	After the first round R_0 is unaffected, but we rename it L_1.
	$R_1 = L_0$ ^ $f_1(R_0)$	R_1 is the encrypted version of L_0, which is generated using a function f_1 of R_0..
second round of encryption	$L_2 = R_1$	After the second round, we don't change R_1, but we call it L_2.
	$R_2 = L_1$ ^ $f_2(R_1)$	R2 is the encrypted version of L_1, using a function of R_1.

After these two rounds, L_0 and R_0 have been encrypted into L_2 and R_2. Each round alters half of the bits, producing a new $L_n = R_{n-1}$, and a new $R_n = L_{n-1}$ ^ $f_n(R_{n-1})$. The functions f can be very complicated, and generally involve a key or keys. However, this kind of ciphering, with a Feistel structure, is easy to reverse. To see this, just consider the 2-round cipher again:

Table 3. The Feistel structure for decryption

Stage	Pseudo-code
first round of decryption	$L_1 = R_2$ ^ $f_2(L_2)$
	= L_1 ^ $f_2(R_1)$ ^ $f_2(L_2)$
	= L_1 ^ $f_2(R_1)$ ^ $f_2(R_1)$
	= L_1
	$R_1 = L_2$
second round of decryption	$L_0 = R_1$ ^ $f_1(L_1)$
	= L_0 ^ $f_1(R_0)$ ^ $f_1(L_1)$
	= L_0 ^ $f1(R_0)$ ^ $f_1(R_0)$
	= L_0
	$R_0 = L_1$

The key concept is that "something ^ anything ^ anything = something", so that the two XOR operations just cancel each other out. The important thing about the Feistel cipher is that each stage only alters half of the data, so reversing the process is straightforward, however complicated the function f. Figure 1 shows the algorithmic structure.

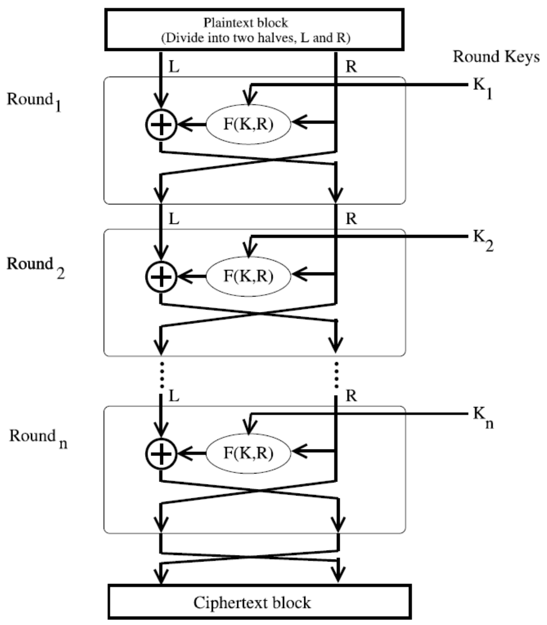

Fig. 1. Structure of a Feistel algorithm for encryption[8]

4.2. The TEA algorithm in particular

The TEA algorithm uses simple operations (addition and exclusive or) to make a potentially very fast design [9]. Its structure is easy to grasp, and requires no advanced knowledge, so is highly suitable for the design exercise for our module. Wikipedia gives a useful presentation [10]. The following section gives the C code for the encryption of data.

4.3. C code

This routine encodes with keys k[0] – k[3], data in v[0] and v[1].

```
void code(long* v, long* k) {
unsigned long     y = v[0], z = v[1], sum = 0,                     /* set up */
                  delta = 0x9E3779B9,                              /* a key schedule constant */
                  n=32;
while (n-->0)     {                                                /* basic cycle start */
                  sum += delta;
                  y += ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
                  z += ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
                  }                                                /* end cycle */
v[0] = y; v[1] = z;}
```

4.3.1. Decode routine

Decoding uses a similar structure with differently ordered keys. Also, the variable "sum" has to be initialised to what it was at the end of the ciphering algorithm. That is, if there were 32 rounds, sum would become 32* delta, or delta<<5, as $32 = 2^5$. Subsequently sum is decremented by delta each round.

4.4. MATLAB

In order for the students to generate test data, they are provided with MATLAB code. The code for one round of ciphering is shown below:

```
function [y, z] = tea_en_round (y, z, sum, k0, k1, k2, k3, nbitz);     % one round of TEA
part1_1 = mod(bitshift(z,4),2^nbitz);
part1 = mod((part1_1 + k0),2^nbitz);                                   % use k[0]
part2 = mod((z + sum),2^nbitz);
part3_1 = mod(bitshift(z,-5),2^nbitz);
part3 = mod((part3_1 + k1),2^nbitz);                                   % use k[1]
combine = bitxor(part1, bitxor(part2, part3));
y = mod((y + combine), 2^nbitz);                                       % end of first half of round
part1_1 = mod(bitshift(y,4),2^nbitz);
part1 = mod((part1_1 + k2),2^nbitz);                                   % use k[2]
part2 = mod((y + sum),2^nbitz);
part3_1 = mod(bitshift(y,-5),2^nbitz);
part3 = mod((part3_1 + k3),2^nbitz);                                   % use k[3]
combine = bitxor(part1, bitxor(part2, part3));
z = mod((z + combine), 2^nbitz);                                       % completes round of ciphering
end;
```

In using this code, the binary (or hexadecimal) data need to be converted to and from decimal in the calling routine. We used the MATLAB functions hex2dec and dec2hex: the code of the calling routine is not included here because of its length (some 30 lines) but is available from the authors.

In Figure 2, y is the input at the top left, and z is fed in from the right, which, after processing by the various operations and keys, is added to y. After the cross-over, the now modified y is fed in from the right, the result added to z, unmodified up to that point.

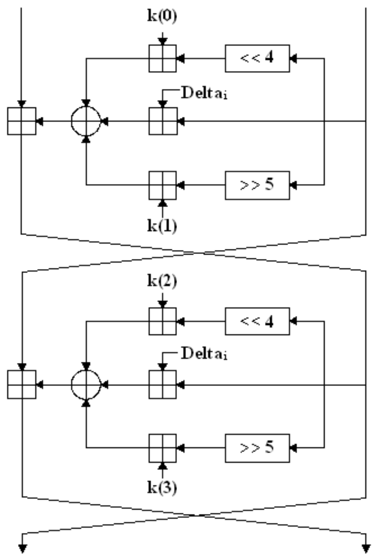

Fig. 2. Structure of the TEA algorithm [Tiny_Encryption_Algorithm, 2012][10]

5. Objectives, attainment and assessment

The coursework task is to design hardware in VHDL which will execute the TEA algorithm both for encryption and decryption.

It would be possible for students to implement the C-code more or less directly, but they are encouraged to do this only as a check of their primary design, which should rather be built up from components, each synthesised and thoroughly tested for functional and timing accuracy before being used in a bottom-up design structure

There will be several stages for the design:

- to produce the elements that will be used in the design, for the various operations required (e.g. shifting, adding, xoring).
- to produce the design to carry out a single round of encryption or decryption, for word lengths of 4, 8 and 16 bits.
- to produce a design that will carry out multiple rounds of encryption and decryption.

The checking at each stage is essential: no one would use a design that had not been proved to be correct, so failure to give good evidence of checking will result in a poor mark for the part or parts of the design concerned.

The task is specified so that students who show basic competence and succeed in implementing designs that are correct in their building blocks, and which can demonstrate ciphering and deciphering for at least one cycle, can, with a good report, pass the exercise.

However, students are encouraged to extend their work in various ways:

- to parameterise the size of data and keys
- to insert registers to make the design capable of pipelining with consequent investigations of the effects on propagation delays, clock speeds and throughput.

- to investigate the use of gate-level modelling, for example using full-adders (or other elements) in comparison with using the built-in "+" operation.
- to implement multiple rounds of encryption or decryption, by using the "generate" function or otherwise.

All these elective investigations help to distinguish individual student's work, and allow a grasp of the methods and potential of VHDL modelling and circuit design methods to be shown, so that motivated students can extend their skills and knowledge, with the bonus of being able to earn very high marks.

6. Conclusions

We have outlined a scheme for teaching students both basic and advanced aspects of VHDL and its applications, in a way that gives students a clearly delineated introduction to the topic, and then allows them freedom to explore design aspects in an open-ended way. Thus we bolster their knowledge and understanding, while letting them explore innovation in a well-supported environment, preparing them for future challenges.

References

[1]The UK Quality Code for Higher Education. *A brief guide*. http://www.qaa.ac.uk/Publications/Information And Guidance/Pages/quality-code-brief-guide.aspx, 2012.

[2] Subject benchmark statement. *Engineering*. http://www.qaa.ac.uk/Publications/InformationAndGuidance/Pages/Subject-benchmark-statement-Engineering-.aspx, 2010.

[3]UK-SPEC. *The UK Standard for Professional Engineering* Competence (UK SPEC).http://www.engc.org.uk/ecukdocuments/internet/document%20library/UK-SPEC.pdf, 2011

[4] EDA Industry Working Groups. http://www.vhdl.org/, 2012

[5] Altera Corporation. http://www.altera.com/, 2012

[6] Schneier, Bruce. *Applied Cryptography*. (2nd ed.) .Bruce Schneier, John Wiley & Sons, 1996

[7] Schneier, Bruce. *Applied Cryptography Source Code*. http://www.schneier.com/book-applied-source.html, 2012.

[8]Kak, A. *Block Ciphers and the Data Encryption Standard*. https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture3.pdf. 2012.

[9] Wheeler, David J.; Needham, Roger M. *TEA, a tiny encryption algorithm*. Lecture Notes in Computer Science (Leuven, Belgium: Fast Software Encryption: Second International Workshop) 1008: 363–366. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.281. 1994.

[10] Tiny_Encryption_Algorithm. http://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm, 2012