The Evolution of a Digital Logic Lab

Jacky Baltes
Computer Science Dept.
University of Calgary
baltes@cpsc.ucalgary.ca

Cameron Patterson
Alberta Microelectronic Centre
Calgary, Alberta
cdp@cpsc.ucalgary.ca

Abstract

This paper describes different technologies that were used in a VLSI design course at the university of Calgary. The main goal of this paper is to show how the advent of new technology allows students to spend more time on design capture, logic simulation, and the design of test vectors, as opposed to the tedious tasks of implementing/fabricating a design and a test environment. This trend has lead to more and more complex and interesting projects. In recent years, the students used VHDL to create a behavioral description of their circuit and synthesize a schematic from it. The synthesis targets Actel or Xilinx FPGAs. The example project is the design of a GCD circuit, which the authors selected because of a number of desirable characteristics: most importantly, (a) it is complex enough to allow the students freedom in their design, and (b) it can easily be adapted to the available hardware resources. The paper includes a small example of the conversion from an algorithm into a finite state machines, one of the crucial steps in the design phase. In the future, we hope to use configurable hardware (the Algotronix CHS2x4) with a powerful connection to a host computer. This will allow students even greater flexibility in their design, since they can choose which parts are implemented in hardware and which are done through software.

1 Introduction

In this paper, the authors describe and evaluate their experiences in preparing digital logic labs for a fourth year undergraduate course (CPSC 521) at the University of Calgary. The goal is to show how the course labs were substantially improved through the advent and use of new hardware and software technology. In particular, more time can be spent on design of the circuit and test sets and less time on tedious wiring, schematic capture, and the creation of test environments.

The authors' perspective is mainly that of the lab assistants. Therefore, this paper emphasizes the practical component of the course. The labs for the course are held at the Alberta Microelectronic Centre (AMC), a company which endeavors to improve cooperation between university and industry.

CPSC 521 is a fourth year course offered to computer science majors who would like to specialize in hardware. The pre-requisites are CPSC 321 (Introduction to Logic Circuit Design) and CPSC 421 (Digital Systems Design). The first course discusses boolean logic, logic minimization, basic arithmetic circuits, and storage elements. The lab component of this course consists of implementing various small circuits using SSI and MSI TTL components.

CPSC 421 examines the structured design of control logic and data paths. The students use schematic capture and simulation tools provided by the AMC for more complex designs (e.g., 8×8 multiplier).

CPSC 521 (VLSI design) surveys the design and construction of CMOS VLSI circuits. The students design, implement and test substantial circuits using VLSI technology. Since CPSC 521 is a one semester course, there are only 10 weeks or 30 hours available for lab sessions. Students usually work in groups of two because of the limited resources (e.g., workstations, software licenses).

The next section describes and compares the technologies used in the labs over the last years. The technologies are compared according to the amount of time allocated to the different phases of the project. Section 3 summarizes the experiences and also previews what technology will be used in the future.

2 Technologies

This section describes and evaluates previously used technologies. The technologies are shown in chronological order as they were used in the course.

The paper compares the amount of time that an average student spends on the different phases. In general, the time spent by students in the lab can be divided into three phases: (a) design capture and simulation, (b) implementation or fabrication time, and (c) creating the test environment and testing the design.

The comparison contrasts the suitability of different technologies to our goal (teaching VLSI design) and does not reflect on the applicability of these technologies to real world problems. However, undoubtedly some of the features, e.g., the quick fabrication time, is also of importance in more practical circumstances.

The design capture and logic simulation phase consists of finding a solution to the given problem. The student creates a description of the intended circuit and satisfies himself that the proposed design is correct through software simulation. The available design time limits the complexity of the project that can be attempted in the course. However, students enjoy the design process and would like to attempt more complex designs, as opposed to simple ones.

The implementation or fabrication phase consists of building a physical realization of the design. Since this phase is mainly a translation process, it is often experienced as tedious by students. The design is known, and all that is required is to convert it into a physical representation. For the fabricated IC technologies discussed in subsections 2.2 and 2.3, the fabrication process was done for the students. However, they were unable to start testing until the chips were fabricated, so that in these cases the turnaround time became critical.

In the testing phase, students produce a test environment (which might include adding switches and LEDs) and test the project. Since students already have convinced themselves of the correctness of the design through logic simulation, the creation of the test environment is experienced as a nuisance.

It can be argued that the creative process is in the design and logic simulation phase, and that the tedious phases of fabrication and testing may be omitted. However, the authors do not share this view. A number of real world constraints only arise because of the need to fabricate and test the chip. Any course on VLSI design would be incomplete, if only the design capture and simulation phase was done. Furthermore, students get a feeling of satisfaction from seeing their designs work in a "real world" environment, that can not be achieved through logic simulation.

2.1 Breadboarding

Prior to 1990, the students used breadboarding to implement their designs. One advantage of this method was that the components (7400 family TTL ICs), were readily available. Also, after implementation, the design could immediately be tested, since only minor changes are required to the implementation, for example adding LEDs and DIP switches.

However, this method suffered significant disadvantages. Most importantly, the long fabrication time. Due to the large amount of work necessary for the fabrication, the projects were simple adders and counters. This meant that the design was straight forward. More care had to be taken in the selection of test vectors to aid in testing and debugging the design. However, testing was limited by the small design.

2.2 Multi-project MPGAs

In 1990, the students used the LSI Logic tools for schematic capture and logic simulation (SC & LS). The students were broken up into groups of up to four students and built separate modules of a serial line interface with builtin hardware compression. The first module received input values and converted them into a huffman code. The huffman code was passed on to a variable length parallel to serial converter, that sent the data over a serial line. At the other end, a variable length serial to parallel converter reads in the data, decodes it, and converts it back into the original bit pattern.

Once the designs were finished, multi-project chips were created at the AMC mask-programmable gate array fabrication facilities in Edmonton. This setup created two problems: (a) fabrication could not be started until the last group was finished with their design, and (b) the turnaround time was almost six weeks. Due to the long delay in fabrication, insufficient time for testing was available.

2.3 Direct-Write Gate Arrays

In 1991, the AMC and Simon Fraser University collaborated to provide fast prototyping of CMOS gate arrays. Dr. Albert Leung, a researcher at SFU, had developed a direct-write system for patterning the metallization of gate array ICs. Rather than using traditional masking and photolithographic techniques, a laser "draws" the desired pattern directly on the die. The Plessey MHD single-metal-layer gate array was used, which contained 100 logic cells, horizontal routing underpasses, and 40 I/O cells. These resources were sufficient for the students to implement a digital clock (seconds and minutes) with enable, synchronous reset, and a scan path.

The IC design and implementation process is fully described in a report available from the AMC (see [ea91]). It consists of the following main steps:

1. The AMC provides workstations with Silvar Lisco's CASS schematic editor and LSI Logic's LSIM and LWAVE simulation tools.
2. AMC Calgary staff perform placement and routing using the Silvar Lisco GARDS system, and produce CIF layout files.
3. The SFU QuickChip^{TM} process customizes the gate array die. The wafers are pre-fabricated with a single aluminum layer. Photo-resist is applied and exposed to a deep-blue laser in a rasterized manner. The resist is developed, and used as an etch mask for the metal. Finally, the remaining resist is removed.
4. AMC Edmonton staff inspect, separate, and package several dies for each student group.
5. Students test the ICs using a breadboard environment.

2.4 Actel FPGAs

Actel FPGAs were first used in 1992 and then again in 1993. The labs used the Actel 1010 parts, one time programmable FPGAs with 1200 gates. The Actel FPGAs are row-based architectures, that is, they provide special routing facilities to distribute signals to different cells in a row. A cell or logic module in the Actel 1000 family is a simple multiplexor based module with 8 inputs and 1 output. It can implement any 2-input boolean function, many 3-input functions, and some 4-input functions or a latch.

Although different methods were used to create a schematic (schematic capture (1992) vs. VHDL synthesis (1993)), in both cases the students used logic simulation to test their designs. After successful simulation, the students used the Actel place-and-route software to back-annotate the simulation netlist with the actual delay information. This additional step is necessary since the signal delay is determined by the number of anti-fuses in the path, which is unknown until the design is mapped to the Actel A1010A FPGA. After re-simulating their design, the FPGAs were programmed.

2.4.1 Schematic Capture

Students used Workview on PCs for schematic capture and simulation of their design. The project was to design a stopwatch with enable, count-up, count-down, and asynchronous reset. A report describing the project in more detail is available from the AMC [ea92].

The simplicity of the project and the effort required to change the schematics discouraged students from experimenting with different designs. Many students perceived schematic capture as tedious and not very instructive.

Furthermore, a special testing environment had to be created by the students, consisting of a breadboard with four LEDs, and a number of buttons to select the different functions of the clock. This task was also time consuming, and left little time for the design of an efficient testing strategy or suitable test vectors.

2.4.2 VHDL Synthesis

In 1993, the second author was the instructor for CPSC 521. The course lectures were revised significantly from the previous years (see [ea93]). Since computer science students are more comfortable with algorithms than ohms, the transistor-level design of CMOS circuits was de-emphasized. Instead, the course material concentrated on the technologies and methodologies that speed up the design and implementation of ICs. These include: (a) giving an algorithmic or behavioral description of a circuit in VHDL, (b) synthesizing logic from VHDL, and (c) mapping the logic to an FPGA architecture. As a result of the positive experience in using VHDL, the authors created training material for the Canadian Microelectronics Corporation, which is available through the CMCcache.

The course labs gave the students hands-on exposure to the above topics. The following three limiting factors were reasonably balanced: (a) the circuit size and complexity that could be completed by average students in the time available for labs, (b) the speed and capacity of VHDL Designer V2.05 running on 33 MHz 80386 PCs with 8 Mbyte RAM, and (c) the capacity of the target FPGAs (1200 gate Actel A1010As). In addition, VHDL Designer was found to be robust enough for student use.

The default student project was to compute the greatest common divisor (GCD) of 2 n-bit numbers. The algorithm to compute the GCD is based on an algorithm described in [Knu73] and shown in table 1. The algorithm employs three well known facts about the gcd and can be explained to students in a short time. No background in number theory is required. It is important to note that it is given to them in a high level format (pseudo-code) that they are familiar with.

```
GCD (integer n1,n2)
{ integer k;
  k = 0
  while even(n1) and even(n2)  /* Loop 1 */
     n1 = n1/2;
     n2 = n2/2;
     k = k + 1;

  while even(n1)               /* Loop 2 */
     n1 = n1/2;

  while even(n2)               /* Loop 3 */
     n2 = n2/2;

  while (n1 != n2)             /* Loop 4 */
     if (n1 > n2) then
        n1 = n1 - n2;
        while even(n1)
           n1 = n1/2;
     else
        n2 = n2 - n1;
        while even(n2)
           n2 = n2/2;

   while (k != 0)              /* Loop 5 */
      n1 = n1*2;
      k = k - 1;
}

Mult/Div by 2 => shift left/right
even(n) => lsb of n (n[0]) = 0
```

Table 1: GCD algorithm

The GCD algorithm has a number of desirable features. Firstly, the IO requirements are small. The input consists of two n bit numbers provided in parallel. A serial input would require fewer IO pads, but requires the students to change the bit for each clock cycle. This is a very error prone operation, if done manually. Secondly, n can be scaled easily to the available resources, for example, given fewer IO cells, 4 or 6 instead of 8 bit numbers can be used. Thirdly, the circuit does not contain complex combinational logic, only shifts left and right, single bit tests, etc. The most expensive operation is the greater than comparison in loop 4. Most importantly, all functions are part of the library and do not have to be created separately.

However, our main reason for choosing this problem is that since the algorithm was intended as a sequential computer program, a large number of optimization for speed or area are possible. These optimization include but are not limited to:

- combine loop 1,2,3 into one loop.
- execute loop 2 and 3 in parallel.
- unrolling loop 1, 2 or 3.
- replace loop 2 by a possible swap. At most one of loop 2 or 3 is executed, since at least one number is odd after loop 1.
- reuse loop 2 and 3 in loop 4.
- replace one of the inner while loops in loop 4 by a swap.

As computer science majors, the students were quickly able to learn the syntax of the VHDL subset that they needed to complete the project. To start, we used the VHDL designer tutorial to show students how to create a simple combinational circuit (a seven segment display driver) and a finite state machine (a simple pattern matcher). The students went through the tutorial quickly, but had significant problems in converting an algorithmic description (the one shown in table 1) into synthesizable VHDL. Once students learned how to convert the algorithm into a finite state machine, they finished and tested the VHDL design in a short amount of time.

Table 2 is an example of this process. It shows a straight forward implementation of loop 4 (shown in table 1) in VHDL. It creates a finite state machine using parallel ifs. Simple arithmetic expressions such as shifts, add, and subtract can be directly typed into the source file. More complex ones such as multiplication and division have to be implemented in other ways. The GCD algorithm, however, only requires arithmetic expressions that are part of the synthesis library. State 1 simply skips to the end of the loop if one of the inputs (`n1`,`n2`) is 0. This is done to speed up the calculation as well as to avoid an infinite loop in states 3 or 4.

State 2 finds the larger one of `n1` and `n2`, replaces the maximum by the difference, and moves to the corresponding state (3 or 4). States 3 and 4 are equivalent to the embedded while loops in loop 4 of the GCD algorithm. If `n1` and `n2` are equal, the loop is terminated.

```
...
------------------- State 1 ---------------------------
if (n1 = 0) or (n2 = 0) then -- skip while loop if one number is 0
   State <= 5;
end if;

------------------- State 2 ---------------------------
if State = 2 then
   if n1 = n2 then          -- parallel ifs with mutually exclusive conditions
      State <= 5;
   end if;                  -- If equal, terminate loop by moving to exit state

   if n1 > n2 then
      n1 <= n1 - n2;
      State <= 3;           -- move to while loop state for n1
   end if;

   if n2 > n1 then
      n2 <= n2 - n1;
      State <= 4;
   end if;
end if;

------------------- State 3 ---------------------------
if State = 3 then
  if n1(0) = '0' then       -- n1 is even ?
     n1 <= shiftrum(n1,1);  -- shift right to divide by 2
  else
     State <= 2;            -- otherwise goto main while loop
  end if;
end if;

------------------- State 4 ---------------------------
if State = 4 then
  if n2(0) = '0' then       -- n2 is even ?
    n2 <= shiftrum(n2,1)
  else
    State <= 2;
  end if;
end if;

------------------- State 5 ---------------------------
if State = 5 then           -- we are done with loop 4
...
```

Table 2: VHDL source file for loop 4 of the GCD algorithm

After synthesizing the design and simulating it using ViewSim, the Actel place and route software was used and chips were fabricated. Unfortunately, the Actel chips proved to be to small to implement the complete design, if optimized for speed or broken up into more than one module. This means, that students had to combine the complete algorithm into one finite state machine rather than into separate modules. The instructors had discouraged students from creating large, hard to understand modules and had encouraged them to break the design up into smaller modules, to simplify the design and debugging. This divide and conquer technique is also at the heart of most programming languages (procedures and functions). Additionally, in contrast to programming language compilers, synthesis times grows extremely quickly once the modules are over a certain size. As a rule of thumb, the different entities should be about one page long. However, the overhead in handshaking and local registers made the resulting designs too large for the Actel chip. Therefore, some students only implemented the main loop (loop 4) of the algorithm, which meant that their chip was limited to computing the GCD of two odd numbers. However, by using VHDL, these changes could be made very quickly, so that all students were able to test at least part of their design.

The students also realized that trying to optimize a design to reduce the area is a sometimes frustrating endeavor. The place and route software is non-trivial to visualize and it is hard to predict the effect that certain changes to the design have in terms of area used. For example, changing the state encoding from a one-hot to a binary generally reduces the area, but especially if only a small number of states are used, can also increase the required area. Fortunately, VHDL allows a student to quickly change the encoding (edit a few lines in your VHDL source file), so that both methods can be tried out.

2.5 VHDL and Xilinx FPGAs

Viewlogic's VHDL Designer was again used in 1994 (see [ea94]), but the students used the larger capacity Xilinx XC4003 FPGAs. The Xilinx FPGAs are SRAM programmed symmetrical arrays of logic blocks. Each logic block contains two flipflops, two 4 input function generators, a dedicated carry logic. Each LookUp function generator can instead be used as 16 bits of RAM.

After a small tutorial and some worked examples, the students were able to start on the GCD project. To ameliorate the problems students had with designing the control logic, the lab assistant worked on a small case study, the problem of normalizing floating point numbers. Given an 8 bit input number f and an exponent e, shift f left until there is a 1 in the most significant bit and decrement the exponent for each shift. This algorithm was expressed as a simple while loop. Using this case study, the students learned to convert an algorithm into a finite state machine, which was one of the major problems students had in the previous year. Students also tried to optimize their designs for speed or area which gave them a feeling for the different tradeoffs in optimization, as well as give them some expertise in trying to determine how a given change will affect the area. Another advantage was the fact, that students were shown one complete iteration of the design cycle. Furthermore, the work done in the case study could very easily be adapted to provide them with loop 1, 2, and 3 of the GCD design. Lastly, it showed them how to implement proper handshaking, which is not part of the original problem specification, but is necessary so that a chip can communicate with the external world.

However, bugs in the viewlogic tools hindered the progress. Also, the Xilinx chips still were too small to implement the complete design as separate modules, so most groups only implemented the main loop in hardware.

Xilinx provides prototype boards with two 7 segment displays, two buttons, one reset button, and eight dip switches. The board also contains a small prototyping area. This means that only an additional bank of switches had to be added to create the testing environment. Students could spend more time on the choice of good test vectors.

We feel that the use of synthesis and programming/testing an FPGA in CPSC 521 is a nice complement to the sole use of schematic capture and simulation in CPSC 421. The students are already familiar with the basic Workview tools employed in the CPSC 421 labs (e.g., Viewsim). This is advantageous because there is a significant amount of time required for the students to become productive with logic synthesis. In particular, the students must structure their VHDL code so that it can be efficiently processed by the synthesizer.

2.6 Algotronix CHS2x4

In 1995, the university of Calgary intends to use recently purchased Algotronix CHS2x4 boards. The CHS2x4 is an add on card for the PC/AT bus. Each CHS2x4 card contains 8 FPGAs of SRAM configurable logic. Each FPGA contains an array of 32×32 logic cells. Each cell can implement any boolean function of two variables or a D-latch. The Algotronix FPGA is a sea of gate architecture, that is there are no dedicated routing matrices. Each cell receives its input from any of the four neighbors or one of two global signals.

The eight FPGAs are connected forming a two by four matrix. Thus, each board contains 8,192 configurable units. This number can be doubled by connecting up to two CHS2x4 boards back to back.

In addition to the FPGAs a CHS2x4 contains up to 2 MB of static local memory. The board provides an interface to the host computer, allowing access to secondary storage etc. The configuration as well as the local memory is accessible to the host computer. Thus, logic designs can be designed, implemented, and tested without touching the hardware. The tight coupling between the host computer and the FPGAs has the great advantage that designs that require more input/output (e.g., image processing hardware or search engines for artificial intelligence) can now be implemented. These designs would be impossible if the data was to be provided through dip switches and there is not sufficient time to design external local memory modules.

In contrast to other FPGA architectures, Algotronix also provides a C-library, that allows the configuration and control of the FPGAs through C programs. This, in conjunction with the fact that the FPGAs are SRAM based, allows even greater flexibility using the Algotronix boards, since they can be quickly reconfigured during execution. Therefore, the hardware does not limit the size of the project any more. Instead, only the size of the largest module is limited. For example, given the GCD algorithm described in table 1, loops 1, 2 and 3, in fact form a pre-processing stage, loop 4 the main processing stage, and loop 5 the post processing stage. Therefore, the GCD algorithm can be solved on the Algotronix boards in the following steps:

1. load FPGA configuration for loop 1,2,3
2. Setup input values
3. compute functions and store result of loop 3
4. load FPGA configuration for loop 4
5. load results of loop 3 and store result of loop 4
6. load FPGA configuration for loop 5
7. load results of loop 4 and compute final result

The FPGAs must only be able to hold the different modules (loop 1/2/3, loop4, and loop5) as opposed to the complete project (loop1/2/3/4/5). A second advantage of the direct access provided by the host computer is that in previous years, students had no choice in the implementation medium. The complete project had to be implemented in the provided hardware. With the Algotronix boards, a closer match to our goal in teaching complete system design is possible. Now students can choose to implement different parts in hardware or software, given design constraints such as speed or available hardware. For example, a student might choose to implement the post processing (i.e., loop 5) in software rather than hardware to speed up the computation by pipelining the hard and software.

Although this combination of hardware and software makes it appealing as a teaching tool, the software support is not as impressive. Algotronix supports schematic capture using popular systems such as OrCad or Viewlogic. However, there is no place and route tool for the Algotronix boards and no high level description language, such as VHDL. Different places are working on these, however, including some work tentatively scheduled to begin at the university of Calgary this summer. Nevertheless, the first author implemented and tested a simple pattern matcher in about three days. It is unclear, how this scales up to irregular structures as the GCD algorithm used in the previous section.

3 Discussion

A short summary comparing different technologies is shown in Table 3. The time distribution column indicates the number of labs allocated for each of the three phases (design capture and simulation, implementation or fabrication, creating the test environment and testing). This column shows an interesting trend. Using breadboarding and fabricated ICs, the students spend most of the time on implementation or waiting for fabrication. Using FPGAs, significantly more time was spent on the design capture and logic simulation phase, and the testing phase. It is important to note that using schematic capture and VHDL synthesis, the students spend roughly the same effort on design capture and logic simulation. However, as can be seen in the project column, using VHDL permits more complex projects. The configurable computer made possible through the Algotronix CHS2x4 boards holds much promise, if decent software support can be obtained and we are looking forward to interesting and complex designs.

Era	Technology	Software Tools	Year	Representative Project	Time Dist.
7400 Family TTL	Breadboarding	none	< 90	Adder	1/8/1
Fabricated ICs	MPGA	SC & LS	90	Huffman En/Decoder	3/5/2
	Direct Write	SC & LS	91	Alarm clock	3/4/3
FPGAs	Actel	SC & LS	92	Count down clock	5/2/3
	VHDL+Actel	Synth & LS	93	GCD chip	5/2/3
	VHDL+Xilinx	Synth & LS	94	GCD chip	6/2/2
	Algotronix	undecided	95	undecided	7/1/2

Table 3: Technologies and projects used in CPSC 521

References

[ea91] Cameron Patterson et al. Selected student reports from CPSC 521, winter term 1991. Technical report, Alberta Microelectronic Centre, 1991.

[ea92] Jacky Baltes et al. Selected student reports from CPSC 521, winter term 1992. Technical report, Alberta Microelectronic Centre, 1992.

[ea93] Jacky Baltes et al. Selected student reports from CPSC 521, winter term 1993. Technical report, Alberta Microelectronic Centre, 1993.

[ea94] Jacky Baltes et al. Selected student reports from CPSC 521, winter term 1994. Technical report, Alberta Microelectronic Centre, 1994.

[Knu73] Donald Ervin Knuth. *The art of computer programming.* Addison-Wesley: Reading, Mass., 1973.