AC 2012-3729: TEACHING DIGITAL DESIGN IN A PROGRAMMABLE LOGIC DEVICE ARENA

Dr. Christopher R. Carroll, University of Minnesota, Duluth

Christopher R. Carroll received a bachelor's degree from Georgia Tech, and M.S. and Ph.D. degrees from Caltech. After teaching at Duke University, he is now Associate Professor of electrical and computer engineering at the University of Minnesota, Duluth, with interests in special-purpose digital system design, VLSI, and microprocessor applications.

Teaching Digital Design in a Programmable Logic Device Arena

Abstract

Programmable logic devices have revolutionized the way in which digital circuits are built. FPGAs (Field Programmable Gate Arrays) and CPLDs (Complex Programmable Logic Devices) have become the standards for implementing digital systems. FPGAs and CPLDs offer much higher circuit density, improved reliability, and fewer system components when compared with traditional digital design using discrete small-scale or medium-scale integrated circuits, all of which make programmable logic devices very attractive to the digital designer. However, these devices hide important details involved in understanding digital fundamentals, and the resulting hardware is really more of a computer-generated black box than it is a carefully crafted, fine-tuned design. Creativity in the design is less visible when using FPGAs or CPLDs, and designers are not rewarded as satisfyingly for "elegant" solutions to design problems.

FPGAs and CPLDs implement solutions to digital design problems quickly and economically, both qualities that are important in an industrial setting. However, in an educational setting, the solution is not as important as understanding how the solution is reached, and these programmable devices automate and hide that process, making them less attractive as educational tools. Teaching digital design in a programmable logic device arena requires the instructor to inform students what is going on behind the scenes in the synthesis software. Otherwise, digital design degenerates into just another programming exercise, albeit using a hardware description language rather than traditional software languages.

During Fall semester 2011, programmable logic devices were used for the first time[1] as the basis for lab exercises in a second semester, advanced digital design laboratory at UMD, replacing design using discrete digital integrated circuits. The experience exposed some limitations imposed by the technology. For example, when circuits must avoid logic hazards (momentary "glitches" during transitions) as in asynchronous finite state machine design, FPGAs cannot be used properly, and CPLDs must be coerced into working by clumsily "fooling" the synthesis software. These specific digital circuit designs cannot be mapped cleanly to programmable devices without some innovative techniques. This paper reveals some of the author's experiences in adapting his digital design laboratory to the programmable logic device arena.

Programmable logic devices, though attractive to the experienced designer, can be awkward to use in certain educational settings. Digital design instructors must be aware of their limitations. Instructors must find creative ways around the limitations, and must restrain themselves from being brainwashed by the glitz of FPGAs and CPLDs. This paper identifies techniques for maintaining the excitement and rewards of creative digital design within the confined restrictions of a programmable logic device arena.

Setting

The techniques discussed here have been used in the laboratory for a second course in digital circuit design. This course depends on students having mastered digital circuit fundamentals at the gate and flip-flop level in an earlier, prerequisite course. Here, the focus is on using higher-level building blocks, traditionally classified as "MSI" (Medium Scale Integration) components, to design functional units found in digital computers and elsewhere, such as arithmetic circuits, data structure implementations, and memory systems. Both this course and its prerequisite digital design course have recently migrated from lab exercises based on physically wiring discrete digital integrated circuits on breadboards to configuring programmable logic devices through hardware description languages, to "build" the circuits being designed by students. This migration follows trends in modern digital design technology, and prepares students for the kind of digital design environment they will face in today's industry. However, this migration does not come without cost. Key fundamentals in digital design are hidden by the programmable logic device technology, and temptation exists to skip over those fundamentals to sprint to the goals of enhanced circuit complexity and capabilities that the programmable logic technology allows. Skipping the fundamentals, however, deprives students of the ground-floor understanding that is essential to secure a solid foundation in digital design.

Problem

Designing digital circuits can be an easy, mechanical exercise, once the basics are mastered. After all, it's just a matter of getting all the 1's and 0's in the right place at the right time, and even if you guess at the answer, you have a 50% chance of being correct, right? That is the reason that automating the design process through hardware description languages and software that reads those languages to configure programmable logic devices has been so successful. On the surface, everything is cookbook and easy. Complex systems are assembled by replicating lots of simple circuits, a process at which digital computers excel. Circuits designed by students in the lab for an introductory digital circuits course fall into this category of design. Once the process is understood, producing designs to satisfy requirements for introductory course lab exercises is straightforward, with no real thought involved, and such a process is automated easily. As long as the designer can live under the restrictions imposed by this surface-level design, life is easy. A good example of this type of restriction is the synchronous clocking philosophy employed in many sequential digital systems. "Synchronous" implies exactly one system clock signal, and all changes that occur in the system must happen in response to transitions on that one clock signal. Many times, such restrictions are fine and appropriate.

However, this second digital circuit design course addresses topics that require below-the-surface understanding of the material. Combinational circuits no longer are "memory-less" as they are treated in introductory classes. Clocking of sequential circuits extends beyond the limitations of synchronous clocking. These second-level characteristics of digital circuits require an understanding of how the circuit works that goes beyond descriptions presented in introductory courses. Designing circuits in an arena where simplifying assumptions do not necessarily hold sometimes prevents programmable logic devices from serving as direct solutions to design problems without some carefully applied accommodations.

Example

A good example of how blind application of digital design basics leads to failed designs is found in the design of a simple transparent D-latch. A straightforward solution to such a design is shown in Figure 1, with the function involved displayed in the accompanying Karnaugh map.

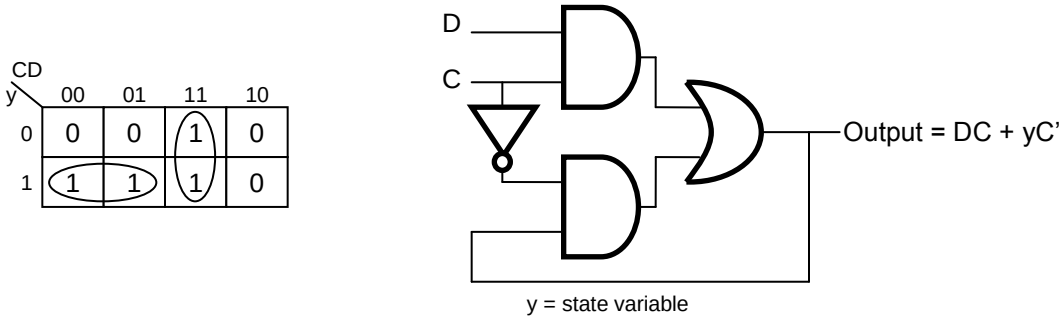

Figure 1. Straightforward (erroneous!) transparent D-latch design

As can be deduced from the circuit diagram in Figure 1, the output of a transparent D-latch is just equal to the D (data) input when the latch is "transparent," i.e. whenever the C (control) input is logic 1. When C is logic 0, the latch maintains whatever value was present on its output when C became 0. However, this simple circuit can fail due to the presence of a static-1 logic hazard in the design. The effect of the hazard can be observed by starting with D=C=y=1 and then letting C change to 0. From the description of the latch operation, it is clear that the output should stay at 1. However, if the delay in the inverter shown in Figure 1 is much larger than the delay in the other gates, then as C changes from 1 to 0 there can be a time when both AND gate outputs are 0 because of the slow change on the inverter. If that happens, then the OR gate output becomes 0, and then the lower AND gate output stays at 0 even after the inverter output eventually changes to 1, leading to failed behavior of the latch.

Logic hazards are always easy to fix. When implementing functions in sum-of-products form as in Figure 1, one must be sure that every pair of adjacent 1's in the Karnaugh map is included together in some implicant of the function. That is not the case in Figure 1, leading to the presence of the static-1 logic hazard. However with the addition of an extra implicant in the function, requiring an additional AND gate and additional input to the OR gate, the logic hazard can be eliminated, as shown in Figure 2.

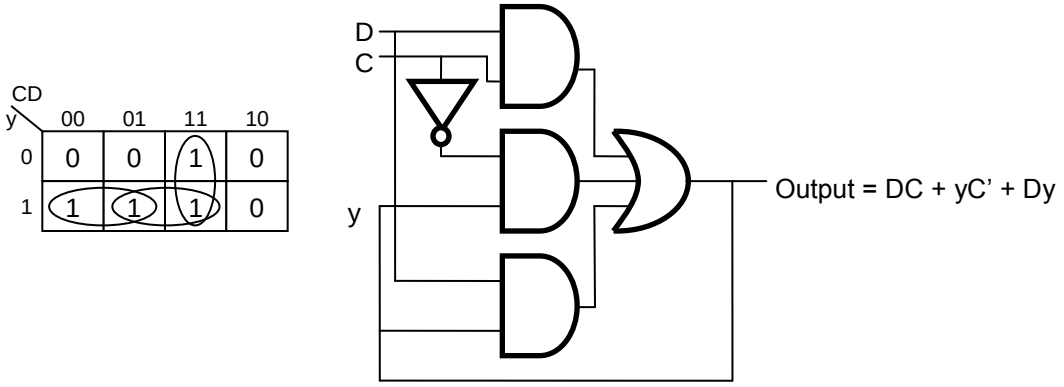

Figure 2. Transparent D-latch design with logic hazard removed

Unfortunately, fixing logic hazards sometimes requires addition of hardware that is "redundant" when hazards are not considered, as is the case in Figure 2. Simple logic minimization removes the additional hardware needed to fix the hazard. Only by understanding hazards and the problems they can present does one realize the importance of the otherwise redundant circuitry.

Programmable Logic Devices

Today there are two dominant types of programmable logic devices in use. They are the Field Programmable Gate Array (FPGA) and the Complex Programmable Logic Device (CPLD). Both devices are easy to configure (using appropriate design software) to implement a wide variety of digital system designs. However, the internal structure of the two devices is very different.

FPGAs consist of a large number of configurable "logic blocks" connected internally via an interconnection network. Each logic block consists of circuitry that implements a combinational logic function and a storage element to build sequential circuits. The combinational logic is implemented with a table look-up scheme using RAM to store the truth table for the desired function to be generated. This approach essentially makes each individual minterm of the function being implemented a separate implicant of the function, as indicated in Figure 3.

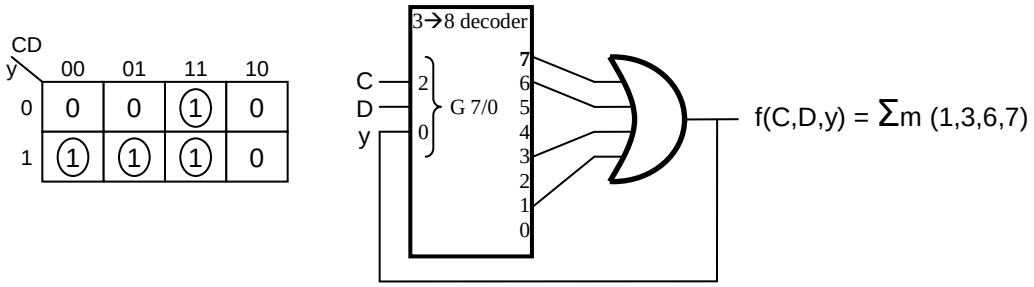

Figure 3. Visualization of FPGA attempt to implement a transparent D-latch

Clearly this implementation of the design shown earlier in Figures 1 and 2 contains multiple logic hazards, as the rule for adjacent 1's in the Karnaugh map being together in shared implicants is not followed here. Generally, FPGA structures are not suited to implementing combinational functions when logic hazards must be avoided. FPGA designs must be constrained to systems that are clocked in such a way that hazards can be ignored.

CPLDs consist of a large number of "macrocells" connected by an internal interconnection network. Each macrocell implements a combinational function and a storage element just like the logic block of the FPGA, but here the combinational function is implemented differently. A Programmable Logic Array (PLA) structure is used, consisting of an array of AND gates with programmable inputs to generate desired implicants of the function, and an OR gate to combine together the desired implicants to form the function being generated. With this approach, implicants can be specified as needed to avoid logic hazards, and implementations such as shown in Figure 2 above are possible.

Unfortunately, digital designers are generally insulated from the actual CPLD configuration by complex configuration software packages. Usually this is good, since the designer usually does not care about the details of the CPLD implementation, only that the correct functions and interconnections are produced. The designer can specify his functions in any way desired, including simply supplying the truth tables for the functions. The configuration software then massages the functions, simplifying them down to minimal form, for implementation in the CPLD. But, as shown in Figures 1 and 2, minimal form is not always the best solution.

In order to generate CPLD solutions to problems like the transparent D-latch described above, the CPLD configuration software must be coerced into behaving in ways that it would not choose if left to itself. It is presumably possible for a technical person with the right knowledge to edit the netlist files produced by the configuration software and "manually" configure the CPLD, but that is a huge task that is beyond the expertise and even the interest of a typical digital system designer. Instead the software must be "tricked" into behaving properly.

Coercion

To implement the transparent D-latch design described above in Figure 2 using a CPLD, the configuration software must be fooled into cooperating. If Figure 2's circuit is input to the software, the software will minimize the function, removing the redundant implicant that is needed to avoid the logic hazard, and implement the function shown in Figure 1 instead, resulting in a circuit that may malfunction.

A solution to this problem that has been used successfully in this course's lab experiments is shown in Figure 4.

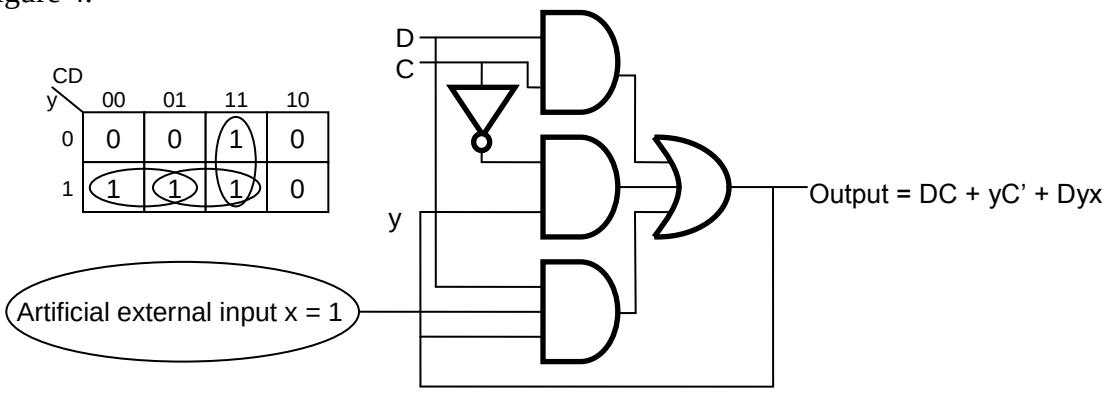

Figure 4. Modified Figure 2 circuit to fool CPLD configuration software

By including an artificial external input in the otherwise redundant implicant for the function, the software is forced to include this implicant in its implementation. During circuit operation, it is the user's responsibility to ensure that this artificial external input is set to an appropriate value (logic 1 in this case) to generate the function properly including the redundant implicant. The same sort of technique can be used in dual form if product-of-sum format is used for implementing the function required. In that case, the artificial external input would have to be a 0 for proper operation of the circuit.

More Problems

The solution to the logic hazard problem described in Figure 4 above has worked well in lab assignments for this second course in digital design. However, there are other problems.

Another pitfall present in many asynchronous finite state machine designs is a second type of hazard, known as an "essential" hazard. Essential hazards arise as a consequence of delays in combinational logic that swamp the necessary delay in the feedback path for the state variable(s) in a system. The only way to fix essential hazards is to "pad" the delay for the state variable(s) in the feedback path of the circuit to ensure that the feedback delay is long enough to overcome other delays that are present in the combinational logic of the system.

At the present time, the author is not aware of any technique to pad delays in specific signal paths of a digital circuit design using CPLD configuration software. Again, by editing the configuration software output netlist files, it should be possible to manually configure the CPLD to achieve any desired performance, but such manipulation of the CPLD configuration is beyond the expertise held by the typical digital system designer. Additional experience and experimentation will be required to discover ways to coerce the configuration software to allow for correction of essential hazards in asynchronous finite state machine designs for CPLDs.

Summary

FPGAs and CPLDs have revolutionized digital design. These programmable logic devices allow much more circuit complexity at much reduced cost and improved reliability over older design techniques. However, FPGAs cannot be used to implement circuits when logic hazards are a concern, as there is no way to avoid logic hazards with the FPGA combinational logic generation strategy. CPLDs can be used in such a setting, but the configuration software for the CPLDs must be coerced into preserving redundant implicants that are needed to eliminate logic hazards.

There are still other potential design troubles such as essential hazards that cannot be solved even with CPLDs, as far as this author is aware. Further research and experience may uncover ways to fool configuration software into solving these problems in the future.

In the meantime, FPGAs and CPLDs are excellent solutions to most digital design problems. Students appreciate the opportunity to use state-of-the art tools and techniques for digital system design. However, these are not perfect solutions, and when certain underlying design requirements arise, these devices sometimes cannot be used.

References

1. Carroll, C. R., "Adapting Digital Design Instruction to a Programmable Logic Device Setting," *Proceedings of the 2011 ASEE North Midwest Section Meeting*, Duluth, MN (2011).