Best FPGA Development Practices

February 20, 2014 [1]

Charles Fulks
Intuitive Research and Technology Corporation
Huntsville, Alabama

RC Cofer
Avnet
Melbourne, Florida

Abstract

This paper focuses on improving FPGA design quality by presenting practical and efficient FPGA development best practices. This paper identifies a starting set of design practices and development processes that reduce the probability of FPGA design errors. Reducing common mistakes saves labor costs during FPGA development and, more significantly, during system integration. This set of techniques can be used to assist in the evaluation of competing FPGA development proposals.

[1] This paper was first presented as class ESC-405 at the Design West Conference in 2012

Intuitive Research and Technology Corporation
Best FPGA Development Practices

Table of Contents

List of Figures

Source Listings

Intuitive Research and Technology Corporation
Best FPGA Development Practices

Preface

We develop opinions based on our personal experience (and reading). Experience is usually field related. The opinions developed in academia are therefore necessarily different from those developed in a safety critical environment. The opinions expressed in this paper are those of an Field Applications Engineer (FAE) and a engineer from a high reliability field[2].

Designers tend to repeat similar classes of mistakes. The intent of this paper is to identify some of the common mistake areas and contributing factors. We also present some techniques to avoid them.

The authors use VHDL in many examples in the paper due to their familiarity with that language. This does not imply an opinion regarding Verilog.

1 Process (or how to spend less time debugging)

Process has been well known in the software field for a long time; decades. The problem is that some *Process People* make the process so onerous[3] that design engineers avoid it, and circumvent it, at all costs. This is a travesty because process is the only way to ensure a quality (defect free) design in complex systems.

At a previous employer I insisted that my FPGA team follow my/our process with the caveat that we could *change the process anywhere it did not add value.* "Add value" is the key phrase and criteria for any process.

A reasonable question is "Why is this way right?" The answer is that this way insists the engineer "**Think**, **Discuss** the design with a peer[4], then **Act**." Many engineers skip the "Think" and "Discuss" steps.

1.1 Why Use a Process

Computers were originally people performing calculations. To increase efficiency, we invented mechanical calculators. Further inventions produced electronic calculators with *hard wired* algorithms. A new algorithm required physically re-wiring the system. This allowed simple algorithms only.

Next came the invention of software; separating the algorithm from the hardware. The complexity of software is very high with respect to hardware[5]. We recognized the difficulty of finding defects in highly complex systems and developed a process to ensure quality of the complex system components, the software.

Then, programmable logic (FPGAs) separated the digital design from the physical hardware. The digital design complexity is on par with software meaning that *you cannot completely test an FPGA design.* We need to follow a process to ensure the quality of complex system components: software and FPGAs. The product test verifies only the functionality tested. It cannot exhaustively test for latent defects. *You can't test quality into a digital design or software program.* Furthermore, a process reduces the variability of cost, schedule, and technical performance.

A Process is not a replacement for thinking or a method to disengage from the realities of the program. The project lead must know when and why to deviate from the process. The process does not replace domain knowledge; it does not replace skilled, motivated engineers.

1.2 How Much Process?

How much process needed is primarily determined by the cost of a failure. A dropped video frame may not be noticed. A dropped phone call will annoy customers and have a detrimental effect on your brand. A latent security system vulnerability may have dire financial consequences. An uncontrolled flight control surface on an aircraft is unacceptable.

The process must *add value* to your design team and it must be *appropriate* to your product.

[2] Your mileage may vary.
[3] Caveat: an onerous process is completely appropriate for safety critical systems.
[4] The peer must be knowledgeable in the art in order to *add value.*
[5] A circuit board may have 10,000 nodes while a software application may have millions of lines of code.

In Advisory Circular 20-152, the Federal Aviation Administration (FAA) recommends DO-254. RTCA/DO-254 *Design Assurance Guidance For Airborne Electronic Hardware* defines "simple electronics" and states that if it is not simple, it is complex.

> A hardware item is identified as simple only if a comprehensive combination of deterministic tests and analyses appropriate to the design assurance level can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior. **When an item cannot be classified as simple, it should be classified as complex** [emphasis added].

The NASA Assurance Process for Complex Electronics addresses FPGAs as follows.

> Complex electronics are programmable devices that can be used to implement specific hardware circuits. The devices that are included under the label of complex electronics are:
> - Complex programmable logic device (CPLD)
> - Field Programmable Gate Array (FPGA)
> - Application Specific Integrated Circuit (ASIC)
> - System on a Chip (SOC)
> - Field Programmable System Chip (FPSC)
> - Variations of FPGAs and ASICs
>
> In the term complex electronics, the complex adjective is used to distinguish between simple devices, such as off-the-shelf ICs and logic gates, and user-creatable devices.
> Note that firmware (which is essentially software stored on a read-only device) is not considered complex electronics. The integrated circuit (e.g. EEPROM) is simple electronics. The program stored in that device is software, which has a defined assurance process in place.

1.3 A Minimal Process

Most, if not all, of the process description standards allow for different levels of process depending on the safety criticality of the system. This section describes minimum process for FPGA development for non-safety critical industries shown in Figure 1.1.

Prioritized Feature List. We can be reasonably sure that a project's requirements will change at some point in the program. A "Prioritized Feature List" attempts two things: it avoids requirements changes due to misunderstanding of what to build, and it can help us determine what to work on first. In other words, what would we want done as soon as possible for a customer demo.

Digital Design. The digital design must be well defined prior to attempting to capture it in Hardware Description Language (HDL). The level of documented detail is debatable; however, if you cannot describe your design with timing diagrams, block diagrams, and text, how can you expect to describe it accurately with HDL? Documenting a design is key to ensuring that someone else, including a future version of you[6], can quickly gain an understanding of the design. *Any design not accompanied by documentation is not acceptable.*

Design Capture and Test Bench Capture. If possible, different engineers should capture the design and the test bench so that a requirement misunderstanding does not reside in both. A code inspection with the author, 1 or 2 senior engineers and 1 or 2 junior engineers helps ensure the code matches the

[6] This phrase courtesy Jack Ganssle.

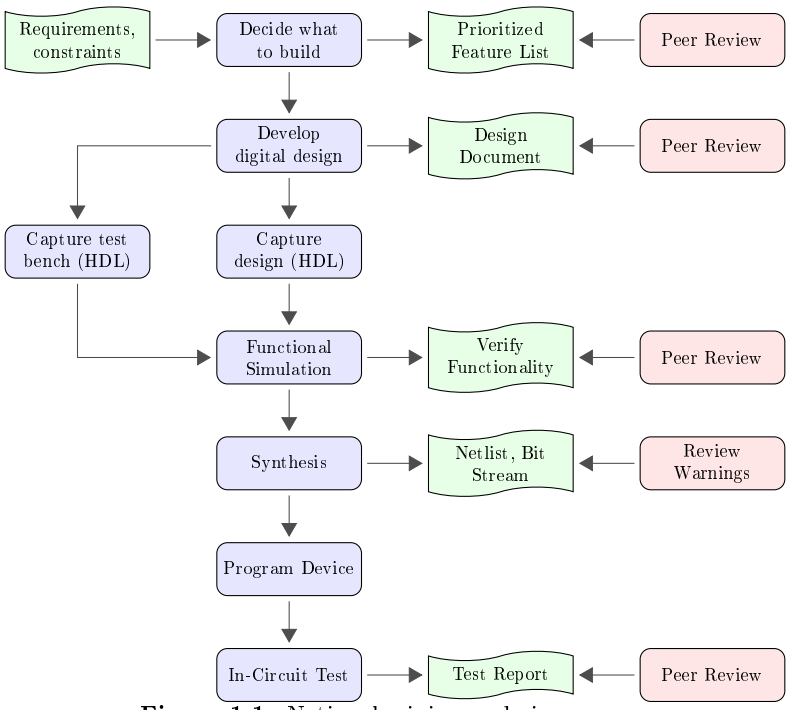

Figure 1.1: Notional minimum design process

design and educates the less experienced engineers. Note that this design review *is not a requirements review.*

Functional Simulation. Developing even the simplest digital design will benefit from some simulation to allow prediction of in-system behavior. Developing good design and simulation practices can pay off in time saved down the road. A HDL simulation can be extremely valuable in verifying a design. It should not be used as a crutch to assist in the design process. Do not create a "rough draft" of a design, then iteratively simulate and modify it until it works since this can lead to a lack of understanding of how and why it works.[7]

In-Circuit Verification. The objective of integration and test is to ensure a design meets all requirements. From an FPGA perspective, we want to verify that the outputs of the module simulation and the module instantiated in the FPGA match for a given input. All requirements must have a post synthesis or post Place and Route (PAR) test. You may have "perfect" HDL; but how do you know for sure what the tool did with it?

1.4 Revision Control: How to avoid "Which version did we ship?"

There are a class of questions that can add significant stress to the product team.

- Which version of the software or FPGA did we ship?
- Do you have a copy of the missing file?
- Which file did you use?
- Did you overwrite my changes? Can I restore an earlier version of a file?
- Which version of the tools were used to implement the design?

A revision control system creates a data base that contains each version of each file and, if used effectively, can avoid the majority of this type of question.

[7] Experimental HDL to understand a syntax or principle is not product development and does not fall under this standard; hack away! However, don't do it in the product development folder.

In an FPGA design there is more to the story than the source code. A revision control system provides a disciplined and documented way to capture key design artifacts.

Two excellent open source revision control systems are Apache™ Subversion® and Concurrent Versions System (CVS).

1.5 Coding Guidelines: A *short* collection of techniques that minimize errors

A coding guideline is a *short* collection of techniques that act collectively to minimize errors. HDL coding guidelines are intended to be enforced by the design team with appropriate engineering judgment. Strict adherence by all FPGA code is not required. The main purposes are to *reduce the probability of logic errors* and to enhance portability among designers.

The priority of the designer should be to *adhere to the manufacturer recommended coding styles* for FPGA HDL capture. Manufacturer coding guidelines are readily available from their websites. Two examples are Altera *Internal Memory (RAM and ROM) User Guide*[1] and Xilinx *Synthesis and Simulation Design Guide*[15]

The following example VHDL capture guidelines are excerpts from *VHDL Capture Guidelines* available at Intuitive Research and Technology Corporation.

Default Values	
Rule: Generics shall not have default values except at the top level of the design[a]. [a] In rare cases it may be prudent to include a default value for a generic.	*Reasoning:* This requires the designer to set generic values at instantiation. This avoids instantiation errors related to assuming generic values.
Signed and Unsigned Libraries	
Rule: Do not use the IEEE `std_logic_arith`, `signed`, or `unsigned` libraries. Use the `ieee.numeric_std` library and declare signals signed or unsigned.	*Reasoning:* Avoiding these libraries forces the designer to keep track of the "signedness" of signals[a]. [a] Even in a test bench it is important for the designer to keep track of the types of signals used.

1.6 TCL Script Automation or "How to Save Days, Minutes at a Time"

The goal in writing Tool Control Language (TCL) scripts is to automate as many steps as possible. The scripts are critical design artifacts that must be in the revision control system. The following paragraphs describe some of the benefits.

Portability among computers: A synthesis script can set the multitude of "behind the scenes" switches and buttons in the synthesis tool to ensure a design will simulate or synthesize independent of the host computer[8].

Remembering all steps: The script necessarily has all simulation or synthesis steps. The script captures knowledge of the set-up. This ensures that the engineer, now or several years from now, does not spend time[9] debugging a nonexistent problem due to a missing file type of problem. This can significantly ease maintenance in months (or years) when you don't remember how to put the project together.

Automate the version update: A script can automatically insert the host computer date and time into the FPGA source to provide a consistent version register. This avoids the annoying question "Which bit stream is in the FPGA?".

[8] This is the "It only works if synthesized on that particular computer." class of problem.
[9] Read: labor dollars

Speed: Running a script is generally much faster than entering (or clicking) commands. This saves a few minutes every hour.

Listing 1.1: A partial example of a simulation script

```
# File header...

echo "Starting the simulation..."
vsim -t 10ps ${LIB_TB}.tb_fpga_example

# Set up the simulation window
configure wave -namecolwidth 250
configure wave -valuecolwidth 70
configure wave -justifyvalue left

add wave tb_reset_n
add wave tb_clk

add wave -expand -group {LocalLink Signals}
add wave -group {LocalLink Signals} -radix hex temac_data
add wave -group {LocalLink Signals}            temac_src_rdy_n
add wave -group {LocalLink Signals}            temac_sof_n
add wave -group {LocalLink Signals}            temac_eof_n
add wave -group {LocalLink Signals} -radix hex temac_fifo_status

run $RUN_TIME
wave zoomrange {299 us} {310 us}

echo "%%TCL $SIM_SCRIPT_NAME: Complete."
```

2 Design Techniques and Common Errors

FPGA development consists of digital design, design capture, and elements of Printed Circuit Board (PCB) design. The following paragraphs address some of the best practices and common errors in these areas.

2.1 PCB Design

Device Configuration. Determine the FPGA configuration source and method of Joint Test Action Group (JTAG) access. It is permissible to have multiple devices in a JTAG chain[10]. Ensure access to the configuration mode control pins.

Clock input pins. Dedicated clock input pins have easy access to internal global clock routing resources.

Vendor IP pins. Build vendor IP prior to PCB layout. Complex IP such as a DRAM controller has very specific timing and PCB layout constraints. A "draft" synthesis allows you to identify specific input/output (IO) constraints. The PCB designer uses the tool generated pin assignments and makes as few changes as possible. Any changes are fed back to the synthesis tool to verify acceptability. Do this prior to PCB fabrication.

Data flow. Consider the data flow through the FPGA. Some FPGAs architectures are more efficient with data flow in a certain direction such as from left to right with carry's moving from top to bottom. Assigning pins to work with the physical architecture increases resource usage efficiency.

Electrical standards. While FPGAs have multiple IO banks, there are limits on which IO standards can be mixed within an individual bank. Trying to mix incompatible IO standards in an FPGA bank is a common error and can be quite painful to discover after the PCB layout is complete.

[10] Use extra caution when devices in the chain are not from the same family or manufacturer.

Simultaneous Switching Output (SSO) Each IO bank is limited in the amount of current it can draw. If too many outputs change simultaneously, the bank may need more current than is available. A good presentation on this subject is Simultaneous Switching Output (SSO) Analysis Using Xilinx Virtex-4 FPGAs.

Power Distribution Read and heed manufacturer recommendations regarding power supply voltages, sequencing, and decoupling.

Debug Support A test connector, as large as the the largest bus in the design plus a few control signals, can provide incredible insight during integration.

2.2 Digital Design

Some engineers seem to believe that HDLs let them write software that magically becomes a digital design in an FPGA. It does not. Digital design techniques are a critical skill set in developing FPGAs. Poor digital designs lead to poor products regardless of which tools or languages are used.

Hierarchy Separate the design into units that are small enough to label with a name that describes *the one thing it does.*

Component design. A common error is ineffective component or interface definition. It takes much more effort to rework something than it does to think it through prior to capture with HDL.

Clock Domain Crossing (CDC). Each design unit, or component, should have one clock (if the design unit has more than one clock, it is too big). Moving data and/or signals between design units requires special attention to CDC (see [14] or Doulos: A counter for fast events, using a Flancter).

Synchronous design. FPGAs require synchronous design techniques. All FPGA manufacturers recommend synchronous design. If your design is not synchronous, the manufacturer's support staff will have difficulty assisting you. Synchronous designs eliminate problems associated with speed variations through different paths of logic. By sampling signals at well-defined time intervals, fast paths and slow paths can be handled in a simple manner. They work well across variations of process, voltage, and temperature (PVT) and are easy to migrate to a new technology. In *Introduction to CPLD and FPGA Design*, Bob Zeidman describes a concise set of rules for synchronous design.

1. Synchronous design simply means that all data are passed through combinatorial logic and flip-flops that are synchronized to a single clock.
2. Delay is always controlled by flip-flops, not combinatorial logic.
3. No signal that is generated by combinatorial logic can be fed back to the same group of combinatorial logic without first going through a synchronizing flip-flop.
4. Clocks cannot be gated - in other words, clocks must go directly to the clock inputs of the flip-flops without going through any combinatorial logic.
5. One clock for the system. Do not clock entities or processes with outputs of other entities or processes.

Synchronous design counter example: ripple counter A ripple counter uses the data output of one circuit as the clock input to another circuit. This violates rule 5; *Do not clock a circuit with the output of another circuit.* This non-synchronous design uses less logic to count and was "OK" when clock speeds were slow, but can be disastrous with high clock speeds. Consider that with $F_{clk} = 100$MHz ($T_{clk} = 10$ns) and $T_{pd} = 1$ns, a 12 stage ripple counter will have bits changing in multiple clock cycles! The resultant "glitchy" output is deterministic.

Reduce uncertainty while coding. Avoid this as it leads to latent defects. You should have a clear vision of the functionality you want before you start to capture the design in HDL.

2.3 VHDL Capture

Inferred Latch. An inferred latch warning[11] is the tool telling you that the designer did not effectively communicate their intention. To complete the equation for the described behavior, the tool had to *add a memory element the designer did not request!* Avoid this by assigning default values for signals in processes and including `else` clauses in `if` statements.

Sensitivity List The authors coding guidelines specify processes for clocked logic elements and *concurrent statements* for combinatorial logic[12]. If a process sensitivity list has more than `clock` and maybe an asynchronous `reset`, I will start looking for errors there.

Variables Synthesizable HDL is not software. Variables can be powerful tool, but they can lure the unwary into writing software instead of describing a digital design. Where there are variables, look for errors.

3 Simulation

Simulation requires a bit more time up front, but the return on investment is excellent. The primary benefit is that you know the FPGA functionality is correct prior to lab integration.

3.1 Why Simulate?

The functional simulation proves that the design, described by the HDL, meets the requirements. Verifying functionality in hardware in the lab is very labor intensive. Because of the limited visibility into the FPGA, it is very difficult to debug. During a simulation you have nearly unlimited visibility into the design. Finding a design or coding error is much faster during simulation than during lab integration.

System Models. When simulating a digital design that communicates with a part external to the FPGA, the simulation requires a model of the external part. When developing these system models, the philosophy revolves around *reasonable fidelity*. If you are simply testing an SPI interface, a very simple model will suffice.

Timing. The functional simulation tells us the first part of the story. Static timing analysis from the synthesis tool shows that the design will or will not run at the required clock speed. These tools require clock related constraint definitions.

How much simulation? When using "hard IP" such as an Ethernet MAC in a Xilinx Virtex-5 device, it makes little sense to run simulations involving the MAC IP. That simulation requires significant time and you cannot modify the IP to correct any deficiencies[13].

3.2 Simulation Tips

VHDL `wait` amplified. The VHDL `wait` statement can be used to detect if a condition is met within a specified time period as shown in Listing 3.1. It can stop the simulation with a reference to the file and process where the violation occurred.

Listing 3.1: VHDL "wait" amplified

```
-- Wait until the downstream device is ready
wait on clock until (clock = '1') and (ready_in = '1') for (20 * CLK_PERIOD) ;
if (ready_in /= '1') then
  report "FATAL file_name : process_name " & "No ready_in"
    severity failure;
end if;
```

[11] This warning should be an error that reboots your computer!
[12] FSMs are an exception to this rule.
[13] Furthermore, it has been extensively tested by the manufacturer and many, many designs.

ModelSim SignalSpy. You can use `init_signal_spy` in ModelSim to "attach" a signal in the test bench to a signal at an arbitrary point in the design hierarchy *without adding temporary entity ports.* This is illustrated in Listing 3.2.

Listing 3.2: ModelSim Tips

```
signal spy_state_reg  : state_type;
signal spy_we_to_uart : std_logic;

begin

-- init_signal_spy(<src_object>, <dest_object>, <verbose>, <control_state>)
init_signal_spy ("/dut/inst_dut_fsm/state_reg", "spy_state_reg");
init_signal_spy ("spy_we_to_uart", "/dut/we_to_uart");
```

3.3 OS-VVM

The Open Source - VHDL Verification Methodology (OS-VVM^{TM})[14] provides constrained random testing and functional coverage at very little cost of entry.

> OS-VVM delivers advanced verification test methodologies, including Constrained and Coverage-driven Randomization, as well as Functional Coverage, providing advanced features to VHDL design engineers while enabling them to continue to develop using VHDL.

Additional information can be found at Business Wire OS-VVM Article, SynthWorks Downloads, and Aldec and SynthWorks OS-VVM webinar.

4 FPGA Project Management

Engineering is business, and business is about relationships. Therefore, successful engineering is about building quality relationships. Projects don't usually fail for technical reasons. They fail because of how the people on the development team behave.

Many engineers claim they don't care for marketing, even though that is what keeps them employed. Profit is a requirement of continued business activity and is strongly related to marketing. The strongest marketing of a company is high quality, well documented (read: maintainable) products[15]. All engineers should strive toward this end. This is how you build a reputation for your company, your group, yourself. Your reputation strongly influences your next job; or lack thereof.

Production of documentation with sentence fragments. Misspelllings also give cause to question your professionalism. *This is a competitive world; do not provide reason to question your attention to detail.*

4.1 Managing Engineers

If you went to the trouble of finding, interviewing, and hiring talented engineers, it behooves you to provide an efficiency enhancing environment.

Productivity In PeopleWare[9], DeMarco and Lister determined that long periods of uninterrupted activity greatly increases productivity. After a day of constant interruptions (meetings, emails, phone calls, etc.) one can leave having accomplished nothing. This damages the schedule and budget, but more importantly it damages morale.

[14] The authors have no financial relationship with SynthWorks or Aldec.
[15] Granted, any documentation is preferable to none, provided it is not incorrect.

Training. Engineering school provides the scientific and mathematical basics, but really prepares us to learn *on the job*. When presenting an engineer with a type of problem with which he is unfamiliar, instructor led training can significantly shorten the learning curve (read: shorten the schedule and lower the budget).

Teams. Team cohesion and individual motivation are keys to project success; equal to or greater than technical challenges. Team cohesion across all disciplines is essential. *If the project fails, it will not matter how perfectly the FPGA was designed*[16].

Company value. *The product development engineers are as valuable to the future success of the company as the products are to its present success*[17]. Treating engineers, highly educated technical professionals, as commodities will drive down team morale and inevitably affect schedule and quality. Under bidding a project to win the business, then expecting or demanding engineers work unpaid overtime to make up the difference is not a path to long term success of a company.

Job satisfaction. Give an engineer an interesting problem to solve, the resources to solve it, and as few interruptions and annoyances as possible and their job satisfaction will be high. Fortunately, nurturing an engineer's self-worth is relatively easy. A pair of large monitors costs a few hundred dollars. Considering the efficiency improvement due to information availability and the likely morale and motivation boost, the return on investment is quite good.

Schedules. A schedule estimate is a *best guess* regarding when events will occur. The farther in the future, the less the accuracy. You must periodically revise a long term schedule. Demanding significant personal sacrifice of employees to meet an arbitrary schedule is a recipe for long term corporate disaster.

4.2 HDLs: Software or Hardware?

While FPGA development (digital design) is not software development, the process and managerial tools are quite similar. The two tasks are essentially creating incredibly complex logic structures.

Listing 4.1: VHDL sample

```
p_igq_reg : process(sysclk)
begin
  if(rising_edge(sysclk))then
    if(reset = '1')then
      igq_reg <= '0';
    else
      igq_reg <= igg_in;
  end if;
end process;
```

One definition of software is anything that can change after manufacture is complete. Because the FPGA design can be changed after Circuit Card Assembly (CCA) manufacture is complete, it is sometimes mistakenly called software. However, VHDL and Verilog are not programming languages. HDLs do not execute anywhere. They do not fit what we think of when we use the term "software".

What is VHDL? VHDL is a text based language used to describe a digital design. HDLs allow designers to describe the functionality of the hardware without actually designing the hardware. They allow designers to separate behavior from implementation at various levels of abstraction. They can describe a logic gate exactly or describe the behavior of a counter.

HDLs: Software or Hardware? This is a misdirected question. VHDL can be used to describe hardware or write software. A test bench may contain file access routines and a text parser. These are software functions written in VHDL. The HDL that describes the digital design is just that; *digital circuit design*, a

[16] Charley's Maxim
[17] Charley's Axiom

hardware skill set. The key point is that *FPGA design is digital circuit design.* Writing reliable, synthesizable, efficient, synchronous VHDL for FPGAs requires knowledge of digital design techniques. In HDL form or schematic form, digital design is still digital design.

Software and Hardware Differences Software is a list of instructions that are executed by one or more processors. Source code is compiled into object code, but it is still a list of instructions. Digital design is the placement of logic circuits on an integrated circuit. VHDL is synthesized into logic circuits that are then placed in the FPGA and connected together with routing resources. The VHDL never "executes". The relative location of circuits is critical to performance.

Software and Hardware Similarities The digital design in an FPGA can change many times per day This adds incredible flexibility, but at the cost of significant potential for confusion. This implies a strong need for revision control.

Key point: HDLs and simulators make it easy to substitute action for thought. Avoid this temptation; design first, then code!

Why is the "hardware or software" question important? For describing digital designs you do not have carte blanche. It is important to capture a digital design using standard syntax preferred recognized by the synthesis tool. Always keep in mind the question: What will a synthesis tool build from this code?

5 FPGA Design Resources

Online training. Major FPGA manufacturers, Microsemi (Actel), Altera, and Xilinx, have significant *free* online training. Several industry media companies also provide *free* online training. Some examples are Design News Curriculum Calendar, EE Times Online Courses, and EE Times / Analog Devices Webinar: Fundamentals of Clocks and Clocking.

Archived answers. Many FPGA development questions have previously been answered. Major manufacturers archive these answers to save time. Two of these resources are the Altera Wiki and Xilinx Answer Records.

High reliability. Many lessons learned in high reliability fields are directly applicable to increasing the quality of typical FPGA designs. High reliability resources include COSMIAC and MAPLD. Additional recommended papers are Oak Ridge National Laboratory (ORNL) *Survey of Field Programmable Gate Array Design Guides and Experience Relevant to Nuclear Power Plant Applications*[5], European Space Agency (ESA) *Lessons Learned from FPGA Developments*[10], and National Aeronautics and Space Administration (NASA) *FPGA Insertion Guideline*[13].

HDL Language resources. Ashenden's book, *The Designer's Guide to VHDL*[4] is an excellent reference. A good Verilog reference is *Verilog HDL: Digital Design and Modeling*[6]. Among the myriad online resources are UMBC Compact Summary of VHDL and University of Pennsylvania VHDL Tutorial.

FPGA Digital design. FPGA Manufacturers provide a wealth of information. The following are good starting points.

- Actel: *Actel HDL Coding Style Guide*, `www.actel.com/documents/hdlcode_ug.pdf`
- Altera: *Advanced Synthesis Cookbook*
- Altera: *Recommended HDL Coding Styles*
- Xilinx: *HDL Coding Practices to Accelerate Design Performance*
- Xilinx: *UG625 Constraints Guide (Oct. 2011)*

A new book, *The Art of Hardware Architecture*[3] was reviewed in EE Times and appears to be a very good treatment of the subject[18].

[18] The authors have not yet read this book.

DSP Resources. The Andraka Consulting Group has published several excellent papers including *A survey of CORDIC algorithms for FPGA based computers*[2]. Richard Lyons' books, *Understanding Digital Signal Processing*[11] and *Streamlining digital signal processing: a tricks of the trade guidebook*[12] are excellent. Lionel Cordesses wrote exellent articles on Direct Digital Synthesis (DDS)[7, 8].

Definition of Acronyms

ASIC	Application Specific Integrated Circuit
CCA	Circuit Card Assembly
CDC	Clock Domain Crossing
COSMIAC	Configurable Space Microsystems Innovations & Applications Center
CPLD	Complex programmable logic device
DDS	Direct Digital Synthesis
DRAM	Dynamic RAM
ESA	European Space Agency
FAA	Federal Aviation Administration
FAE	Field Applications Engineer
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IO	input/output
IP	Intellectual Property
JTAG	Joint Test Action Group
MAC	Medium Access Control
MAPLD	Military and Aerospace Programmable Logic Devices
NASA	National Aeronautics and Space Administration
ORNL	Oak Ridge National Laboratory
PAR	Place and Route
PCB	Printed Circuit Board
RTCA	Radio Technical Commission for Aeronautics
SOC	System on a Chip
SPI	Serial Peripheral Interface
SSO	Simultaneous Switching Output
TCL	Tool Control Language
VHDL	VHSIC Hardware Description Language

References

[1] Altera. *Internal Memory (RAM and ROM) User Guide*. Altera UG-01068-4.0. 2012.

[2] R. Andraka. *A survey of CORDIC algorithms for FPGA based computers*. Technical report. ACM, 1998.

[3] M. Arora. *The Art of Hardware Architecture*. Springer, 2011.

[4] P. J. Ashenden. *The Designer's Guide to VHDL*. 3nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[5] M. Bobrek et al. *Survey of Field Programmable Gate Array Design Guides and Experience Relevant to Nuclear Power Plant Applications*. Technical report. 2007.

[6] J. Cavanagh. *Verilog HDL: Digital Design and Modeling*. CRC Press, 2007.

[7] L. Cordesses. *Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 1)*. `http://lionel.cordesses.free.fr/gpages/electronics.html`. 2004.

[8] L. Cordesses. *Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 2)*. `http://lionel.cordesses.free.fr/gpages/electronics.html`. 2004.

[9] T. DeMarco and T. Lister. *Peopleware - Productive Projects and Teams*. 1999.

[10] S. Habinc. *Lessons Learned from FPGA Developments*. Technical report. 2002.

[11] R. Lyons. *Understanding Digital Signal Processing*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2004. ISBN: 0201634678.

[12] R. G. Lyons. *Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook*. Wiley-IEEE Press, 2007. ISBN: 0470131578.

[13] D. Sheldon. *FPGA Insertion Guidelines*. `http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/40824/1/08-20.pdf`. 2008.

[14] R. Weinstein. *Application Note: The "Flancter"*. 2000.

[15] Xilinx. *Synthesis and Simulation Design Guide*. Xilinx UG626.

Index