# 1

# Field Programmable Gate Arrays

## 1.1 Overview

Field programmable gate arrays (FPGAs) were invented in the 1980s at Xilinx® as programmable array logic (PAL) devices [1, 2] and underwent three distinct stages of evolution each lasting a span of approximately 10 years [3]: Age of Invention, Age of Expansion, and the Age of Accumulation. Over this 30 year time-span, FPGAs have increased in logic capacity by a factor of over 10 000 and increased in speed by over 100; over the same period, their cost and power consumption per operation decreased by a factor of 1000. These impressive developments in processor technology led to their expansion into myriad applications [4]. The fine-grained resolution of logic resources and memory on the FPGA allows reconfiguring the hardware for a specific user application.

Reconfiguration was made possible due to static random-access memory (SRAM) based FPGAs, which can be reprogrammed using a configuration bitstream. The traditional FPGA design flow creates a register transfer level (RTL) abstraction using a hardware description language (HDL). The HDL code is translated by a synthesis tool into netlist files that are textual representation of the logic design. Then place-and-route tool is used to map the netlists to a specific FPGA device, and bit-generation tool creates a configuration bitstream. At each stage of the design, simulation can be used to verify the accuracy, timing, and functionality. While users can emulate any digital logic design on the FPGA, practical constraints such as the amount of hardware resources consumed, clock speed, and inputs/outputs (I/Os) need to be observed to achieve a successful emulation.

Although many-core graphics processing units (GPUs) may be more attractive for large-scale applications where floating-point computations predominate, algorithms implemented on GPUs must confirm to the single instruction multiple data (SIMD) paradigm to achieve the most acceleration. In comparison with GPUs, FPGAs have much lower latencies and higher computational density per watt

*Real-Time Electromagnetic Transient Simulation of AC-DC Networks,* First Edition.
Venkata Dinavahi and Ning Lin.

which is becoming increasingly significant as energy and cooling expenditures form the major portion of operating cost of high-performance compute (HPC) facilities. Currently available FPGAs contain millions of programmable logic gates, high-bandwidth memory, dedicated multiplier blocks, gigabit per second communication transceivers, and on-chip processors, making the platform very flexible and extensible for any specific application. FPGA vendors provide families of devices of various speeds, sizes, and functionality, each employing slightly different proprietary technologies, to target groups of application domains. In addition to hardware, vendors provide highly integrated computer-aided design (CAD) tools such as Vivado® from Xilinx® and Quartus Prime® from Intel®.

The future of the FPGA looks bright, and industry trends point to on-chip heterogeneous integration of programmable logic with multi-core CPU and many-core GPU resources to create multi-processing system-on-chip (MPSoC) [5] and adaptive computing acceleration platform (ACAP) [6] architectures to address the needs of evolving and diverse applications. FPGAs are currently being used in a wide range of application domains including consumer electronics, medical devices, robotics, industrial instrumentation and controllers, wireless communications, automotive electronics, and aerospace and defense equipment. Major cloud service providers are now integrating FPGAs [7] into their compute servers that are used for big data analytics. This chapter briefly introduces some of the terms and definitions related to FPGA architecture and design flow for hardware-in-the-loop (HIL) emulation.

### 1.1.1 FPGA Hardware Architecture

The FPGA is an integrated circuit containing an array of 2-D configurable logic blocks (CLBs) which are interconnected through wires and programmable switch matrices. A fundamental CLB is able to implement both combinational and sequential logic functions, and the programmable switch matrices also help to achieve hardware reconfigurability. Two typical FPGA hardware architectures are given in Figure 1.1 [8–10], which shows the I/O blocks connecting the CLBs and programmable switch matrices are arranged at the periphery of the logic array. The column-based advanced silicon modular block (ASMBL) architecture created by Xilinx® offers users a greater convenience in choosing an FPGA device with proper features for their design. This structure is adopted for the 7-series and Ultrascale+® FPGA devices. According to Figure 1.1b, the FPGA is composed of 6-input look-up tables (LUTs) based CLBs, on-chip block memory, digital signal processing (DSP) slices, precise clocking resources, enhanced PCIe® interface blocks, and the programmable switches interconnected via wires.

The Xilinx® Virtex®-7 FPGA VC707 Evaluation Kit [10] and the Xilinx® Virtex® Ultrascale+ FPGA VCU118 Evaluation Kit [11] are the two boards used in HIL

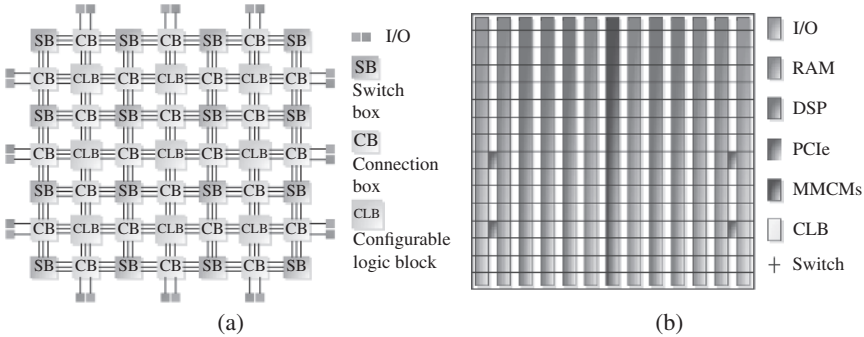


**Figure 1.1** FPGA hardware: (a) mesh architecture and (b) Virtex®-7 ASMBL architecture. Source: Kilts [8], Farooq et al. [9]

emulation in many of the chapters of this book. The Virtex®-7 XC7VX485T FPGA was manufactured with 28 nm process technology, while the Virtex® Ultrascale+ XCVU9P FPGA was manufactured on 16 nm 3D FinFET process technology. A comparison of the main logic resources on the XC7VX485T and the XCVU9P devices is presented in Table 1.2 [12, 13].

While the two generations of FPGA boards share many types of resources, the latest UltraScale+ XCVU9P FPGA is more resource-abundant and efficient in data exchange than the Virtex®-7 FPGAs. The availability of some of the resources is significant as it affects the scale of the emulated system for HIL application. Thus, they are introduced in the following subsections.

### 1.1.2 Configurable Logic Block

The CLB is the fundamental component in the FPGA for providing basic logic and arithmetic functions as well as data storage. In the Xilinx® Virtex®-7 series FPGAs, the CLB contains two side-by-side slices, each of which is composed of four 6-input LUTs, which has two flip-flops [14, 15]. In addition, a CLB also has three wide-function multiplexers and the carry chain to perform arithmetic adding and subtracting operands in its slices.

Figure 1.2 shows the architecture of a CLB. The slices, organized as two individual columns, are not directly connected to each other. *Slice0* is at the bottom of the CLB and place in the left column, while *Slice1* locates at the top and in the right column of the die.

There are two types of CLB slices: those support data storage using distributed RAM and data shift with 32-bit registers are categorized as SLICEM, while the rest are named as SLICEL. Then, a CLB can contain either two SLICEL or one SLICEL and SLICEM. The LUT in the Virtex®-7 FPGA can be implemented as one 6-input 1-output LUT for 64-bit ROM or two 5-input LUTs with individual outputs

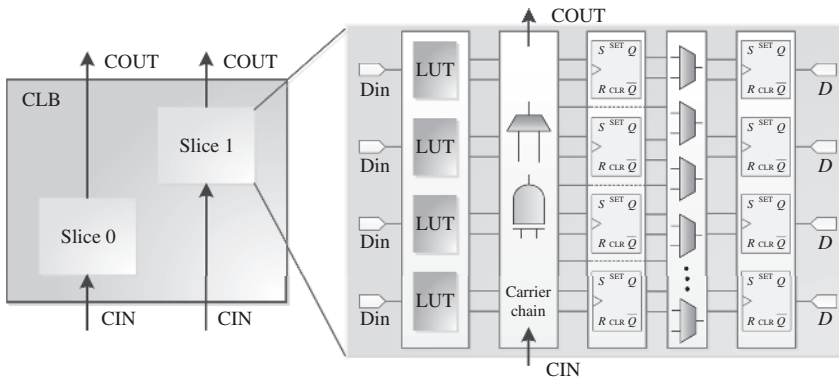


**Figure 1.2** Configurable logic block architecture.

for 32-bit ROMs. The carrier chain contains multiplexers and an XOR logic gate for the addition or subtraction operation. As can be seen, the inputs and outputs of a slice are also its ports.

### 1.1.3 Block RAM

In Xilinx® Virtex®-7 series and Ultrascale+ FPGAs, the block RAM (BRAM) has up to 36 kbits data storage capability, and it can be implemented as either 1 RAM or 2 separate RAMs with each having 18 kbits data [16, 17]. In addition, it also has the cascaded manner when an adjacent 36 kbits BRAM is implemented, i.e. $1 \times 64$ kbits, and under simple dual-port mode, there are a variety of configurations, e.g. $1 \times 32$ kbits, $2 \times 16$ kbits, or even $72 \times 512$bits. Similar configurations are also available to the two separated 18 kbits RAMs.

Under simple dual-port mode, there is only one read-only port and write-only port, which has a high degree of independence, e.g. they are controlled by two clocks, and the data width can also be different, and independent read/write actions can take place simultaneously. Correspondingly, another BRAM type is the true dual-port RAM, whose symmetrical configuration is given in Figure 1.3. It ensures a flexible data access to either or both ports by enabling them to have an individual address, input/output data, a clock signal, write enable, etc. The description of those port names is provided in Table 1.1.

### 1.1.4 Digital Signal Processing Slice

Programmable logic devices are efficient carriers for DSP applications, which use many binary multipliers and accumulators. Both the 7-series and UltraScale+ FPGAs have a number of dedicated low-power DSP slices, integrating high

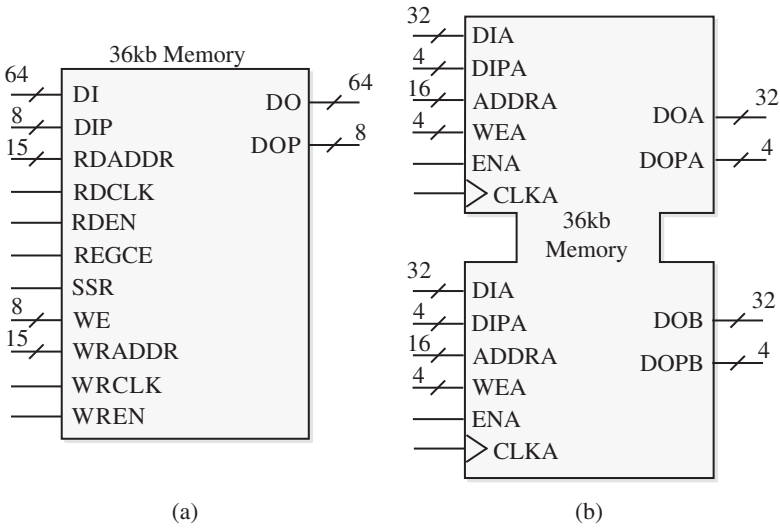


**Figure 1.3** Virtex®-7 series FPGA block memory: (a) simple dual-port RAM and (b) true dual-port RAM.

**Table 1.1** Dual-port RAM description.

| Port | Direction | Description |
|---|---|---|
| DI | In | Data input bus |
| DIP | In | Data input parity bus |
| ADDR | In | Address bus |
| WE | In | Byte-wide write enable |
| EN | In | BRAM write enable |
| CLK | In | Clock input |
| DO | Out | Data output bus |
| DOP | Out | Data output parity bus |

speed with compact size while at the same time the system design flexibility is maintained. In addition to DSP, the DSP slices also enable wide dynamic bus shifters, memory address generators, memory-mapped I/O registers, etc. On the 7-series FPGA boards, DSP48E1 slice is adopted [18], as shown in Figure 1.4, while its UltraScale+ counterpart is defined using more advanced DSP48E2 [19]. As shown in its slice architecture, the DSP48E1 slice includes $25 \times 18$ two's-complement multiplier, a 48-bit accumulator, 25-bit power-saving pre-adder, a pattern detector, etc.

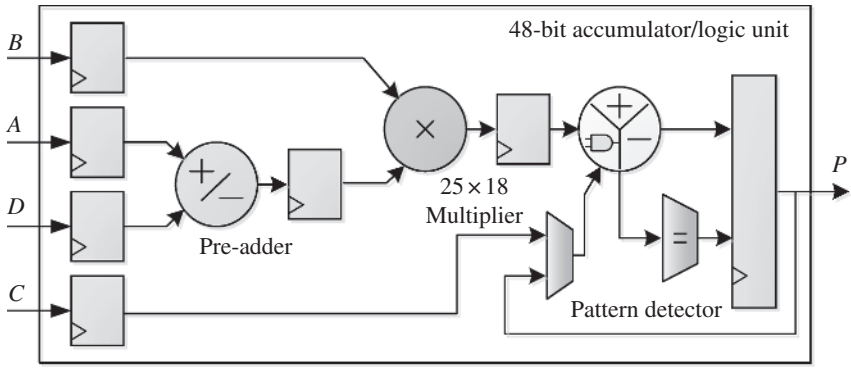


**Figure 1.4** Basic DSP48E1 slice functionality.

## 1.2 Multiprocessing System-on-Chip Architecture

Conventionally, an MPSoC is a device that contains multiple processing elements (microprocessors) that are heterogenous in nature and address the computing needs of an application area. In the context of a MPSoC from Xilinx®, the device consists of a multi-core CPU, many-core GPU, programmable logic (FPGA), memory, and I/O peripherals. As shown in Figure 1.5, these hardware resources are divided into two processing domains: processing system (PS) and processing logic (PL). On the Xilinx® UltraScale+ XCZU9EG MPSoC, the PS consists of ARM® Cortex-A53 quad-core application processing units (APUs), ARM® Cortex-R5 dual-core real-time processing units (RPUs), the ARM® Mali-400 MP2 GPU, memory, and high-speed connectivity, whereas the PL consists of system logic resources, memory, high-speed, and general purpose I/O [5, 21]. Within each APU, various components are available to accelerate computation, such as NEON (an advanced SIMD architecture), floating point unit (FPU), etc.

The PS communicates with PL using high bandwidth and low-latency Advanced eXtensible Interface (AXI) channels. Such architecture provides high flexibility to merge the advantages of the fast sequential calculation and the hardwired parallelism to meet the requirements of high-performance computing. However, the logic resources of MPSoC are typically lower than the FPGA device with the same manufacturing technology. Due to the complex routing and additional combinational and sequential logics inferred for reconfiguration on FPGA, the clock frequency of such a device is lower than that of the PS. It is critical to analyze the characteristics of the application before determining the implementation platform. The usage of FPGA resources can be low if the implemented functions are not frequently used. The resources can be a major limitation for the simulation of complex and large-scale systems on FPGA. A combination of sequential CPU

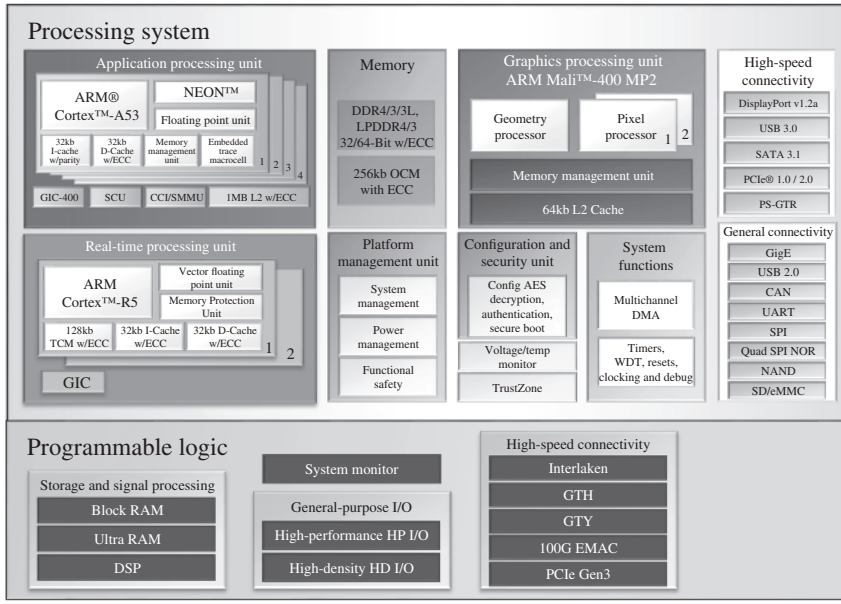


**Figure 1.5** Zynq® Ultrascale+™ MPSoC block diagram. Source: Xilinx [20]. ©2018, Xilinx, Inc.

and parallel hardware implementation on the MPSoC can be an efficient and cost-effective compromise in such cases. A summary of the main resources of the devices used in this work are shown in Table 1.2.

Xilinx® provides the Software Development Kit (SDK) tools for the software design of the MPSoC device. SDK imports the hardware design from Xilinx® Vivado® tool and provide the board support package which includes various fundamental drivers for the resources of both PS and PL.

## 1.3 Communication

Both Xilinx® VCU118 and ZCU102 boards have plenty of components and communication interfaces, such as double data rate fourth-generation (DDR4) memory, quad serial peripheral interface (QSPI) flash, general purpose IO (GPIO), FPGA mezzanine (FMC) interface, and small form-factor pluggable (SFP) interface. Utilizing these interfaces and communication protocols, data transfer can be accomplished between multiple boards. For example, the SFP interface of ZCU102 and quad-SFP (QSFP) interface of VCU118 can be interconnected with cable and the Xilinx® Aurora IP cores can be used to accomplish the communication between the two boards as shown in Figure 1.6a. The Aurora 64B/66B core is

**Table 1.2** Hardware resource comparison.

| Resource | Virtex®-7 XC7VX485T FPGA | UltraScale+ XCVU9P FPGA | Zynq UltraScale+ XCZU9EG MPSoC |
|---|---|---|---|
| Logic cells | 485 760 | 2 586 150 | 600 000 |
| CLB FFs | 607 200 | 2 364 480 | 548 160 |
| CLB LUT | 303 600 | 1 182 240 | 274 080 |
| Block RAM (kbits) | 37 080 | 75 900 | 32 100 |
| Clocking (CMTs) | 14 | 30 | 4 |
| DSP slices | 2 800 | 6 840 | 2 520 |
| PCIe® | 4 Gen2 | 6 Gen3 × 16/Gen4 × 8 | — |
| Transceivers | GTX (12.5 Gb/s) 56 | GTY (32.75 Gb/s) 120 | GTH (16.3 Gb/s) 24 |

Source: Xilinx [12].

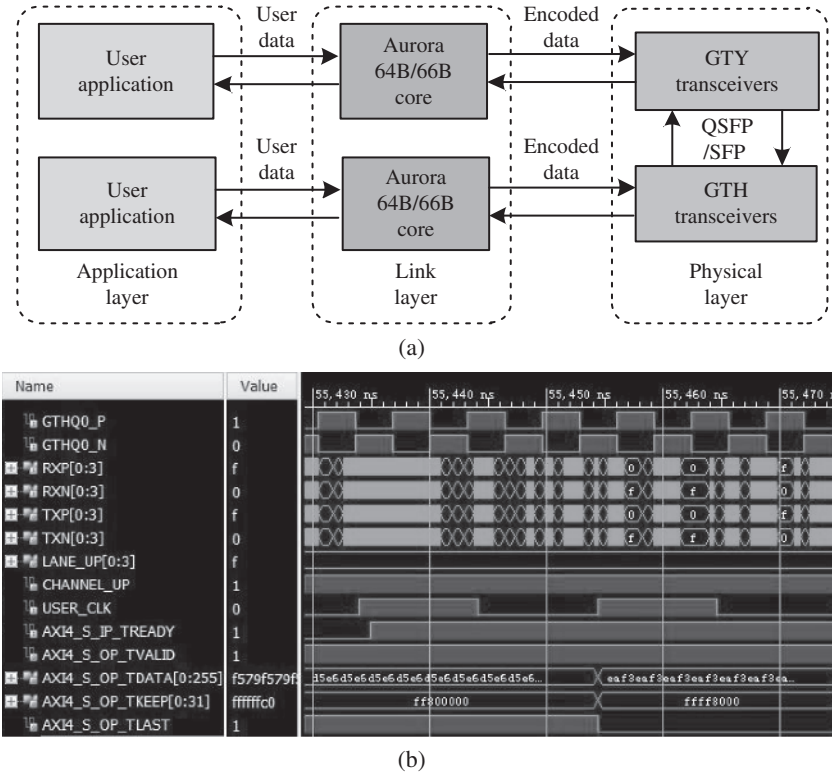


**Figure 1.6** Communication process: (a) block diagram and (b) digital signal waveforms. Source: Xilinx, Inc.

a scalable, lightweight, high-data rate, link-layer protocol for high-speed serial communication, supporting bi-directional transfer of data between devices using consecutive bonded gigabit transceiver Y (GTY) on VCU118 board and gigabit transceiver H (GTH) on ZCU102 board [22]. Four transceivers located in ZCU102 SFP and VCU118 QSFP interface can be used to construct four lanes with 64-bit AXI-4 user data stream transmitting in each lane, which can achieve a throughput from 500 Mb/s to over 254 Gb/s. Figure 1.6b shows the waveforms of major signals during the communication process. When CHANNEL_UP signal is asserted, the Aurora cores have initialized and established four channel lanes for user applications to pass frames of data. User data are loaded on AXI4_TDATA[0:255] bus (64 bits × 4 lanes) at each edge of USER_CLK when AXI_TREADY is asserted. Then user data are transferred into encoded differential serial data (RXP/N[0:3] and TXP/N[0:3]) and transmitted through the four GTH or GTY transceivers and the QSFP/SFP cable.

## 1.4 HIL Emulation

The entire hardware design procedure for HIL emulation is depicted in Figure 1.7. The whole process can be completed in three stages, summarized as follows:

- Data entry by high-level synthesis.
- Top-level design and simulation with Vivado®.
- Bit file generation and experimental test on FPGA.

Thus, in the following each stage is specified, including tools necessary for the design, the programming language, and prototype setting.

### 1.4.1 Vivado® High-Level Synthesis Tool

The Xilinx® high-level synthesis software Vivado HLS® is able to transform C/C++ functions into an RTL implementation which synthesizes into the vendor's FPGAs. During this stage of design, the user can develop a hardware module using the programming language C/C++, rather than VHDL at the logic gate level, which greatly facilitates the hardware design. For example, to realize a complex multiple-input-multiple-output algebraic module, the C/C++ function can be written as:

```
void func (float ai, float bi, float *ao, float *bo){
  algebraic functions here;
}
```

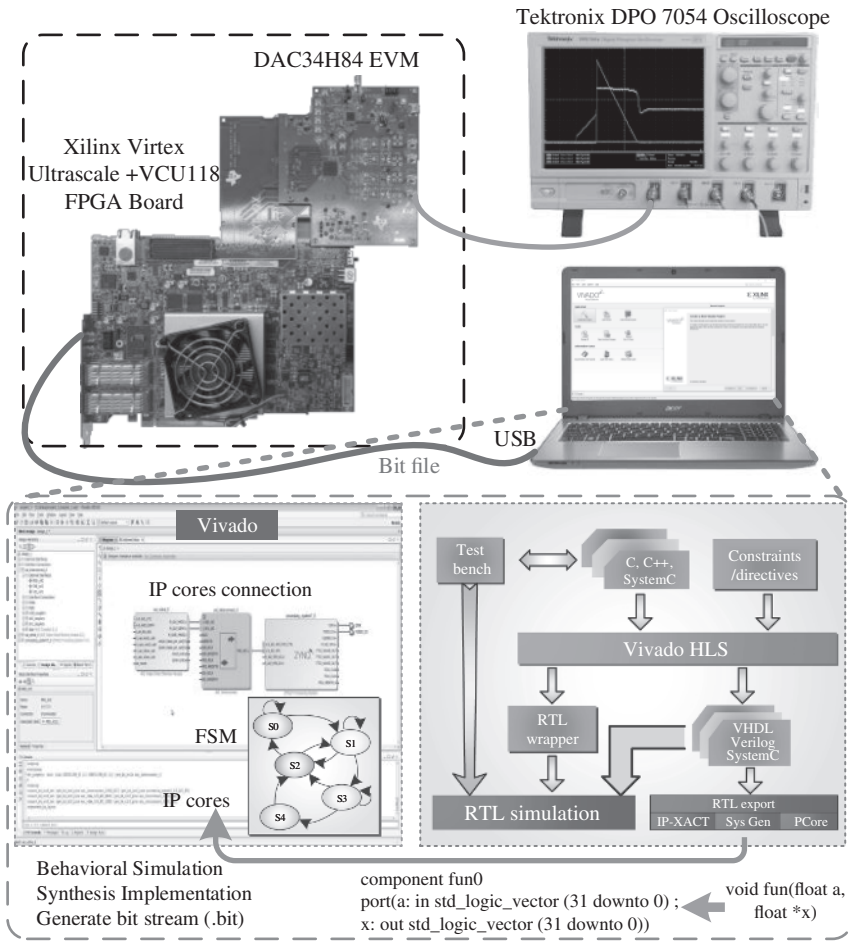


**Figure 1.7** Hardware design procedure and experimental setup. Source: Xilinx, Inc.

Note that the variables are defined as a floating point which corresponds to 32 bits because it is more efficient for computation than 64-bit digits. The algebraic function description could contain potentially parallel operations, and therefore the design tool offers pipeline structure option in its directives, which greatly facilitates programming. Then, the C synthesis function provided by that tool creates the RTL design of the written function automatically. The syntax is also checked during this process: an erroneous function would lead to immature termination of C synthesis. The option *Export RTL* enables the RTL design to be exported as an

intellectual property (IP), which has corresponding input/output ports in VHDL format:

```
COMPONENT func
  PORT (
  ao_ap_vld: OUT STD_LOGIC;
  bo_ap_vld: OUT STD_LOGIC;
  ap_clk: IN STD_LOGIC;
  ap_rst: IN STD_LOGIC;
  ap_start: IN STD_LOGIC;
  ap_done: OUT STD_LOGIC;
  ap_idle: OUT STD_LOGIC;
  ap_ready: OUT STD_LOGIC;
  ai: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
  bi: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
  ao: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
  bo: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
)
```

Since RTL design could be completed by Vivado HLS®, C/RTL co-simulation is available in the software after writing a corresponding test-bench. The co-simulation is deemed equivalent to the hardware behavioral simulation, and the results are a preliminary validation of the hardware design even though it is C-based.

### 1.4.2 Vivado® Top-Level Design

A user application or system model may contain a number of subsystems, normally classified according to their functionality. For HIL emulation, all those subsystems are first written as separate C/C++ functions under Vivado HLS® environment, and after IP generation and export they can be identified by Vivado® as VHDL components. Those user-defined IPs are in fact treated as the same as the default hardware modules in the IP catalog.

In a typical real-time application, signals between VHDL components are exchanged at the end of each time-step, e.g. a controller sends command signals to the modeled plant, which returns its sampled voltages and currents. In some cases, the outputs of a component should be fed into its input ports. All those data exchanges are not included in the hardware modules designed by C/C++ functions in HLS®. Instead, their connection is achieved in Vivado® using the programming language VHDL, and the typical syntax is identical for the two

scenarios. Take the above hardware module *func* for example, when the outputs need to be sent to its input ports, it can be written as

```
if clk='1' and clk'event then
if ao_ap_vld='1' then ai$<$=ao; end if;
if bo_ap_vld='1' then bi$<$=bo; end if;
end if;
```

which can be synthesized into two separate latches, and Figure 1.8 shows the self-connection of the hardware module *func*. Once a valid output is generated, the corresponding data valid signal *ao_ap_vld* or *bo_ap_vld* becomes binary 1, which is taken as the clock signal of the D latch. Therefore, the outputs *ao* and *bo* will be respectively fed to the input ports *ai* and *bi* of the same module when the clock signal *ap_clk* arrives.

The *ap_ctrl* port includes four binary ports, among which the start control port is controlled by the finite state machine (FSM) along with the reset port *ap_rst*. Thus, unnecessary calculation by the module which leads to incorrect results can be avoided with an appropriate operation sequence in the FSM, and in case a rerun of the emulation is needed for observation of particular power system phenomena, giving a reset order is sufficient. On the other hand, the other three signals indicating the operation status of the module are taken by the FSM as feedbacks for state shift judgment, e.g. the emulation remains at the current state till the module with the largest hardware latency completes calculation and set its binary signal *ap_done* to 1.

Since the above design is carried out manually while the RTL design for Vivado HLS® co-simulation is conducted automatically, the results from the artificial design are not guaranteed to be correct. Thus, the behavioral simulation offered by Vivado® is a further validation approach of the top-level hardware design.

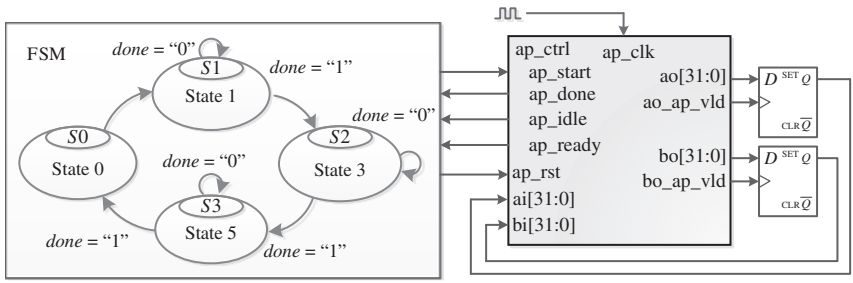


**Figure 1.8** Demonstration of top-level hardware design.

### 1.4.3 Number Representation and Operations

For computational programming, the first step is to define the data format and arithmetic operations. In spite of the dependency of data format and the accuracy of computation, number representation also affects the hardware resource utilization in the FPGA design and programming. Basically, there are two types of number systems:

1. **Fixed-point number:**
   A fixed-point number is characterized by their word size in bits, binary point, and their sign. A common representation of a binary fixed-point number, either signed or unsigned, is shown in Figure 1.9a.
   As can be seen, $a_i$s are the binary digits, $n$ is the word length in bits, $a_{n-1}$ is the most significant bit (MSB), and $a_0$ is the least significant bit (LSB), and the binary point is depicted three bits to the left of the LSB.
   It is worth mentioning that the preferred representation of signed fixed-point numbers is Two's complement method.
2. **Floating-point number:**
   According to *IEEE* standard 754, floating-point numbers may be represented in either single- or double-precision format.
   - *Single-precision floating-point number*:
     Any value stored in this representation has 32 bits. It is formatted as depicted in Figure 1.9b.
     A 32-bit floating-point number consists of 1 *sign* bits, 8 *exponent* bits, and 23 *fraction* bits. The single-precision floating-point number is expressed as follows:

$$d = (-1)^{sign} \times 2^{(exponent-127)} \times (1.fraction). \tag{1.1}$$

     The value in this format is approximately in the range of $-10^{38}$ to $10^{38}$.

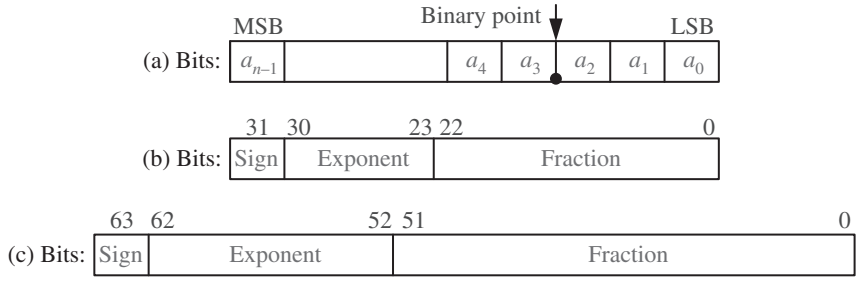


**Figure 1.9** (a) Floating-point number representation, (b) single-precision, and (c) double-precision.

- *Double-precision floating-point number*:
  Any value stored in this representation has 64 bits. It is formatted as depicted in Figure 1.9c.
  A 64-bit floating-point number consists of 1 *sign* bits, 11 *exponent* bits, and 52 *fraction* bits. The double-precision floating-point number is expressed as follows:

$$d = (-1)^{sign} \times 2^{(exponent-1023)} \times (1.fraction). \tag{1.2}$$

  The value in this format is approximately in the range of $-10^{308}$ to $10^{308}$.

In this thesis, single-precision floating-point is employed for computational programming of FPGA. It actually offers high-speed computation, dynamic range of data, and acceptable accuracy. These are the main reasons for choosing single-precision format for real-time emulation.

### 1.4.4 FPGA Design Schemes

Parallel and pipeline paradigms are two interesting features for FPGA programming to improve computational speed and throughput. In this section, these two FPGA design techniques will be explained.

#### 1.4.4.1 Pipeline Design Architecture

In the pipeline technique, a function is divided into the several stages, and the registers are inserted between the stages in the data path. Thus, data can march through the registers at every clock cycle. Although pipelining increases the number of clock cycles per operation, corresponding to the number of stages in a pipeline path, it improves the computational throughput by increasing the number of operations per unit of time and also conserves FPGA hardware resources. As can be seen from Figure 1.10, the pipeline path of FPGA design has three stages. The first valid results become available after three clock cycles, and the next results will be ready only after one clock cycle instead of waiting for another three clock cycles, consequently, the data throughput is one result per clock cycle.

#### 1.4.4.2 Parallel Design Architecture

Unlike CPUs or DSPs that are sequential computational engines, an FPGA enjoys its intrinsic parallel architecture for high-speed operations and computations. Basically, an algorithm can be partitioned into several independent circuits and computed simultaneously on an FPGA. As can be seen from Figure 1.10, the operations in the first and second paths are executed concurrently on FPGA, whereas these two operations must be performed sequentially on CPU.

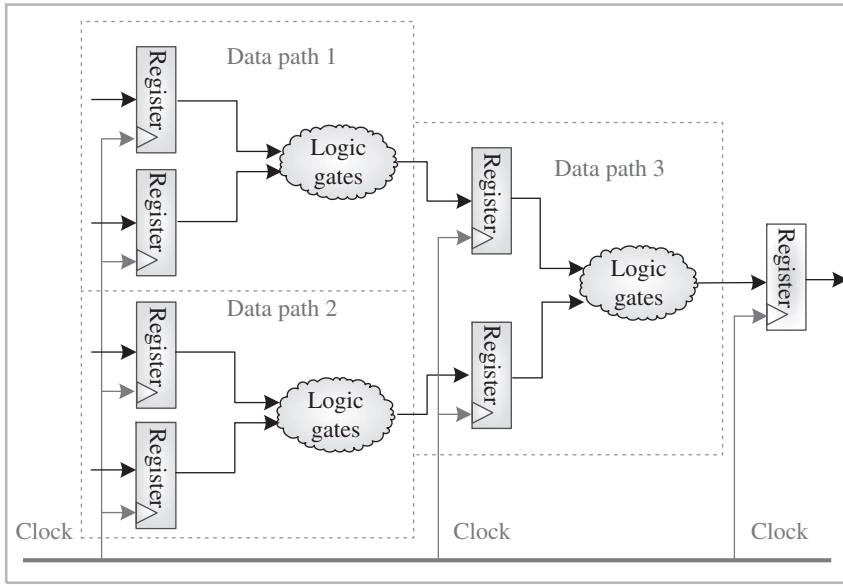


**Figure 1.10** Pipeline and parallel paths of FPGA design.

### 1.4.5 FPGA Experiment

The HIL emulation results are ultimately expected to be observed on the oscilloscope or interfaced with an external device under real-time HIL conditions. To achieve that goal, the designed top-level needs to be implemented on the FPGA after following steps in Vivado®:

- *Run synthesis*: This is a process of transforming an RTL design into a gate-level representation.
- *Run implementation*: This includes all necessary stages to place and route the netlist onto FPGA resources, under various logical, physical, and timing constraints.
- *Generate bit stream*: This implements the embedded design and creates a bit file that can be downloaded into the targeted FPGA board.

As shown in Figure 1.7, a digital-to-analog conversion medium is mandatory since the oscilloscope channels receive analog signals. The Texas Instruments® DAC34H84 quad-channel, 16-bit, digital-to-analog converter (DAC) with a sample rate as high as 1.25 GSPS is connected to the FPGA or MPSoC board and the Tektronix® DPO7054 oscilloscope so that the hardware design results can be displayed as real-time waveforms.

## 1.5 Summary

This chapter described the fundamental aspects of FPGA architecture, design flow, tools, and programming technologies. Due to its intrinsic massively parallel architecture, hardware pipelining computation and custom configuration FPGAs outperform the general purpose CPU for real-time applications. In addition, the FPGA manufacturers provide all the necessary environmental support design software, and well-designed fully-tested IP cores allow the designer to implement a FPGA-based digital hardware emulator efficiently. Nevertheless, future HPC technologies are increasingly moving toward MPSoC like co-processing wherein heterogeneous compute architectures are seamlessly integrated to achieve the most efficient software and hardware workload distribution.