

## Chapter 2

# FPGA and Digital Signal Processing

**Abstract** This chapter will introduce the essential information of field-programmable gate-array (FPGA) and FPGA-based digital signal processing at system level without getting into too much detailed hardware design and implementation issues. The contents of this chapter will cover the following three topics: the state-of-the-art FPGAs, FPGA-based DSP basics and FPGA-based DSP system design. We'd like provide a concise system level view and use this chapter as a “soft” appetizer prior the “hard” main dishes.

### 2.1 Introduction

Different than general purpose integrated circuit (IC), like microcontroller, an application specific integrated circuit (ASIC) is a type of IC that is customized for a particular application. Usually, we categorize the ASICs into three types: (1) full-custom ones, (2) standard cell-based or semi-custom ones and (3) programmable ones. Among the programmable ASICs, field programmable gate array (FPGA) is one of the most important and widely utilized. Simply speaking, FPGA chips are manufactured in a ready-to-use state, users can customize the FPGA chips with software-defined “programs” to achieve desired functions and re-program the chips later for carrying out other functions.

In 1985, Xilinx [1] co-founders Ross Freeman and Bernard Vonderschmitt invented the first commercial FPGA—XC2064, one year before, Altera [2] released the first industrial reprogrammable logic device—EP300. After thirty years growing, the top two FPGA vendors regarding market shares at the moment are Xilinx and Altera (an Intel company since end of 2015), but quite a few semiconductor companies have been very actively engaged in FPGA related activities and business, such as Lattice Semiconductor [3] and Microsemi [4] (was Actel).

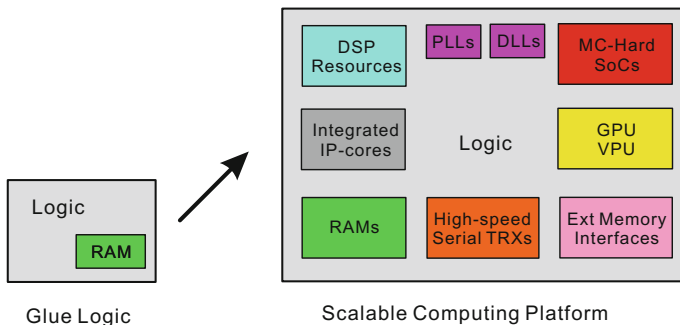
## 2.2 State-of-the-Art FPGAs

FPGAs were born in the 1980s when digital system design engineers were struggling with the continuously evolving requirements for larger digital system design tasks. After three decades, nowadays FPGAs have already become one of the most popular platforms for building digital systems in many applications across a large range of industries. Modern FPGAs are not only providing parallel operations but also capable of high speed processing and massive data throughput interfacing. It is quite amazing to see the role of FPGAs has been changing along the time.

### 2.2.1 From Glue Logic to Scalable Computing Platform

At the early stage, FPGAs were quite small regarding the available number of logics on the chip, resulting that they were mainly used as glue logic to attach two or more logical modules together. In other words, the early FPGAs were widely utilized as the interfaces towards a number of off-the-shelf ICs or self-contained electronic modules for building a larger-size and more complicated digital application. At that time, the main resources on the FPGAs were merely logic and memory. Thanks to the growth of semiconductor process technology in an evolution manner, the state-of-the-art FPGAs are fundamentally evolved in all dimensions, particularly the chip-architecture, logic density and on-chip hard-core functionalities.

Figure 2.1 illustrates the evolution of FPGA functionalities on a single chip from glue logic to the scalable computing platform with heterogeneous processing units including, hardware multipliers, soft-core CPU, ARM [5], graphic processing unit (GPU) [6], vision processing unit (VPU). Besides the those delicate processing units, quite a few other high-performance units can be equipped on a FPGA chip, such as high-speed serial transceivers (over 10 Gbps per lane), large on-chip random access memory (RAM), customizable external memory interfaces and many integrated interfaces, like SPI, USB, Ethernet, PCI-express and so on. Actually, we



**Fig. 2.1** FPGA evolution regarding functionalities on the chip

may give another name for this monster IC, like heterogeneous computing IC, since this chip architecture is far beyond the original FPGA concept.

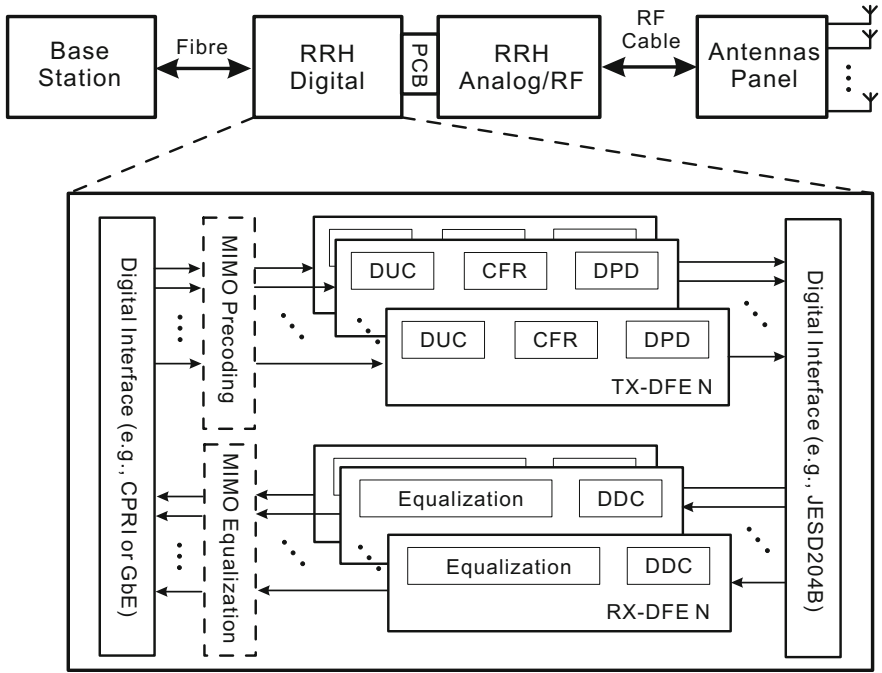
### ***2.2.2 From Control Logic to High-Speed Parallel Processing***

FPGAs are inherently parallel processing capable ICs. This amazing feature enables very robust multi-branch simultaneous control functions associated to the multiple real-time co-operated machines. Without any surprise, FPGAs are quite popular and have been widely used in the industry as central control units monitoring and adjusting multiple devices/machines simultaneously.

As the analog switching performance of transistor has been improved continuously, the state-of-the-art FPGA fabrics can be reliably running at over 500 MHz processing. Together with its nature parallel operation, FPGAs not only provide super control logics for the applications with scaled number of devices, such as millions of Internet of things (IoT) terminals, but also enable the high-speed parallel data transmission and processing applications. Because of the above features, FPGAs have been heavily utilized in the telecom industry, e.g., on the communications infrastructures. Let's have a look at the modern remote radio head (RRH) in the wireless network infrastructures.

RRHs have become one the most important network equipments for both centralized and distributed wireless network architectures. In the forthcoming 5th generation (5G) wireless network, RRHs will be used as very important equipments to flexibly extend the coverage of centralized radio access network (C-RAN) or cloud-based radio access network (Cloud-RAN) with virtualized base-station function. The basic function of a RRH is to up-convert the digital baseband signal to the analog RF signal at the transmitter and down-convert the received analog RF signal back to digital baseband signal at the receiver branch. At one end, the RRHs are usually connected to the baseband processing units via CPRI interface using fibre as the real transmission media. At the other end, RRHs are positioned quite close to the antenna panel, so that short RF cables (saving very valuable RF power) can be used to connect them on the top of buildings. An example of simplified functional diagram of a typical wireless RRH is illustrated in Fig. 2.2. A typical RRH usually contains digital part and analog/RF part and FPGAs are normally used as the host at the digital part.

On the modern RRHs, like the ones deployed in the 4G wireless network, there are a few necessary functions are running in the FPGA chips working as digital front end (DFE) [7] including digital up-conversion (DUC), crest factor reduction (CFR), digital predistortion (DPD) and digital down-conversion (DDC). And some of them may have MIMO precoding module and MIMO equalization module to support multiple transceivers operations. Thanks to the state-of-the-art FPGA technology, we can see multiple similar function modules can be achieved on a



**Fig. 2.2** Typical digital functions on the FPGA used on the RRH digital part

single chip for processing multiple data branches in parallel, saving quite a bit processing and interfaces power and reducing the size of print circuit board (PCB) and bill of materials (BOM) comparing to the multiple chips solution.

### 2.2.3 From VHDL Design to Dense IP-Cores Integration

The design philosophy of modern FPGA has changed. For simple glue logics, control functions or small-size digital applications, writing hardware description language (HDL), like VHDL, Verilog would be a quite efficient and powerful approach. However, as the chip design processing developing and the increasing number of transistors on the same silicon area, modern FPGAs are becoming monsters regarding the available number of logics and number of processing units. Building from scratch at register level won't be the optimal approach to do the FPGA design any more. Instead, pre-defined or customizable intellectual property (IP) core integration based design strategy will be more suitable for majority advanced DSP applications. IP-cores are reusable units of logic, cell or chip layout design provided by FPGA vendors, 3rd party companies or self-developed. By using IP-cores integration-based FPGA system design approach, we can

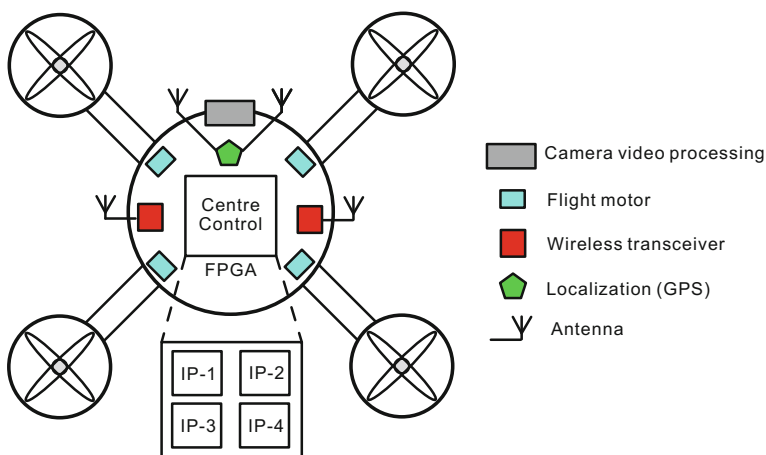
significantly reduce the R&D cycle and minimize/eliminate the designer-in-the-loop risk (like buggy coding) that could be a very annoying issue.

In wireless infrastructures, digitally-assisted RF concept and corresponding techniques have been used to facilitate and improve the physical RF transmission. For example, CFR function can reduce the signal peak to average power ratio (PAPR) helping RF power amplifiers working in an efficient way; DPD function pre-distorts the signal to combat against the nonlinearities introduced by real RF power amplifiers. The emerging Massive MIMO wireless system [8] will utilize digital algorithms (precoding) to dynamically shape antenna beams minimizing the energy radiated towards the unwanted areas, so that the system capacity and signal to interference ratio (SIR) can be significantly improved.

For mature applications, like wireless infrastructure, since the standard functions have been well studied with pre-defined input/output module interfaces, we can easily build a complex digital system by directly integrating the commercial IP-cores from FPGA vendors or the ones from 3rd-Party IP-core vendors or the ones designed in-house.

For emerging applications of smart hardware innovation, like designing a drone, IP-core integration approach would be the first choice to rapidly create a ready-to-use drone with minimized risk at prototyping stage. A simplified functional diagram of a typical drone is shown in Fig. 2.3, where contains the essential hardware sub-systems including flight motor control sub-system, camera video processing sub-system, wireless transceiver sub-system and localization sub-system. On top of those sub-systems, there is a centre control function (contains four IP-cores taking care of the corresponding four sub-systems) that will initialize, re-configure, co-ordinate the sub-systems.

Moreover, using IP-core integration approach to design a digital system will provide smooth upgrading flexibility, for example, if we decide to use 3G signal,



**Fig. 2.3** A simplified diagram of a drone with typical four sub-systems

like wideband code division multiple access (WCDMA) type wireless air waveform to transmit the video information from the drone to the ground station, we could directly place a WCDMA baseband processing IP-core in the FPGA to handle the corresponding waveform-related processing. Then later, if the WCDMA type wireless signal cannot satisfy the increasing data rate requirement for high-definition video with high frame rate, we'd like to upgrade the air waveform to 4G orthogonal frequency-division multiplexing (OFDM) based, we only need to replace the WCDMA baseband processing IP-core with an OFDM baseband processing IP-core in the FPGA.

Above all, as modern digital systems are getting more complicated, it is quite difficult and not an economic way to start the design from zero. Simply, we are not going to invent the wheel again. IP-core integration should be the first choice for system architects and DSP engineers, since this approach significantly simplifies the FPGA design procedures, especially for large digital systems.

## 2.3 FPGA-based DSP Basics

Signal processing has been widely and heavily utilized to improve our daily life, any smart thing you can think of with even just a little bit intelligence will have certain signal processing units behind. Particularly comparing to the analog signal processing, the digital signal processing (DSP) has quite a few advantages including more flexibility, lower power consumption, higher reliability, higher accuracy and much better scalability. DSP functions can be achieved on different types of hardware processors, like microcontroller, ARM, CPU, GPU, SoC, FPGA and so on. In this chapter, we will focus on the essentials of FPGA-based digital signal processing approaches.

### 2.3.1 *Fixed-Point Representation*

First things first! The first fundamental question of the FPGA-based DSP system design is how to describe a digital signal in the actual processing machine with limited resources.

In modern digital computing world, we use binary bits to represent a signal. Specifically, 1-bit binary contains two states 1 and 0, which are corresponding to voltage high and low physically.  $N$ -bit binary number contains  $2^N$  different stages, thus can be used to represent  $2^N$  different numbers. On top of this, two types of representations are normally used to describe a digital signal, i.e., floating-point manner and fixed-point manner. Floating-point approach represents and manipulates numbers via  $N$ -bit binary in a manner similar to scientific notation, i.e., a number is represented with a mantissa and an exponent (e.g.,  $a \times 2^b$ , where 'a' is the mantissa and 'b' is the exponent). While fixed-point approach is simpler and

just use a fixed number of bits to express fractional numbers (e.g.,  $2^{N-M} \cdot 2^M$ , where ‘ $M$ ’ is the number of bits to express fractional numbers).

Modern FPGAs can handle both floating-point processing and fixed-point processing, however comparing to the fixed-point processing, floating-point processing will need significant resources and running power on FPGAs. Comparing to the fixed-point processing, floating-point processing generally provides higher accuracy with higher resolution at cost of higher resource utilization and higher power consumption. Luckily, proper selection of the fixed-point representation with dynamical tuning the fixed-point position can achieve satisfactory performance comparing to the floating-point processing with negligible differences for majority industrial applications. In short, with given tolerance, fixed-point processing would be the first choice for FPGA-based DSP system design.

For the sake of simplicity, we directly give an example of how to describe the signal by a 4-bit fixed-point representation with different number of fractional bits as shown in Table 2.1. The term Fix\_4\_2 represents a 4-bit signed signal with 2-bit fractional part, while UFix\_4\_2 represents a 4-bit unsigned signal with 2-bit fractional part. In signed representation like Fix\_4\_2, the most significant bit (MSB) will be used as sign, i.e., 0 stands for positive while 1 stands for negative, and 2’s complement representation is utilized (will describe in the next sub-section). Particularly, the Fix\_4\_0 and UFix\_4\_0 are corresponding to 4-bit 2’s complement binary and 4-bit original binary, respectively.

**Table 2.1** Comparison of 4-bit fixed-point binary numbers with different fractional bits

Binary				Interpreted values					
MSB to LSB				Fix_4_0	Fix_4_1	Fix_4_2	UFix_4_0	UFix_4_1	UFix_4_2
b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> .b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> .b <sub>1</sub> b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> .b <sub>0</sub>	b <sub>3</sub> b <sub>2</sub> .b <sub>1</sub> b <sub>0</sub>
0	0	0	0	0	0.0	0.00	0	0.0	0.00
0	0	0	1	1	0.5	0.25	1	0.5	0.25
0	0	1	0	2	1.0	0.50	2	1.0	0.50
0	0	1	1	3	1.5	0.75	3	1.5	0.75
0	1	0	0	4	2.0	1.00	4	2.0	1.00
0	1	0	1	5	2.5	1.25	5	2.5	1.25
0	1	1	0	6	3.0	1.50	6	3.0	1.50
0	1	1	1	7	3.5	1.75	7	3.5	1.75
1	0	0	0	-8	-4.0	-2.00	8	4.0	2.00
1	0	0	1	-7	-3.5	-1.75	9	4.5	2.25
1	0	1	0	-6	-3.0	-1.50	10	5.0	2.50
1	0	1	1	-5	-2.5	-1.25	11	5.5	2.75
1	1	0	0	-4	-2.0	-1.00	12	6.0	3.00
1	1	0	1	-3	-1.5	-0.75	13	6.5	3.25
1	1	1	0	-2	-1.0	-0.50	14	7.0	3.50
1	1	1	1	-1	-0.5	-0.25	15	7.5	3.75

### 2.3.2 Two's Complement Binary Arithmetic Basic

The second question of the FPGA-based DSP approach is how to carry out the arithmetic using binary presentations, especially with minus sign.

One may think using the most significant bit (MSB) as sign and the rest of the bits to represent the amplitude as natural representation could do the work. However this approach will cause a systematic unstable situation when there is a positive zero and a negative zero (e.g., under this approach,  $(0000)_2$  refers to positive zero while  $(1000)_2$  will refer to negative zero).

2's complement approach solves the above issue and is one of the most important representations in the modern digital computing world. Let's take a closer look from 1's complement approach. Given an  $N$ -bit binary number  $x$ , the 1's complement of  $x$  is defined as an operation to invert every bit of the original binary number. For example, consider a binary number  $(0101)_2$  in natural representation, then 1's complement representation of this number is  $(1010)_2$ . Straightforwardly, 1's complement representation of an  $N$ -bit unsigned binary number can be derived by subtracting the number from the maximum value  $2^N - 1$ .

The so-called 2's complement of an  $N$ -bit binary number is defined as the complement with respect to the value  $2^N$ , and can be achieved simply by adding one to its 1's complement representation. By using this approach, the positive and negative zero dilemmas can be eliminated.

Furthermore, taking both unsigned and signed binary into consideration, the 1's complement and 2's complement operations of an  $N$ -bit binary number  $x$  can be described as Eqs. (2.1) and (2.2), respectively.

$$x_{1's} = 2^N - 1 - |x| \quad (2.1)$$

$$x_{2's} = x_{1's} + 1 = 2^N - |x| \quad (2.2)$$

The 2's complement representation makes fundamental arithmetic operations (like addition, subtraction, multiplication) for signed binary number quite simple, actually those operations are identical to those for unsigned binary numbers, which is really suitable for implementation of DSP functions on FPGAs. For example, subtraction can be done in the same way as addition by changing one operand sign using 2's complement representation. We provide a simple example as below, calculating " $5 - 2$ " in binary representation.

Approach 1: 4-bit subtraction operation using natural binary representation:

$$(0101)_2 - (0010)_2 = (0011)_2 \quad (2.3)$$

Approach 2: 4-bit addition operation, " $5 - 2$ " = " $5 + (-2)$ ". Using 2's complement representation, decimal value of  $(-2)$  will be  $2^N - |-2| = 14$  and its corresponding binary representation is  $(1110)_2$ , then the subtraction can be done in the



way of addition, producing the correct results (the overflow bit is removed without affecting the calculation result). 2's complement of a positive number is itself.

$$(0101)_{2's} + (1110)_{2's} = (0011)_{2's} = (0011)_2 \quad (2.4)$$

### 2.3.3 Real Number and Complex Number

The third question of the FPGA-based DSP approach is how to describe a complex signal and how to perform complex arithmetic.

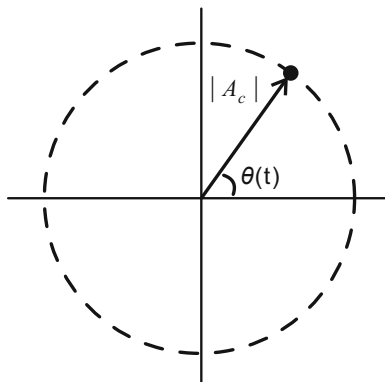
Simple signals contain the information that can be characterized by only real numbers, for example, voltage signal has amplitude information together with positive or negative information to indicate the direction of the voltage (positive or negative, e.g., +12 or -3.3 V). However some applications need complex representation of the signals. In wireless communications, a simple signal modulation scheme will shape a sine wave to encode information. For example, Eq. (2.5) illustrates a modulated signal sine wave  $s(t)$  with amplitude  $A_c$ , carrier frequency  $f_c$  and phase information  $\theta_c$ .

$$s(t) = A_c \cdot \cos(2\pi f_c t + \theta_c) \quad (2.5)$$

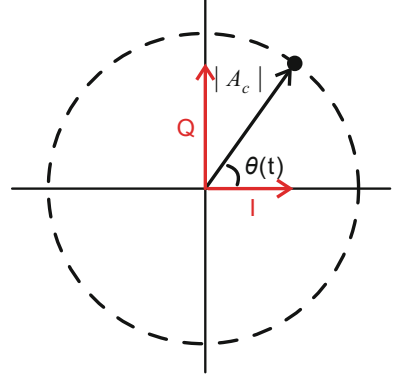
The polar representation of sine wave is shown in Fig. 2.4, where  $\theta(t)$  represents the instantaneous phase information.

However, fast high-precise phase varying of a high-speed carrier sine wave will be very difficult to achieve in the hardware circuits, resulting in very expensive to build such a modulation system in the hardware. An alternative way is to use so-called IQ modulation scheme. Let's reform the equation Eq. (2.5) as below

**Fig. 2.4** A modulated sine wave signal in the polar coordinates



**Fig. 2.5** A modulated sine wave signal in the Cartesian coordinate with I and Q data



$$\begin{aligned} A_c \cos(2\pi f_c t + \theta_c) &= A_c \cos(2\pi f_c t) \cos(\theta_c) - A_c \sin(2\pi f_c t) \sin(\theta_c) \\ &= I \cos(2\pi f_c t) - Q \sin(2\pi f_c t) \end{aligned} \quad (2.6)$$

where  $I = \cos(\theta_c)$  represents the amplitude of in-phase carrier and  $Q = \sin(\theta_c)$  represents the amplitude of quadrature-phase carrier. In this way, we transform the information from polar data system to the so-called Cartesian I and Q complex data system as shown in Fig. 2.5.

And the Eq. (2.5) can be simplified as below:

$$\begin{aligned} s(t) &= s_I(t) + s_Q(t) \\ \text{where } \begin{cases} s_I(t) &= I \cos(2\pi f_c t) \\ s_Q(t) &= -Q \sin(2\pi f_c t) \end{cases} \end{aligned} \quad (2.7)$$

In the modern wireless systems, complex IQ signals can be simply represented by two signed binary numbers in the hardware, one for in-phase I part and other one for quadrature-phase Q part. Now let's have a closer look at the basic arithmetic of complex signals, given two complex numbers  $a = a_I + ja_Q$  and  $b = b_I + jb_Q$ .

Complex addition and subtraction can be processed separately as below:

$$\begin{aligned} a \pm b &= (a_I + ja_Q) \pm (b_I + jb_Q) \\ &= (a_I \pm b_I) + j(a_Q \pm b_Q) \end{aligned} \quad (2.8)$$

Complex multiplication processed by the direct architecture with 4 real multipliers and 2 real additions (subtraction is considered as addition):

$$\begin{aligned} a \cdot b &= (a_I + ja_Q) \cdot (b_I + jb_Q) \\ &= (a_I \cdot b_I - a_Q \cdot b_Q) + j(a_I \cdot b_Q + a_Q \cdot b_I) \end{aligned} \quad (2.9)$$

Complex multiplication processed by the compact architecture with 3 real multipliers and 5 real additions:

$$\begin{aligned} a \cdot b &= (a_I + ja_Q) \cdot (b_I + jb_Q) \\ &= (a_I \cdot b_Q - a_Q \cdot b_Q) + j[(a_I + a_Q) \cdot (b_I + b_Q) - a_I b_I - a_Q b_Q] \end{aligned} \quad (2.10)$$

Please note that in the above examples, the addition operation between I part and Q part are just in place for illustration purpose, and I and Q signals will be two separated signals physically in the hardware.

### 2.3.4 Data Sampling Rate and Processing Rate

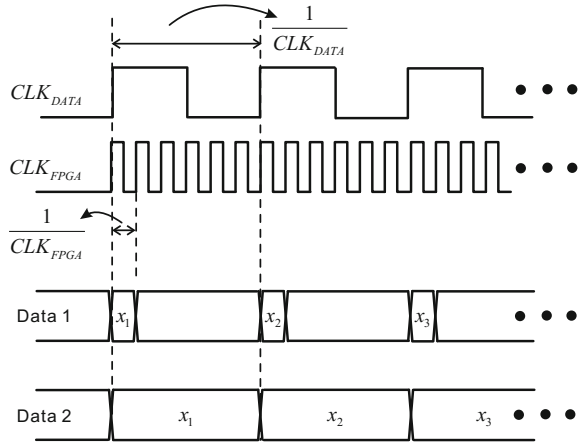
The fourth question of FPGA-based DSP approach is how to decide the data sampling rate and processing rate.

Based on the information theory, particular Nyquist-Shannon sampling theorem, in order to fully capture an analog signal of finite bandwidth, we need to sample the signal at a rate at least of the two times of signal bandwidth. In FPGA-based DSP system, the data sampling rate refers to the input digital signal refreshing rate which is associated with the type of signals. For example, in the 4G wireless network, one component carrier of long term evolution (LTE) advanced signal will need a corresponding 30.72 mega samples per second (MSPS) sampling rate. Each sample is an  $N$ -bit (e.g., 16-bit) complex binary number for both I-part and Q-part. In this case, the refreshing rate  $CLK_{DATA}$  of the input data will be 30.72 MHz and 32-bit complex IQ data will be updated per refreshing. Processing rate refers to the actual FPGA running clock  $CLK_{FPGA}$ , i.e., how fast the FPGA chip is operating for a given DSP task. Figure 2.6 illustrates the data rate and processing rate, where  $CLK_{FPGA} = 6 CLK_{DATA}$  as an example. Though the contents and refreshing rate of Data 1 and Data 2 are the same, the valid time of each sample are different in the two cases. Each sample of Data 2 is valid for one cycle of  $CLK_{DATA}$  while each sample of Data 1 is valid for one cycle of  $CLK_{FPGA}$ .

In general, the processing rate of FPGA  $CLK_{FPGA}$  should be at least as the same as input data sampling rate  $CLK_{DATA}$ . As the modern semiconductor technology improving, the state-of-the-art FPGAs can be reliably operating at couple of hundred MHz. Without any power constraints, the higher of the processor the better of DSP performance, simply because we can do more tasks within the same time period. Using higher-than required processing rate in the FPGA-based DSP-design has two direct appealing impacts: (1) lower processing latency and (2) lower processing resource utilization. Two examples are provided below.

An example of lower processing latency: given the data sampling rate  $CLK_{DATA} = 4$  MHz and we are going to perform complex multiplication by a

**Fig. 2.6** Illustration of data rate and processing rate



complex multiplier IP-core with 4 clock processing delays. If the processing rate  $CLK_{FPGA} = 4$  MHz, which is the same as the data sampling rate, the latency between sending complex data in until pushing complex data out is 4 clocks, which is  $4 \times 1/4$  MHz = 1 ms latency. However, if the processing rate  $CLK_{FPGA} = 40$  MHz, which is 10 times of data sampling rate, the latency will be  $4 \times 1/40$  MHz = 0.1 ms in time, in other words, the processing latency has been reduced 10 times.

An example of lower processing resource utilization: given the data sampling rate  $CLK_{DATA} = 4$  MHz and we are going to perform complex multiplication for 10 data branches simultaneously. If the processing rate  $CLK_{FPGA} = 4$  MHz, which is the same as the data sampling rate, we will need 10 complex multiplier IP-cores operating in parallel with total 4 clock processing delay, which is 1 ms as the same as in the last example. However, if the processing rate  $CLK_{FPGA} = 40$  MHz, which is 10 times of data sampling rate. Processing one complex multiplication will consume 4 clock cycles, which is 0.1 ms at the  $CLK_{FPGA} = 40$  MHz, then within the same latency 1 ms, we can finish the complex multiplications for 10 branches using only one complex multiplier IP. We will explain more in the design strategy sub-section regarding this point.

### 2.3.5 Accuracy and Complexity

Without any optimization algorithms or design tricks, in general the complexity and accuracy of the FPGA-based DSP system blocks are a pair of trade-offs. Accuracy will be preliminary determined by the bit-width, i.e., enlarging the bit-width of a signal will directly increase the accuracy of the processing, but simultaneously resulting in the increments of the complexity (occupying more silicon area) due to the requirement for processing more bits per second.

The practical rules are quite simple: for performance-driven applications, we will use more resource to ensure high accurate processing, while for resource-constrained applications, we should maximize the processing accuracy by using the available resources. As design resources are in general constrained, in order to achieve better revenue to cost ratio for a product/solution, low complexity algorithms (achieving the same functionality) with smart design tricks that can minimize resources utilization will be always welcome in the DSP system design.

### 2.3.6 Dynamic Processing Range and Overflow

In a DSP system containing multiple processing stages, each stage may need a different processing range depending on the processing requirements. Figure 2.7 illustrates a generalized multiple stages processing system. Each DSP stage may increase the bit-width of the signals to combat against the so-called overflow issue due to the nature of binary processing. For example, a  $\text{Fix}_{N_1, N_{F,1}}$  number multiplying by a  $\text{Fix}_{N_2, N_{F,2}}$  number will result in a number in the form of  $\text{Fix}_{(N_1 + N_2), (N_{F,1} + N_{F,2})}$  to ensure the output is not out of “legal” representation range causing overflow problems. However, in a multi-stage DSP system, the bit-width may grow to a large number that is not possible or very difficult to handle by the next processing stage. Proper “re-arranging” functions are required.

For fixed-point processing, a dynamic range tuning function can be used to scale (left shift or right shift operation) and select (by slicing) the necessary/proper bit-widths for the next stage. Please note, those dynamic range tuning functions will cost nothing in the hardware.

Let’s have a look at a quick example with meaningful settings. Assume a real multiplication operation,  $a \times b = c$ , where both inputs ( $a$  and  $b$ ) are within the range of  $[-1, 1)$  thus 16-bit fixed-point representation of inputs will be  $\text{Fix}_{16,15}$ . Without any range tuning function, the output  $c$  will be in the form of  $\text{Fix}_{32,30}$  as full precision processing. Mathematically, if we carry out a multiplication between two numbers of range  $[-1, 1)$ , the output will be within the same range  $[-1, 1)$ . Therefore, if 16-bit resolution would be enough for the next stage processing, we can select the most left 16-bit out of 32-bit and move corresponding binary point to the 14-bit with regards to the least bit, e.g.,  $\text{Fix}_{16,14}$ , this is equivalent to perform the quantization operation on the high dynamic range digital signals. In this way,

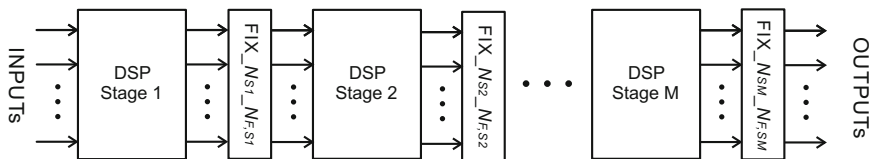


Fig. 2.7 Generalized  $M$ -stage signal processing with dynamic processing range

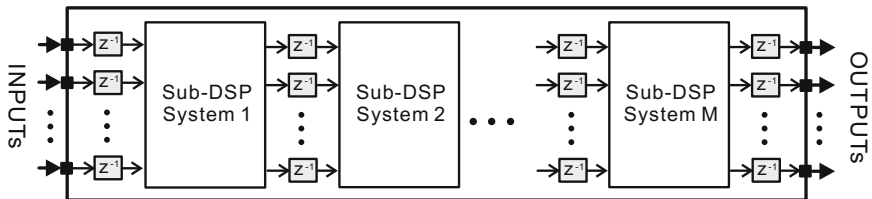
next stage will have 16-bit inputs as starting point. Best practice regarding this point is to scale the inputs and outputs of each processing stage to  $[-1, 1)$ , and tailor the representation by the available processing bit-width.

## 2.4 FPGA-based DSP System Design

Design a DSP system on the FPGA platform has been made quite easy and straightforward by the revolutionary efforts of FPGA vendors like Xilinx and Altera (An Intel Company). In the following sub-sections, we summarized the essential design strategies that will be helpful during FPGA-based DSP system design procedures. We recommend using those strategies from every beginning of a FPGA-based DSP system design project, eliminating the avoidable issues.

### 2.4.1 Self-contained Modular Design Strategy

Great wall was built by small pieces! A complex FPGA-based DSP system will be built by a few logical sub-functions, each of which is taking care of some processing tasks. Creating reasonable self-contained sub-modules (proper partitioning the FPGA regarding the DSP sub-systems) is the essential point to achieve a high performance DSP system on the FPGA platforms. The fundamental rule is to use so-called modular design strategy. If you have ever purchased some furniture from IKEA, you may notice that a lot of products contain some similar parts, so yes, they follow the modular design strategy. The parts they designed can be reused on different products, saving the cost and making the re-construction work easier. This self-contained modular design strategy together with re-timing registers between multiple sub-modules (as shown in the Fig. 2.8) is highly recommended as the base architecture of the FPGA-based DSP system (more detailed examples will be provided in following chapters), because:



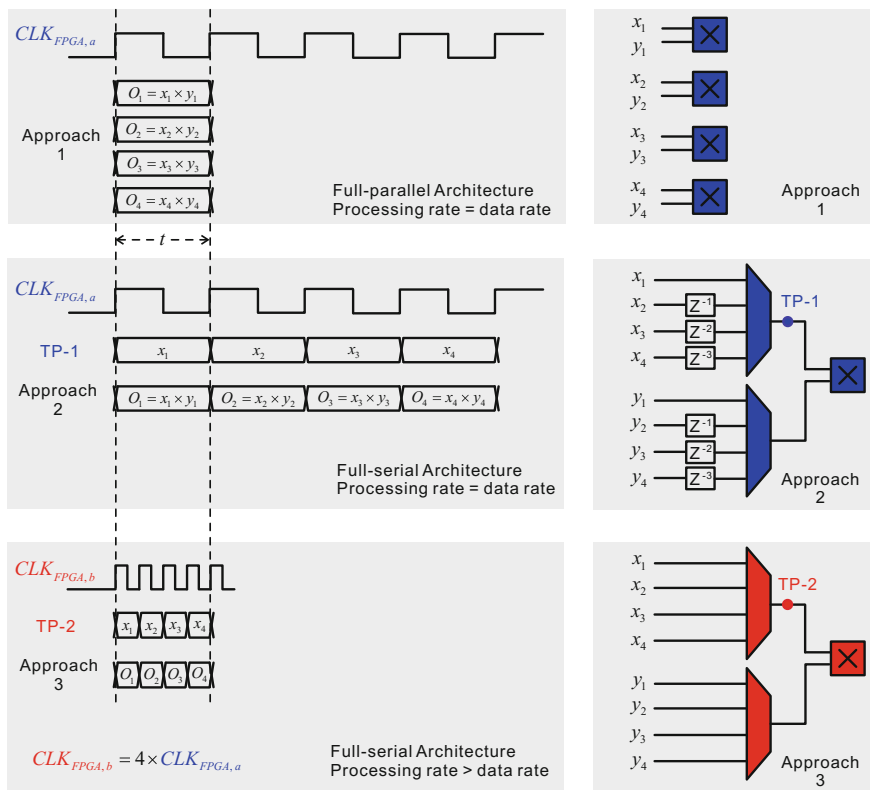
**Fig. 2.8** Base implementation architecture of FPGA-based DSP system using modular design strategy with re-timing registers, which can be used as an architectural template for creating robust FPGA-based DSP systems

- (1) Smaller sub-modules are relatively easy to handle both in the simulation and on silicon debugging/verification;
- (2) Reasonable hierarchies of the sub-modules with pipelined registers can prevent FPGA timing issue for some certain degree and leave comfortable optimization space for the FPGA synthesizer and implementation tool to properly route the logics in the chip. Also in this way, it is relatively easy to find the critical path that is associated with potential or existing timing hazards;
- (3) Upgrading the modular designed DSP system is straightforward, i.e., simply replacing the sub-module by a modified one. This approach won't affect any other validated sub-modules and avoids the effort for re-designing the similar function blocks in different projects.

### 2.4.2 Time-Space Trade-off Implementation Strategy

Time-space trade-off implementation strategy naturally comes along with modern FPGA because of its inherent parallel processing capability. You may hear it before in the form of latency-resource trade-off on the CPU processing platform. The basic concept of this time-space trade-off enables very flexible design approaches: (1) in a resource-rich scenario (e.g., plenty of design resources on the chip), digital functions can be designed in the full parallel way to minimize the input to output latency, i.e., using more silicon processing area (space) to gain less processing latency (time); (2) in a resource-non-rich environment (e.g., limited resources assigned for achieving one digital function), the same physical resources have to be re-used to handle multiple tasks causing extra processing latency, i.e., reducing the resource utilization (space) by using multiple cycles processing (time). Figure 2.9 illustrates an example of time-space trade-off for multiple independent multiplication operations. Approach 1, i.e., full parallel architecture (when processing rate is equal to data rate) utilizes 4 multipliers to carry out 4 independent multiplication operations within one unit of time  $t$ , while the approach 2, i.e., full-serial architecture (when processing rate is equal to data rate) utilize 1 multiplier to process 4 multiplication operations but requiring 4 units of time.

As the silicon process improving, modern FPGAs can be running at high-speed, e.g., over 800 MHz on Xilinx Ultrascale FPGA device fabric has been reported, thus we can apply a technique, which is termed as time division multiplexing (TDM), to reuse the same physical resources in different time periods. In Fig. 2.9, the TDM idea is illustrated by approach 3, which contains the same full-serial architecture but in this case the processing rate is higher than data rate. Parallel inputs at data rate are multiplexed at processing rate and generating a high-speed serial data stream. In this way, 1 multiplier can process 4 multiplications within 4 shorter clock cycles (due to the higher processing rate), which is equivalent to one unit of time  $t$ . Comparing approach 3 and approach 1, it is not difficult to realize that



**Fig. 2.9** Time-space trade-off illustration using an example of four multiplication operations

increasing the processing rate within physically achievable range will bring us the benefits of applying TDM approach, which will carry out the same DSP function with lower physical resource utilization.

### 2.4.3 Model-based Design Flow Using System Generator

Next, let's have a look how we can build a DSP system on a FPGA. First of all, DSP systems are naturally structured in steps, in other words, DSP systems can be abstracted/defined by a few mathematical equations. This is the beauty of the DSP systems, i.e., so-called black-box effect. Given the inputs and some pre-defined system parameters, the black-boxes (DSP systems) produce the desired outputs if the "boxes" have been properly described. For the sake of simplicity, we name those boxes as models, which reflect certain functions or some particular input-to-output mapping relationship using mathematical equations.



Quite a few design approaches are available to create a DSP system on the FPGA, the top three of most popular approaches include:

- (1) Hardware description language (HDL)-based approach, e.g., using VHDL, Verilog, System Verilog to directly describe the hardware behavior at the RTL level;
- (2) High level synthesis (HLS)-based approach, e.g., using C type high level language synthesis tool to describe the function behavior and create corresponding digital hardware logics;
- (3) Model-based approach, e.g., using and extending MATLAB/Simulink [9] blocks to enable hardware design with high-level abstractions.

Among those three, because of its easy-to-use and quick-to-verify features at the system level, the model-based approach is quite suitable for DSP design on FPGAs. Xilinx provides a high-performance DSP design tool, i.e., System Generator for DSP [10], to help DSP/FPGA Engineers create production-quality DSP functions on a FPGA with short R&D time. We will use this Xilinx tool to tell the story in this book.

Based on Xilinx recommendations and our FPGA-based DSP design experience since 2006, System Generator can be quite helpful at the R&D prototyping stage for many scenarios. At early R&D stage of a brand new product, your algorithms are under developing so you may want to explore the algorithms regarding its system-level performance and corresponding complexity but without translating those “half-finished” algorithms into real hardware. Or at the prototyping stage for adding in new features on top of the existing products, you only need to design an add-on DSP function as part of the existing whole system design, however you have physical resource constraints so you may want to evaluate the new add-on DSP function first regarding its complexity with regards to the available physical resources left. In common, the model-based design flow using System Generator contains the following procedures (Table 2.2).

We will use this design flow as the base procedures in Chaps. 3–5.

#### **2.4.4 Overview of MATLAB and Simulink in FPGA DSP Design**

Due to its powerful scientific computing capability and flexible functional blocks re-customizability, MATLAB together with Simulink, has become the preferred algorithm R&D tool that is widely utilized in the both academia and industry. Precisely, MATLAB and Simulink are quite helpful in FPGA-based DSP system design in the following three perspectives:

- (1) DSP Design preparation: Inherently organized in a column vector way, MATLAB provides quite efficient mathematical computing for vectors-based signal processing, especially in the wireless communication area. Before

**Table 2.2** Model-based design flow using system generator

Step	Task	Target
1	Reference algorithm exploration using floating-point Simulink modules and partition the big design into smaller functional sub-modules	Ensure algorithm with desired functionalities and proper sub-modules partitioning
2	System parameters exploration regarding the interfaces between neighbour sub-modules	Ensure only necessary information pass to the next processing stage
3	Design constraints analysis with regards to the actual FPGA resources and latency requirement	Be clear with design constrains, both available resource and latency
4	Choose base implementation architecture, target device and corresponding clock parameters	Ensure proper system setting, e.g., data rate and processing rate ...
5	Design the first sub-module using Xilinx tool-box in the System Generator; and verify the designed sub-module by comparing its outputs with counterparts in the reference model given the same inputs as excitation	Ensure the functionality and the accuracy of the designed sub-module meet the requirement
6	Repeat step 5 for the rest sub-modules	Ensure the functionalities of the sub-modules and interfaces
7	Whole system verification from original inputs to the final outputs	Ensure the functionality and accuracy of the whole module meet the requirement
8	Perform hardware-in-loop co-simulation (if possible)	Ensure the designed DSP system is functionally working on the real FPGA device
9	Generate design outputs, which can be VHDL, Verilog or synthesizable netlist, ...	Produce the design files for integration

starting anything in the hardware, it is critical to ensure that the “ideas”, i.e., DSP algorithms, are functional with given assumptions. We call this algorithm simulation. MATLAB makes this simulation relatively easy, and moreover, with the help of Simulink, you will be able to see the dynamic processing flow for your algorithms. In order words, MATLAB and Simulink provide a chance to know the inside out performance and functionality of your algorithms.

- (2) DSP Design: Once you have a good and reliable algorithm simulation, translating the DSP algorithm into FPGA hardware can be easily achieved by MATLAB and Simulink together with Xilinx System Generator. We will provide more detailed examples from Chaps. 3–5.
- (3) DSP Design validation: In order to make your DSP system work properly in the hardware, you will need to verify the functionality and timing performance of your design. For functionality validation, we recommend carry out comparison between designed module and reference MATLAB/Simulink model step by step. For timing validation, we recommend avoiding timing issue at

the early design stage using modular design strategy with re-timing registers, rather than solving the timing issue at the implementation stage. Simply because, the early you solve this timing problem, the less messy of your DSP system design project, and the shorter time you spend on the whole project and the more robust of your DSP module.

## 2.5 Conclusions

In this chapter, we quickly re-flashed the FPGA evolution and FPGA-based digital signal processing at the system level. And moreover, we introduced the essential and fundamental elements in the FPGA-based DSP system design including signal representation and model-based DSP system design strategy, which will be utilized in the following chapters with real examples in the wireless applications, especially the digital convolution applications.

## References

1. Xilinx company website. <http://www.xilinx.com>
2. Altera (now part of Intel) company website. <http://www.altera.com>
3. Lattice Semiconductor company website. <http://www.latticesemi.com>
4. Microsemi company website. <http://www.microsemi.com>
5. ARM company website. <http://www.arm.com>
6. Nvidia company website, <http://www.nvidia.com>
7. Luo F (2011) Digital front-end in wireless communications and broadcasting. Cambridge University Press, Cambridge
8. Weldon MK (2016) The future X network: a Bell Labs perspective. CRC Press, New York
9. MathWorks company website. <http://www.mathworks.com>
10. Xilinx (2015) Model based DSP design using system generator. User Guide, UG897

FPGA-based Digital Convolution for Wireless  
Applications

Guan, L.

2017, XVII, 151 p. 128 illus., 61 illus. in color.,  
Hardcover

ISBN: 978-3-319-51999-9