Chapter 2
Hardware Platforms

2.1 Chapter Overview

As discussed in Chapter 1, single-threaded software applications no longer obtain significant gains in performance with the current processor scaling trends. With the growing complexity of VLSI designs, this is a significant problem for the electronic design automation (EDA) community. In addition to multi-core processors, hardware-based accelerators such as custom-designed ICs, reconfigurable hardware such as FPGAs, and streaming processors such as graphics processing units (GPUs) are being investigated as a potential solution to this problem. These platforms allow the CPU to offload compute-intensive portions of an application to the hardware for a faster computation, and the results are transferred back to the CPU upon completion. Different platforms are best suited for different application scenarios and algorithms. The pros and cons of the platforms under consideration are discussed in this chapter.

The rest of this chapter is organized as follows. Section 2.2 discusses the hardware platforms studied in this monograph, with a brief introduction of custom ICs, FPGAs, and GPUs in Section 2.3. Sections 2.4 and 2.5 compare the hardware architecture and programming environment of these platforms. Scalability of these platforms is discussed in Section 2.6, while design turn-around time on these platforms is compared in Section 2.7. These platforms are contrasted for performance and cost of hardware in Sections 2.8 and 2.9, respectively. The implementation of floating point operations on these platforms is compared in Section 2.10, while security concerns are discussed in Section 2.11. Suitable applications for these platforms are discussed in Section 2.12. The chapter is summarized in Section 2.13.

2.2 Introduction

Most hardware accelerators are not stand-alone platforms, but are co-processors to a CPU. In other words, a CPU is needed for initial processing, before the compute-intensive task is off-loaded to the hardware accelerators. In some cases the hardware

K. Gulati, S.P. Khatri, *Hardware Acceleration of EDA Algorithms*,
DOI 10.1007/978-1-4419-0944-2_2,

accelerator might communicate with the CPU even during the computation. The different platforms for hardware acceleration in this monograph are compared in the following sections.

2.3 Hardware Platforms Studied in This Research Monograph

2.3.1 Custom ICs

Traditionally, custom ICs are included in a product to improve its performance. With a high production volume, the high manufacturing cost of the IC is easily amortized. Among existing hardware platforms, custom ICs are easily the fastest accelerators. By being application specific, they can deliver very high performance for the target application. There exist a vast literature of advanced circuit design techniques which help in reducing the power consumption of such ICs while maintaining high performance [36]. Some of the more well-known techniques to reduce power consumption (both dynamic and leakage) are design and protocol changes [31, 20], reducing supply voltage [17], variable Vt devices, dynamic bulk modulation [39, 40], power gating [18], and input vector control [25, 16, 41]. Also, newer gate materials which help achieve further performance gains at a low power cost are being investigated [32]. Due to their high performance and small footprint, custom ICs are the most suitable accelerators for space, military, and medical applications that are compute intensive.

2.3.2 FPGAs

A field-programmable gate array (FPGA) is an integrated circuit which is designed to be configured by the designer in the field. The FPGA is generally programmed using a hardware description language (HDL). The ability of the user to program the functionality of the FPGA in the field, along with the low non-recurring engineering costs (relative to a custom IC design), makes the FPGA an attractive platform for many applications. FPGAs have significant performance advantages over microprocessors due to their highly parallel architectures and significant flexibility. Hardware-level parallelism allows FPGA-based applications to operate 1 to 2 orders of magnitude faster than equivalent applications running on an embedded processor or even a high-end workstation. Compared to custom ICs, FPGAs have a somewhat lower performance, but their reconfigurability makes them an easy choice for several (particularly low-volume) applications.

2.3.3 Graphics Processors

General-purpose graphics processors turn the massive computational power of a modern graphics accelerator into general-purpose computing power. In certain

applications which include vector processing, this can yield several orders of magnitude higher performance than a conventional CPU. In recent times, general-purpose computation on graphics processors has been actively explored for several scientific computations [23, 34, 29, 35, 24]. The rapid increase in the number and diversity of scientific communities exploring the computational power of GPUs for their data-intensive algorithms has arguably had a contribution in encouraging GPU manufacturers to design GPUs that are easy to program for general-purpose applications as well. GPU architectures have been continuously evolving toward higher performance, larger memory sizes, larger memory bandwidths, and relatively lower costs. Additionally, the development of open-source programming tools and languages for interfacing with the GPU platforms, along with the continuous evolution of the computational power of GPUs, has further fueled the growth of general-purpose GPU (GPGPU) applications.

A comparison of hardware platforms considered in this monograph is presented next, in Sections 2.4 through 2.12.

2.4 General Overview and Architecture

Custom-designed ICs have no fixed architecture. Depending on the algorithm, technology, target application, and skill of the designers, custom ICs can have extremely diverse architectures. This flexibility allows the designer to trade off design parameters such as throughput, latency, power, and clock speed. The smaller features also open the door to higher levels of system integration, making the architecture even more diverse.

FPGAs are high-density arrays of reconfigurable logic, as shown in Fig. 2.1 [14]. They allow a designer the ability to trade off hardware resources versus performance, by giving the hardware designers the choice to select the appropriate level of parallelism to implement an algorithm. The ability to tradeoff parallelism and pipelining yields significant architectural variety. The circuit diagram for a typical FPGA logic block is shown in Fig. 2.2, and it can implement both combinational and sequential logic, based on the value of the MUX select signal X. The lookup table (LUT) in this FPGA logic block is shown in Fig. 2.3. It consists of a 16:1 MUX circuit, implemented using NMOS passgates. This is the typical circuit used for implementing LUTs [30, 21]. The circuit for the 16 SRAM configuration bits (labeled as 'S' in Fig. 2.3) is shown in Fig. 2.4. The DFF of Fig. 2.2 is implemented using identical master and slave latches, each of which has an NMOS passgate connected to the clock and a pair of inverters in a feedback configuration to implement the storage element.

In the FPGA paradigm, the hardware consists of a regular array of logic blocks. Wiring between these blocks is achieved by reconfigurable interconnect, which can be programmed via passgates and SRAM configuration bits to drive these passgates (and thereby customize the wiring).

Recent FPGAs provide on-board hardware IP blocks for DSP, hard processor macros, and large amounts of on-chip block RAM (BRAM). These hardware IP

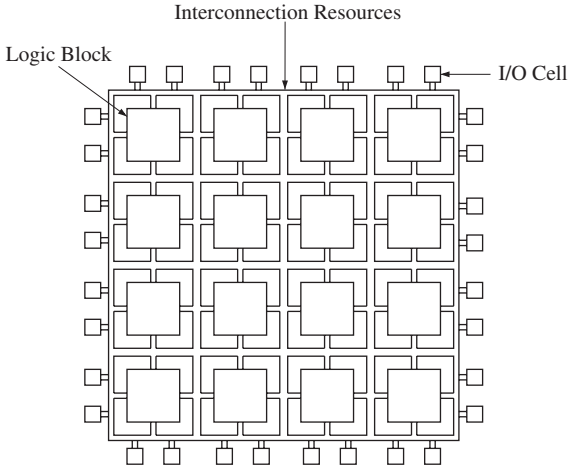

Fig. 2.1 FPGA layout [14]

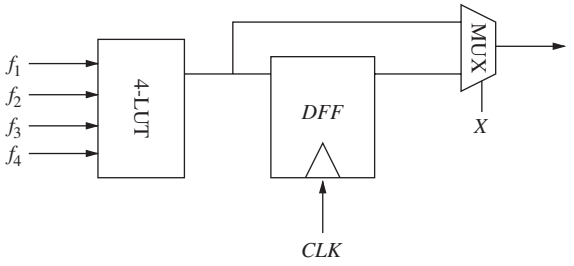

Fig. 2.2 Logic block in the FPGA

blocks allow a designer to perform many common computations without using FPGA logic blocks or LUTs, resulting in a more efficient design.

One downside of FPGA devices is that they have to be reconfigured every time the system is powered up. This requires the use of either a special external memory device (which has an associated cost and consumes real estate on the board) or an on-board microprocessor (or some variation of these techniques).

GPUs are commodity parallel devices which provide extremely high memory bandwidths and a large number of programmable cores. They can support thousand of simultaneously issued software threads operating in a SIMD fashion. GPUs have several multiprocessors which execute these software threads. Each multiprocessor has a special function unit, which handles infrequent, expensive operations, like divide and square root. There is a high bandwidth, low latency local memory attached to each multiprocessor. The threads executing on that multiprocessor can communicate among themselves using this local memory. In the current generation of NVIDIA GPUs, the local memory is quite small (16 KB). There is also a large global device memory (over 4 GB in some models) of GPU cards. Virtual memory is not implemented, and so paging is not supported. Due to this limitation,

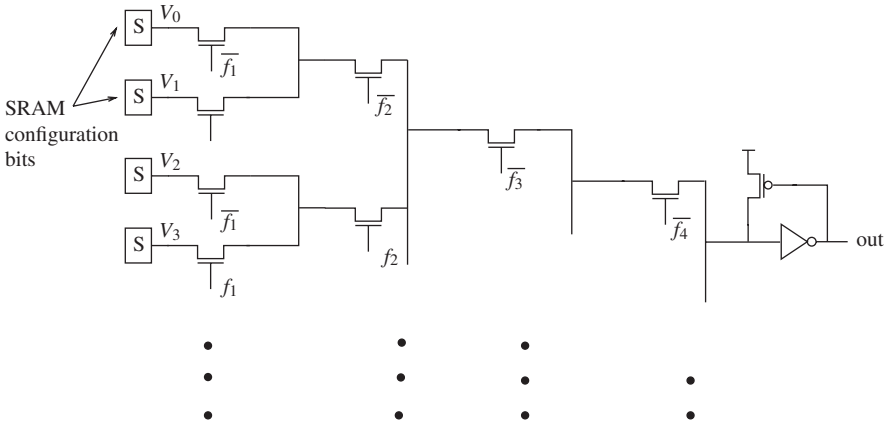

Fig. 2.3 LUT implementation using a 16:1 MUX

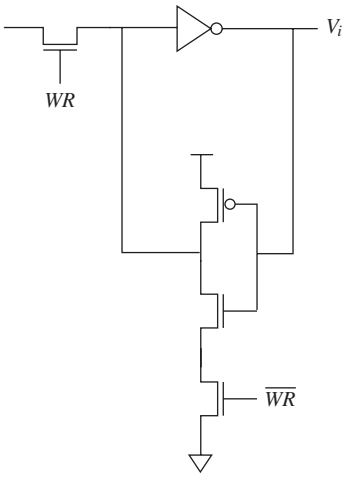

Fig. 2.4 SRAM configuration bit design

all the data has to fit in the global memory. The global device memory has very high bandwidth (but also has high latency) to the multiprocessors. The global device memory is not directly accessible by the host CPU nor is the host memory directly accessible to the GPU. Data from the host that needs to be processed by the GPU must be transferred via DMA (across an IO bus) from the host to the device memory. Similarly, data is transferred via DMA from the GPU to the CPU memory as well. GPU memory bandwidths have grown from 42 GB/s for the ATI Radeon X1800XT to 141.7 GB/s for the NVIDIA GeForce GTX 280 GPU [37]. A recent comparison of the performance in Gflops of GPUs to CPUs is shown in Fig. 2.5. A key drawback of the current GPU architectures (as compared to FPGAs) is that the on-chip memory cannot be used to store the intermediate data [22] of a

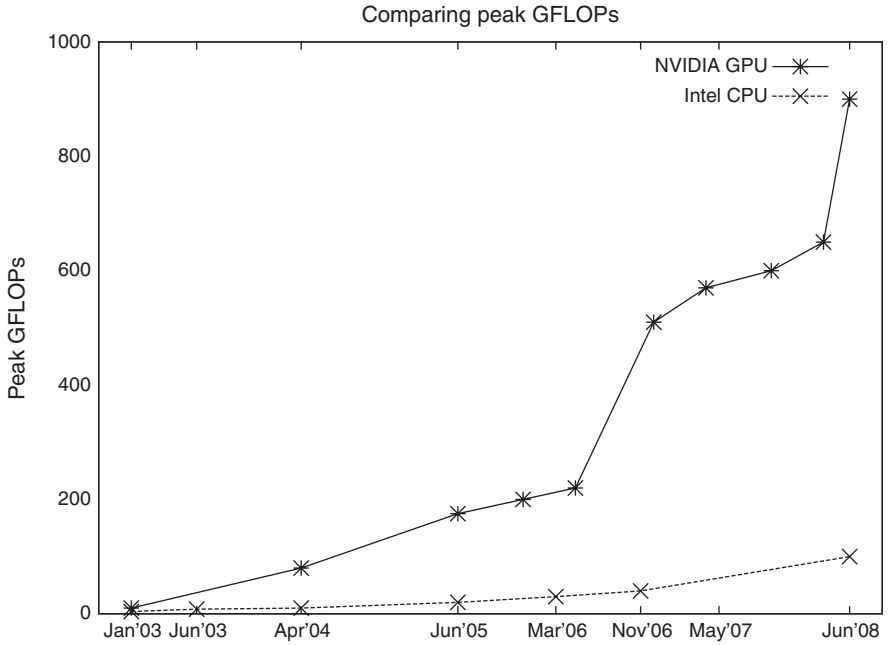

Fig. 2.5 Comparing Gflops of GPUs and CPUs [11]

computation. Only off-chip global memory (DRAM) can be used for storing intermediate data. On the FPGA, processed data can be stored in on-chip block RAM (BRAM).

2.5 Programming Model and Environment

Custom-designed ICs require several EDA tools in their design process. From functional correctness at the RTL/HDL level to the hardware testing and debugging of the final silicon, EDA tools and simulators are required at every step. For certain steps, a designer has to manually fix the design or interface signals to meet timing or power requirements. Needless to say, for ICs with several million transistors, design and testing can take months before the hardware masks are finalized for fabrication. Unless the design and manufacturing cost can be justified by large volumes or extremely high performance requirements, the custom design approach is typically not practical.

FPGAs are generally customized based on the use of SRAM configuration cells. The main advantage of this technique is that new design ideas can be implemented and tested much faster compared to a custom IC. Further, evolving standards and protocols can be accommodated relatively easily, since design changes are much simpler to incorporate. On the FPGA, when the system is first powered up, it

can initially be programmed to perform one function such as a self-test and/or board/system test, and it can then be reprogrammed to perform its main task. FPGA vendors provide software and hardware IP cores [3] that implement several common processing functions. More recently, high-end FPGAs have become available that contain one or more embedded microprocessors. Tasks that used to be performed by an external microprocessor can now be moved into the FPGA core. This provides several advantages such as cost reduction, significantly reduced data transfer times from FPGA to the microprocessor, simplified circuit board design, and a smaller, more power-efficient system. Debugging the FPGA is usually performed using embedded logic analyzers at the bitstream level [26]. FPGA debugging, depending on the design density and complexity, can easily take weeks. However, this is still a small fraction of the time taken for similar activities in the custom IC approach. Given these advantages, FPGAs are often used in low- and medium-volume applications.

In the recent high-level languages released for interfacing with GPUs, the hardware details of the graphics processor are abstracted away. High-level APIs have made GPU programming very flexible. Existing libraries such as ACML-GPU [2] for AMD GPUs and CUFFT and CUBLAS [4] for NVIDIA GPUs have inbuilt efficient parallel implementations of commonly used mathematical functions. CUDA [10] from NVIDIA provides guidelines for memory access and the usage of hardware resources for maximal speedup. Brook+ [2] from AMD-ATI provides a lower level API for the programmer to extract higher performance from the hardware. Further, GPU debugging and profiling tools are available for verification and optimization. In comparison to FPGAs or custom ICs, using GPUs as accelerators incurs a significantly lower design turn-around time.

General-purpose CPU programming has all the advantages of GPGPU programming and is a mature field. Several programming environments, debugging and profiling tools, and operating systems have been around for decades now. The vast amount of existing code libraries for CPU-based applications is an added advantage of system implementation on a general-purpose CPU.

2.6 Scalability

In high-performance computing, scalability is an important issue. Combining multiple ICs together for more computing power and using an array of FPGAs for emulation purposes are known techniques to enhance scalability. However, the extra hardware usually requires careful reimplementation of some critical portions of the design. Further, parallel connectivity standards (PCI, PCI-X, EMIF) often fall short when scalability and extensibility are taken into consideration.

Scalability is hard to achieve in general and should be considered during the architectural and design phases of FPGA-based or custom IC-based algorithm acceleration efforts. Scalability concerns are very specific to the algorithm being targeted, as well as the acceleration approach employed.

For graphics processors, existing techniques for scaling are intracluster and intercluster scaling. GPU providers such as NVIDIA and AMD provide multi-GPU solutions such as [12] and [1], respectively. These multi-GPU architectures claim high scalability, in spite of limited parallel connectivity, provided the application lends itself well to the architecture. Scalability requires efficient use of hardware as well as communication resources in multi-core architectures, custom ICs, FPGAs, and GPUs. Architecting applications for scalability remains a challenging open problem for all platforms.

2.7 Design Turn-Around Time

Custom ICs have a high design turn-around time. Even for modest sized designs, it takes many months from the start of the design to when the silicon is delivered. If design revisions are required, the cost and design turn-around time of custom ICs can become even higher.

FPGAs offer better flexibility and rapid prototyping capabilities as compared to custom designs. An idea or concept can be tested and verified in an FPGA without going through the long and expensive fabrication process of custom design. Further, incremental changes or design revisions (on an FPGA) can be implemented within hours or days instead of months. Commercial off-the-shelf prototyping hardware is readily available, making it easier to rapidly prototype a design. The growing availability of high-level software tools for FPGA design, along with valuable IP cores (prebuilt functions) for several commonly used control and signal processing tasks, makes it possible to achieve rapid design turn-arounds.

GPUs and CPUs allow for a far more flexible development environment and faster turn-around times. Newer compilers and debuggers help trace software bugs rapidly. Incremental changes or design revisions can be compiled much faster than in custom IC or FPGA designs. Code profiling technique for optimization purposes is a mature area [15, 10]. Thus, a software implementation can easily be used to rapidly prototype a new design or to modify an existing design.

2.8 Performance

Depending on the application, custom-designed ICs offer speedups of several orders of magnitude as compared to the single-threaded software performance on the CPU. However, as mentioned earlier, the time taken to design an IC can be prohibitive. FPGAs provide a performance that is intermediate between that of custom ICs and single-threaded CPUs. Hardware-level parallelism allows some FPGA-based applications to operate 1–2 orders of magnitude faster than an equivalent application running on a higher-end workstation. More recently, high-performance system designers have begun to explore the capabilities of FPGAs [28]. Advances in FPGA tool flows and the increasing FPGA speed and density characteristics

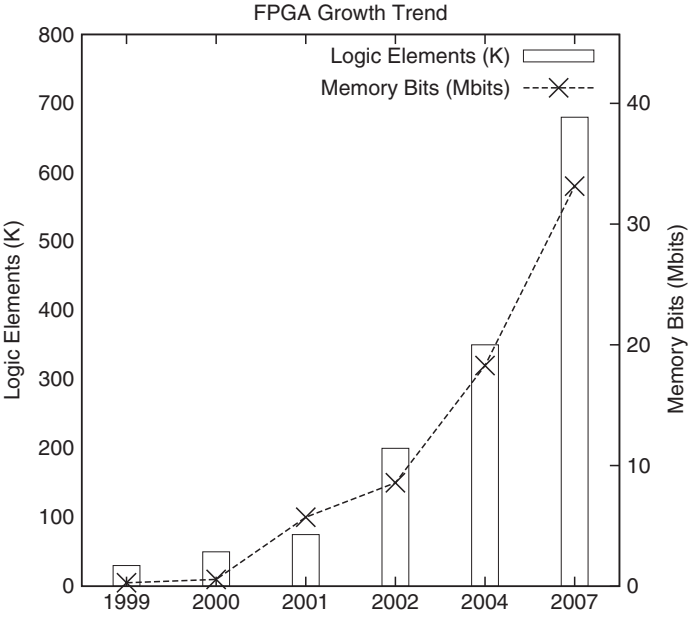

Fig. 2.6 FPGA growth trend [9]

(shown in Fig. 2.6) have made FPGAs increasingly popular. Compared to custom-designed ICs, FPGA-based designs yield lower performance, but the reconfigurable property gives it an edge over custom designs, especially since custom ICs incur significant NRE costs.

When measured in terms of power efficiency, the advantages of an FPGA-based computing strategy become even more apparent. Calculated as a function of millions of operations (MOPs) per watt, FPGAs have demonstrated greater than 1,000× power/performance advantages over today's most powerful processors [5]. For this reason, FPGA accelerators are now being deployed for a wide variety of power-hungry computing applications.

The power of the GPGPU paradigm stems from the fact that GPUs, with their large memories, large memory bandwidths, and high degrees of parallelism, are readily available as off-the-shelf devices, at very inexpensive prices. The theoretical performance of the GPU [37] has grown from 50 Gflops for the NV40 GPU in 2004 to more than 900 Gflops for GTX 280 GPU in 2008. This high computing power mainly arises due to a heavily pipelined and highly parallel architecture, with extremely high memory bandwidths. GPU memory bandwidths have grown from 42 GB/s for the ATI Radeon X1800XT to 141.7 GB/s for the NVIDIA GeForce GTX 280 GPU. In contrast, the theoretical performance of a 3 GHz Pentium4 CPU is 12 Gflops, with a memory bandwidth of 8–10 GB/s to main memory. The GPU IC is arguably one of the few VLSI platforms which has faithfully kept up with Moore's law in recent times. Recent CPU cores have 2–4 GHz core clocks, with single- and

multi-threaded performance capabilities. The Intel QuickPath Interconnect (4.8 GT/s version) copy bandwidth (using triple-channel 1,066 MHz DDR3) is 12.0 GB/s [7]. A 3.0 GHz Core 2 Quad system using dual-channel 1,066 MHz DDR3 achieves 6.9 GB/s. The level 2 and 3 caches have 10–40 cycle latencies. CPU cores today also support a limited amount of SIMD parallelism, with SEE [8] instructions.

Another key difference between GPUs and more general-purpose multi-core processors is hardware support for parallelism. GPUs have a hardware thread control unit that manages the distribution and assignment of thread blocks to multiprocessors. There is additional hardware support for synchronization within a thread block. Multi-core processors, on the other hand, depend on software and the OS to perform these tasks. However, the amount of power consumed by GPUs for executing only the accelerated portion of the computation is typically more than twice that needed by the CPU with all its peripherals. It can be argued that, since the execution is sped up, the power delay product (PDP) of a GPU-based implementation would potentially be lower. However, such a comparison is application dependent, and thus cannot be generalized.

2.9 Cost of Hardware

The non-recurring engineering (NRE) expense associated with custom IC design far exceeds that of FPGA-based hardware solutions. The large investment in custom IC development is easy to justify if the anticipated shipping volumes are large. However, many designers need custom hardware functionality for systems with low-to-medium shipping volumes. The very nature of programmable silicon eliminates the cost for fabrication and long lead times for chip assembly. Further, if system requirements change over time, the cost of making incremental changes to FPGA designs are negligible when compared to the large expense of redesigning custom ICs. The reconfigurability feature of FPGAs can add to the cost saving, based on the application. GPUs are the least expensive hardware platform for the performance they can deliver. Also, the cost of the software tool-chain required for programming GPUs is negligible compared to the EDA tool costs incurred by custom design and FPGAs.

2.10 Floating Point Operations

In comparison to software-based implementations, a higher numerical precision is a bigger problem for FPGAs and custom ICs. In FPGAs, for instance, on-chip programmable logic resources are utilized to implement floating point functionality for higher precisions [19]. These implementations consume significant die-area and tend to require deep pipelining before acceptable performance can be obtained. For example, hardware implementations of double precision multipliers typically require around 20 pipeline stages, and the square root operation requires 30–40 stages [38].

GPUs targeting scientific computations can handle IEEE double precision floating point [6, 13] while providing peak performance as high as 900 Gflops. GPUs, unlike FPGAs and custom ICs, provide native support for floating point operations.

2.11 Security and Real-Time Applications

In industry practice, design details (including HDL code) are typically documented to make reuse more convenient. At the same time, this makes IP piracy and infringement easier. It is estimated that the annual revenue loss due to IP infringement in the IC industry is in excess of $5 billion [42]. The goals of IP protection include enabling IP providers to protect their IPs against unauthorized use, protecting all types of design data used to produce and deliver IPs, and detecting and tracing the use of IPs [42].

FPGAs, because of their re-programmability, are becoming very popular for creating and exchanging VLSI IPs in the reuse-based design paradigm [27]. Existing watermarking and fingerprinting techniques embed identification information into FPGA designs to deter IP infringement. However, such methods incur timing and/or resource overheads and cause performance degradation. Custom ICs offer much better protection for intellectual property [33].

CPU/GPU software IPs have higher IP protection risks. The emerging trend is that most IP exchange and reuse will be in the form of soft IPs because of the design flexibility they provide. The IP provider may also prefer to release soft IPs and leave the customer-dependent optimization process to the users [27]. From a security point of view, protecting soft IPs is a much more challenging task than protecting hard IPs. Soft IPs are hard to trace and therefore not preferred in highly secure application scenarios.

Compared to a CPU/GPU-based implementation, FPGA and custom IC designs are truly *hard* implementations. Software-based systems like CPUs and GPUs, on the other hand, often involve several layers of abstraction to schedule tasks and share resources among multiple processors or software threads. The driver layer controls hardware resources and the operating system manages memory and processor utilization. For a given processor core, only one instruction can execute at a time, and hence processor-based systems continually run the risk of time-critical tasks pre-empting one another. FPGAs and custom ICs, which do not use operating systems, minimize these concerns with true parallel execution and dedicated hardware. As a consequence, FPGA and custom IC implementations are more suitable for applications that demand hard real-time computation guarantees.

2.12 Applications

Custom ICs are a good match for space, military, and medical compute-intensive applications, where the footprint and weight constraints are tight. Due to their high

performance, several DSP-based applications make use of custom-designed ICs. A custom IC designer can create highly efficient special functions such as arithmetic units, multi-port memories, and a variety of non-volatile storage units. Due to their cost and high performance, custom IC implementations are best suited for high-volume and high-performance applications.

Applications for FPGA are primarily hybrid software/hardware-embedded applications including DSP, video processing, robotics, radar processing, secure communications, and many others. These applications are often instances of implementing new and evolving standards, where the cost of designing custom ICs cannot be justified. Further, the performance obtained from high-end FPGAs is reasonable. In general, FPGA solutions are used for low-to-medium volume applications that do not demand extreme high performance.

GPUs are an upcoming field, but have already been used for accelerating scientific computations in fluid mechanics, image processing, and financial applications among other areas. The number of commercial products using GPUs is currently limited, but this might change due to newer architectures and high-level languages that make it easy to program the powerful hardware.

2.13 Chapter Summary

In recent times, due to the power, memory, and ILP walls, single-threaded applications do not see any significant gains in performance. Existing hardware-based accelerators such as custom-designed ICs, reconfigurable hardware such as FPGAs, and streaming processors such as GPUs are being heavily investigated as potential solutions. In this chapter we discussed these hardware platforms and pointed out several key differences among them.

In the next chapter we discuss the CUDA programming environment, used for interfacing with the GPUs. We describe the hardware, memory, and programming models for the GPU devices used in this monograph. This discussion is intended to serve as background material for the reader, to ease the explanation of the details of the GPU-based implementations of several EDA algorithms described in this monograph.

References

1. ATI CrossFire. `http://ati.amd.com/technology/crossfire/features.html`
2. ATI Stream Computing. `http://ati.amd.com/technology/streamcomputing/sdkdwnld.html`
3. CORE Generator System. `http://www.xilinx.com/products/design-tools/logic-design/design-entry/coregenerator.htm`
4. CUDA Zone. `http://www.nvidia.com/object/cuda.html`
5. FPGA-based hardware acceleration of C/C++ based applications. `http://www.pldesignline.com/howto/201800344`

6. Industry's First GPU with Double-Precision Floating Point. http://ati.amd.com/products/streamprocessor/specs.html
7. Intel Nehalem (microarchitecture). http://en.wikipedia.org/wiki/Nehalem-CPU-architecture
8. Intel SSE. http://www.tommesani.com/SSE.html
9. Mammoth FPGAs Require New Tools. http://www.gaterocket.com/device-native-verification/bid/7966/Mammoth-FPGAs-Require-New-Tools
10. NVIDIA CUDA Homepage. http://developer.nvidia.com/object/cuda.html
11. NVIDIA CUDA Introduction. http://www.beyond3d.com/content/articles/12/1
12. SLI Technology. http://www.slizone.com/page/slizone.html
13. Tesla S1070. http://www.nvidia.com/object/product-tesla-s1070-us.html
14. The Death of the Structured ASIC. http://www.chipdesignmag.com/print.php/articleId/434/issueId/16
15. Valgrind. http://valgrind.org/
16. Abdollahi, A., Fallah, F., Massoud, P.: An effective power mode transition technique in MTCMOS circuits. In: Proceedings, IEEE Design Automation Conference, pp. 13–17 (2005)
17. Bhavnagarwala, A.J., Austin, B.L., Bowman, K.A., Meindl, J.D.: A minimum total power methodology for projecting limits on CMOS GSI. IEEE Transactions Very Large Scale Integration Systems **8**(3), 235–251 (2000)
18. Bhunia, S., Banerjee, N., Chen, Q., Mahmoodi, H., Roy, K.: A novel synthesis approach for active leakage power reduction using dynamic supply gating. In: DAC '05: Proceedings of the 42nd Annual Conference on Design Automation, pp. 479–484 (2005)
19. Che, S., Li, J., Sheaffer, J., Skadron, K., Lach, J.: Accelerating compute-intensive applications with GPUs and FPGAs. In: Application Specific Processors, 2008. SASP 2008. Symposium on, pp. 101 – 107 (2008)
20. Chinnery, D.G., Keutzer, K.: Closing the power gap between ASIC and custom: An ASIC perspective. In: DAC '05: Proceedings of the 42nd Annual Design Automation Conference, pp. 275–280 (2005)
21. Chow, P., Seo, S., Rose, J., Chung, K., Paez-Monzon, G., Rahardja, I.: The design of a SRAM-based field-programmable gate array – part II : Circuit design and layout. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **7**(3), 321–330 (1999)
22. Cope, B., Cheung, P., Luk, W., Witt, S.: Have GPUs made FPGAs redundant in the field of video processing? In: Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on, pp. 111–118 (2005)
23. Fan, Z., Qiu, F., Kaufman, A., Yoakum-Stover, S.: GPU cluster for high performance computing. In: SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, p. 47 (2004)
24. Feng, Z., Li, P.: Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms. In: ICCAD '08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, pp. 647–654. IEEE Press, Piscataway, NJ (2008)
25. Gao, F., Hayes, J.: Exact and heuristic approaches to input vector control for leakage power reduction. In: Proceedings, International Conference on Computer-Aided Design, pp. 527–532 (2004)
26. Graham, P., Nelson, B., Hutchings, B.: Instrumenting bitstreams for debugging FPGA circuits. In: FCCM '01: Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 41–50 (2001)
27. Jain, A.K., Yuan, L., Pari, P.R., Qu, G.: Zero overhead watermarking technique for FPGA designs. In: GLSVLSI '03: Proceedings of the 13th ACM Great Lakes symposium on VLSI, pp. 147–152 (2003)
28. Kuon, I., Rose, J.: Measuring the gap between FPGAs and ASICs. In: FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, pp. 21–30 (2006)

29. Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., Segal, M., Papakipos, M., Buck, I.: GPGPU: General-purpose computation on graphics hardware. In: SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, p. 208 (2006)
30. Mal, P., Cantin, J., Beyette, F.: The circuit designs of an SRAM based look-up table for high performance FPGA architecture. In: 45th Midwest Symposium on Circuits and Systems (MWCAS), vol. III, pp. 227–230 (2002)
31. Minana, G., Garnica, O., Hidalgo, J.I., Lanchares, J., Colmenar, J.M.: A power-aware technique for functional units in high-performance processors. In: DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design, pp. 456–459 (2006)
32. Molas, G., Bocquet, M., Buckley, J., Grampeix, H., Gély, M., Colonna, J.P., Martin, F., Brianceau, P., Vidal, V., Bongiorno, C., Lombardo, S., Pananakakis, G., Ghibaudo, G., De Salvo, B., Deleonibus, S.: Evaluation of HfAlO high-k materials for control dielectric applications in non-volatile memories. Microelectronic Engineering **85**(12), 2393–2399 (2008)
33. Oliveira, A.L.: Robust techniques for watermarking sequential circuit designs. In: DAC '99: Proceedings of the 36th ACM/IEEE Conference on Design Automation, pp. 837–842 (1999)
34. Owens, J.: GPU architecture overview. In: SIGGRAPH '07: ACM SIGGRAPH 2007 Courses, p. 2 (2007)
35. Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Philips, J.C.: GPU Computing. In: Proceedings of the IEEE, vol. 96, pp. 879–899 (2008)
36. Raja, T., Agrawal, V.D., Bushnell, M.L.: CMOS circuit design for minimum dynamic power and highest speed. In: VLSID '04: Proceedings of the 17th International Conference on VLSI Design, p. 1035. IEEE Computer Society, Washington, DC (2004)
37. Schive, H.Y., Chien, C.H., Wong, S.K., Tsai, Y.C., Chiueh, T.: Graphic-card cluster for astrophysics (GraCCA) – performance tests. In: Submitted to NewAstronomy (2007)
38. Scrofano, R., G.Govindu, Prasanna, V.: A library of parameterizable floating point cores for FPGAs and their application to scientific computing. In: Proceedings of the 2005 International Conference on Engineering of Reconfigurable Systems and Algorithms, pp. 137–148 (2005)
39. Wei, L., Chen, Z., Johnson, M., Roy, K., De, V.: Design and optimization of low voltage high performance dual threshold CMOS circuits. In: DAC '98: Proceedings of the 35th Annual Conference on Design Automation, pp. 489–494 (1998)
40. Yu, B., Bushnell, M.L.: A novel dynamic power cutoff technique DPCT for active leakage reduction in deep submicron CMOS circuits. In: ISLPED '06: Proceedings of the 2006 International Symposium on Low Power Electronics and Design, pp. 214–219 (2006)
41. Yuan, L., Qu, G.: Enhanced leakage reduction technique by gate replacement. In: DAC, pp. 47–50 (2005)
42. Yuan, L., Qu, G., Ghout, L., Bouridane, A.: VLSI design IP protection: solutions, new challenges, and opportunities. In: AHS '06: Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems, pp. 469–476 (2006)