Open-Source Operating System Support for Information Security with FPGA Platforms

Chen-Mou Cheng Sheng-De Wang
{ccheng, sdwang}@cc.ee.ntu.edu.tw
Dept. Electrical Engineering
National Taiwan University

Bo-Yin Yang
byyang@iis.sinica.edu.tw
Inst. Information Science
Academia Sinica

September 21, 2007

Abstract

We propose to apply the open-source software development model to the domain of hardware/software codesign. Specifically, we propose to build around the GNU/Linux operating system a POSIX-thread compatible multi-threaded programming model for the reconfigurable CPU/FPGA hybrid platform. Under this programming model, hardware and software computation tasks will be able to communicate efficiently while executing in parallel. Also, the computations carried out by software and hardware are treated in a homogeneous way and thus interchangeable.

Our target application domain is information security. Specifically, we will implement two applications to demonstrate the effectiveness of this new framework. The first one is the acceleration of packet classification in wire-speed firewall. The second one is the implementation of a family of emergent multivariate public-key cryptosystems, called the Tame Transformation Signature (TTS). We hope the synergy between the two chosen applications and the proposed programming model will bring fruitful results to both hardware/software codesign and information security communities.

1 Background Information and Project Goals

1.1 Hardware/Software Codesign and Reconfigurable Computing

Hardware/software codesign has emerged as a mainstream design paradigm since it was first conceived over a decade ago [1, 2, 3]. With the wide spread of microprocessor-based designs, people have come to realize the benefits of incorporating general-purpose microprocessors in digital circuit designs. For one, this tends to notably reduce circuit design complexity by moving some of the logic to software running on microprocessors. Another benefit is the ability to take advantage of a wide selection of commercial, off-the-shelf (COTS) microprocessor designs, some of which have evolved for a long period of time and hence have been highly optimized for a variety of common computational tasks. As a result, these COTS components often enjoy a significantly lower cost-performance ratio due to economy of scale [4]. With the emergence of complex and high-performance field-programmable gate array (FPGA) devices, this eventually leads to the proliferation of reconfigurable computers [5].

In order to fully realize the potential of the new hardware/software codesign paradigm, software will need to be treated as a first-class component in the design activities [2]. However, after more than a decade of research and practice, this is hardly the case. The emphasis is still largely on hardware, despite the pressing and urgent demand for more mature software development tools and better programming models from the participating developers and engineers of these design activities. This is especially true in Taiwan's circuit design industry, where the majority of revenue comes from sales of tangible hardware products. As a result, there is a serious lack of mature programming models and middleware, let alone high-quality operating systems, in "software" part of the design and development activities of the codesign paradigm [6, 7].

Lately, academic researchers have looked into this problem. For example, Niehaus et al. argued that the lack of an integrated view has hindered the developers from fully realizing the promises brought about by the marriage of CPU and FPGA in reconfigurable computing [6, 4, 8]. They proposed to use the multi-threaded computation model as the unifying framework for hardware/software codesign activities and implemented as an extension to a real-time, embedded Linux kernel. Later on, Vuletić et al. proposed to implement such a multi-threaded programming model via a virtualization interface at the virtual memory (VM) layer in the Linux 2.4 kernel [9, 10]. More recently, So, Tkachenko, and Brodersen proposed to abstract hardware computation tasks as processes on a modified Linux 2.4 kernel to facilitate close interaction and multi-tasking between hardware and software components in a hybrid CPU/FPGA system [11, 12]. All these independent efforts are in line with our proposal and have attained some initial success in the application realms they have chosen. Unfortunately, they did not receive a wide appreciation in the community and did not have enough impact on the mainstream practice. As it will become clear later on, one of the major goals of this proposal is to investigate this new multi-threaded programming model proposed by these prominent researchers in the field. By integrating all these research and development efforts and applying them to an eminent field of application, we hope that we will be able to promote the role played by software in the hardware/software codesign community.

1.2 Open-Source Software

To promote the role played by software in the hardware/software codesign paradigm, perhaps we can borrow a lesson from the open-source software movement. The open-source software movement has successfully created a community and an ecosystem for all sorts of players to take part and generate revenues. It also establishes "convergence points" around which standards are built and efforts are focused. The GNU/Linux operating system is a good example of such a convergence point. With the success of GNU/Linux, application developers benefit tremendously from a stable, feature-rich, yet still actively evolving set of application programming interfaces (API). With this set of API, various developers can build their works on top of others' and do not need to repeatedly reinvent the wheels. It is also of great merit to have a central repository in which innovation and recognition can accumulate; such an invaluable "brand name" would otherwise take an astronomical amount of resources to build. Another less well-known example is the GNURadio project [13], which has spawned an entire array of new research and development efforts across academia and industry on software-defined radio [14]. Yet another famous example from the hardware community is the OpenCores project [15], from which we have witnessed the power of sharing and how that power can stimulate rapid growth of interest and activity in a community. As we have noted in the previ-

ous section, there have been quite some efforts, albeit scattered, in investigating efficient programming models that make software a first-class component. We argue that the open-source way provides a viable answer: by setting up an open-source framework, more developers from both hardware and software community will be attracted to build their works on top of this framework, which will in turn ignite more design and refinement activities on searching for better programming models for hardware/software codesign, completing a positive feedback cycle.

1.3 Driving Applications in Information Security

We will use two types of applications to drive the design exploration in this project. The first type is drawn from public-key cryptography. It is well known that many public-key cryptosystems (PKCs) can benefit from special hardware for accelerating the involved algebraic operations. Here we will look at such an application from a different angle, namely, how the new programming model can help development in a rapidly growing field where there is great uncertainty. That is, we would like to implement a family of analogous yet still evolving PKCs with hardware/software codesign. The algorithmic partitioning and design reuse will play an important role in this application. The second type of application is from network security. This has also been a major source of hardware acceleration works due to the demand of high speed and high throughput in modern networking equipment.

1.3.1 Emergent Public-Key Cryptosystems

Traditionally, public-key cryptography has been an important application realm for hardware/software codesign due to its stringent computational requirements. There have been quite some efforts in implementing mature or near-mature PKCs [16, 17, 18, 19, 20]. We feel that it is also important to pay special attention to emergent PKCs, for the experience from implementing these PKCs can provide insightful feedback on the performance—sometimes even the security—of these new PKCs to their respective design cryptologists, not to mention the head-start we would have once they become mainstream standards.

We are especially interested in several families of emergent PKCs, collectively known as the multivariate PKCs. They are alternatives to traditional PKCs, such as the RSA scheme and the Elliptic Curve Cryptography (ECC), that are mostly based on large algebraic structures. The new multivariate PKCs usually execute much faster than traditional PKCs on the same hardware. More importantly, they represent a future-proof investment against many new cryptanalysis techniques [21, 22, 23]. We have obtained some initial success along this research direction. We have investigated ways to save die space and energy consumption in hardware via an ASIC implementation of the Tame Transformation Signature (TTS) scheme on a TSMC 0.25μm process. The current consumption of TTS is only 21 μA for computing a signature, using 22,000 gate equivalents and 16,000 100-kHz cycles (160 milliseconds). Figure 1 shows a block diagram of our ASIC implementation.

Currently, multivariate PKCs have not received enough attention in applications due to their novelty. As a low-resource alternative and a hedge against the rising quantum computers, prominent cryptologists are calling for more efforts on investigating the security and applicability of these new multivariate PKCs [21, 22, 23]. However, without a reasonably optimized hardware implementation, it is difficult—if possible at all—to assess the effectiveness of some of the cryptanalysis techniques, e.g., the differential power analysis [24]. Moreover,

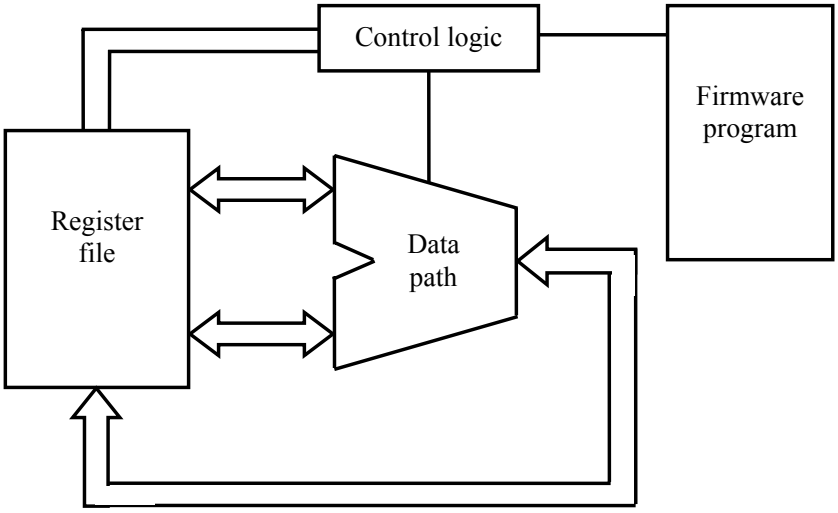

Figure 1: The block diagram of our ASIC implementation of the TTS scheme.

implementing these new multivariate PKCs presents a unique challenge to hardware/software codesign because of the constant evolution of these systems. A good hardware/software codesign model should allow rapid adaption when security patches become available from the design cryptologists.

1.3.2 Network Security

Another arena where a great amount of hardware/software codesign activities take place is network security. The base form of the constituting components in network security is the firewall, a machine that inspects packets and determines what to do with them based on a set of predefined or dynamically generated rules, preferably at wire speed. Because of the high throughput requirement put forth by modern network equipment, it has stimulated ample innovation and development in hardware/software codesign.

We will focus on one of the most important techniques in building a high-speed firewall, namely, the hardware/software codesign of packet classification for netfilter/iptables in the Linux kernel. The netfiter/iptables project is a firewalling subsystem in Linux 2.4 and 2.6 kernels. It provides the firewall and many other derivative functionalities, including stateless and stateful packet filtering, a wide variety of network address translation, and packet mangling [25].

The major part of netfilter/iptables, the netfilter framework, is a set of hooks inside Linux kernel's network protocol stack. It allows kernel modules to register callback functions at these hooks. These kernel callback functions can in turn perform packet filtering, network address translation, and other packet mangling on packets that match certain patterns. The iptables, on the other hand, is a table of rulesets. A rule in a ruleset specifies what action to take once a packet is matched against a particular pattern. A userland program, iptables(8), can be used to inspect, create, and modify these rulesets.

The rulesets are organized as tables, and each table contains a number of built-in chains and may contain user-defined chains. Each chain is a list of rules, with each rule specifying what to do with a packet that matches the pattern specified in that rule. In its own terminology, this is a target, which may be a standard action such as ACCEPT, DROP, QUEUE, or

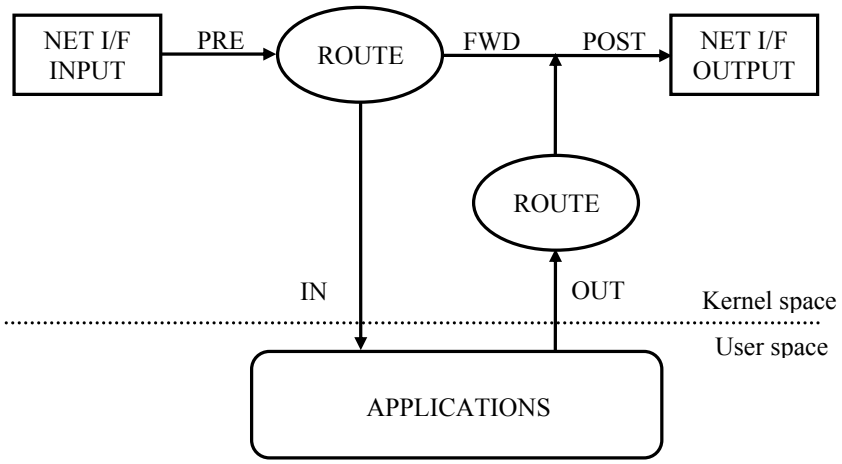

Figure 2: The block diagram of the netfilter/iptables architecture.

RETURN, or a jump to a user-defined chain in the same table. For IPv4, there are five hooks defined in netfilter: PREROUTING, INPUT, FORWARD, POSTROUTING, and OUTPUT. The traversal of these hooks inside the kernel protocol stack are described in detail in section 3 of the Linux netfilter Hacking HOWTO [26]. Figure 2 depicts the block diagram of the netfilter architecture.

We have attained some initial success along this direction. In a previous NSC project, we have built a hardware accelerator for packet classification in netfilter/iptables. We have successfully offloaded packet classification on a firewall onto an FPGA-based network coprocessor for the host processor in a high-speed networking environment. We have verified that our network coprocessor can significantly speed up the packet processing in a prototype firewall system based on the Xilinx Virtex 4 FX FPGA development platform.

1.4 Project Goals

In this project, we aim to promote the role played by software in the hardware/software codesign paradigm so that software is indeed treated as a first-class component, and the potential promised by the paradigm can be fully realized. We hope to achieve this goal via reinforcing the habit of sharing intellectual output to the hardware/software codesign community, and hopefully we can replicate the success experience of many open-source software projects.

We hope to validate the effectiveness of the multi-threaded programming model proposed by the prominent researchers in the field mentioned previously. We will demonstrate this using two applications: packet classification in network security, which demands data high throughput, as well as several emergent public-key cryptosystems, which demand an expressive and efficient programming model. We hope to demonstrate that the flexibility afforded by the multi-threaded programming model enables rapid prototyping of the new schemes.

We also hope to help emergent PKCs gain more recognition and attention by providing high-quality, open-source implementations for performance and security analysis. We hope the close synergy between the two chosen applications and the programming model under investigation will bring fruitful results to both sides.

We summarize below the high-level goals of this project.

1. Verify the effectiveness of the multi-threaded programming model as a unifying frame-

work of hardware/software codesign using applications from information security.

2. Promote the role played by software in hardware/software codesign by reinforcing the spirit of open-source in the community.

3. Help emergent multivariate PKCs gain momentum through the synergy with open-source hardware/software codesign.

2 Proposed Approaches

We will first investigate and compare three existing proposals and implementations, labeled as Method A [6, 4, 8], B [9, 10], and C [11, 12]. We hope that we can reuse these designs as much as possible in designing and implementing a POSIX-thread compatible multi-threaded programming model suitable for information security applications. Under this programming model, the computations carried out by software and hardware are treated in a homogeneous way and thus interchangeable. That is, computation can be migrated from software to hardware or vice versa in a seamless manner. This will enable system designers to roam freely across the hardware/software boundary, as well as to explore more efficient different hardware/software algorithmic partitioning strategies, not to mention that the familiar programming model will and attract more developers and enable more rapid development cycles. This programming model will also enable parallel execution of hardware coprocessors, as well as close and efficient collaboration between hardware and software components. Figure 3 depicts the overall system architecture for the proposed framework.

We will use two applications to demonstrate the power of this new framework. The first one, labeled as Application A, is the acceleration of the netfilter/iptables inside the Linux kernel. We will validate the effectiveness by inspecting the obtained speedup with the framework and compare it with the result obtained by hand-crafted hardware/software codesign from the literature. The second application we want to demonstrate, labeled as Application B, is a hybrid hardware/software implementation of a family of cutting-edge multivariate PKCs, the TTS schemes [21]. The members in the family share similar traits and thus have similar requirements when it comes to implementation, yet their detailed algorithmic features exhibit a large degree of variation, which will put forth a harsh test for the flexibility and expressiveness of the new programming model.

The relationship between the major tasks of this project, as well as the respective investigators in charge, is depicted in Figure 4.

3 Deliverables

We list the major deliverables of this project below. We plan to complete items 1 and 2 before the end of first year, while deliver items 3 and 4 at the end of second year. Items 3 and 4 will be delivered on an appropriate FPGA platform, whose architecture is depicted in Figure 5.

1. Specification of a POSIX-thread compatible framework for hardware/software codesign

2. Implementation, testing, and performance evaluation of the proposed framework

3. Hardware-accelerated netfilter/iptables using the proposed framework

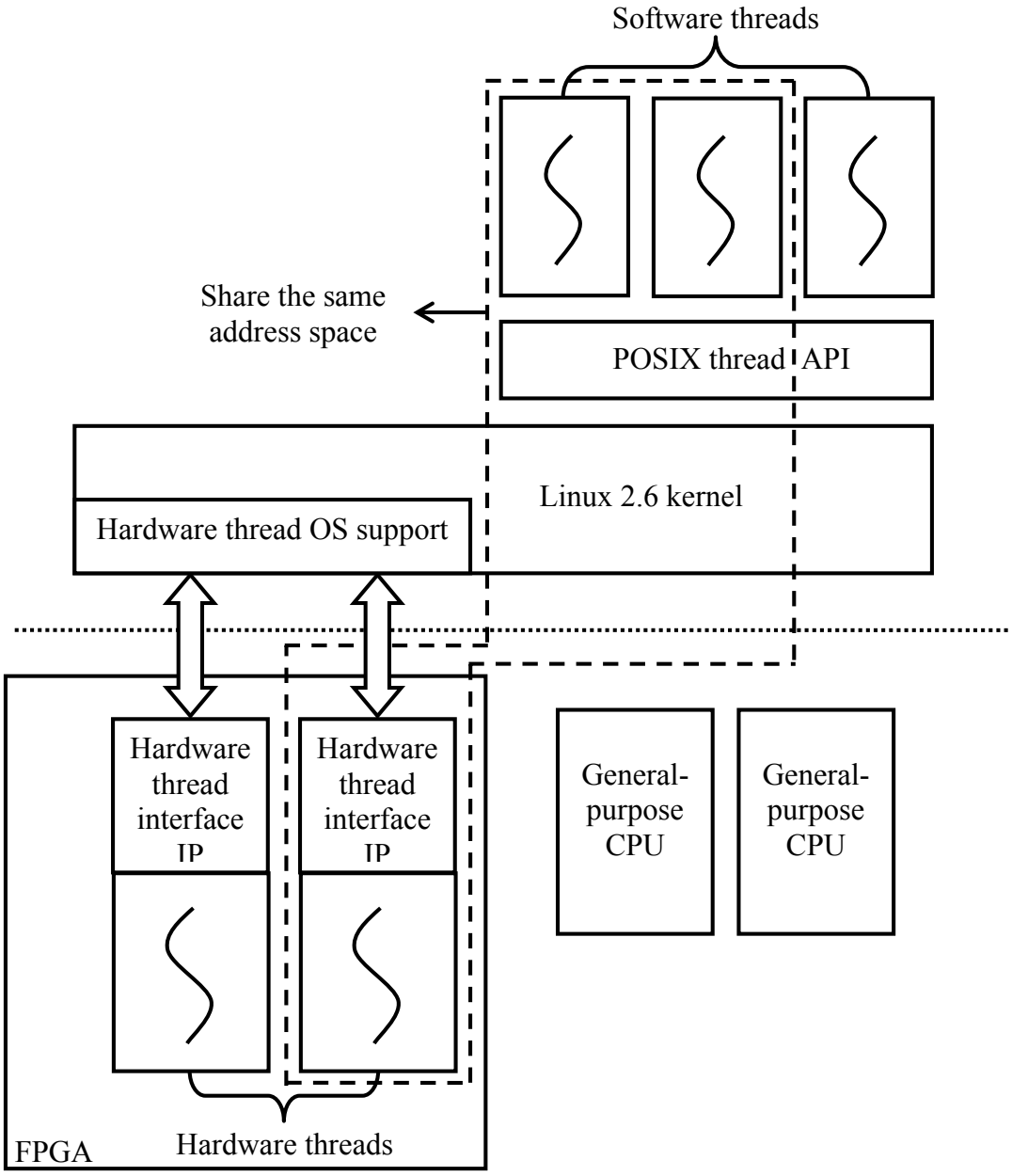

Figure 3: The architecture of the POSIX-thread compatible programming framework.

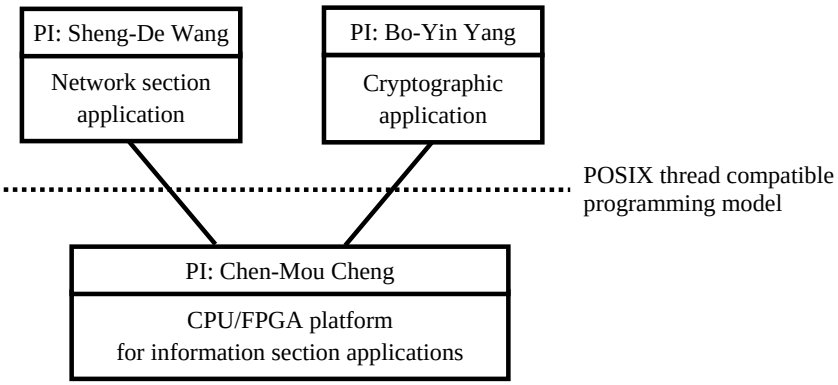

Figure 4: The major tasks of this project and their corresponding investigators in charge.

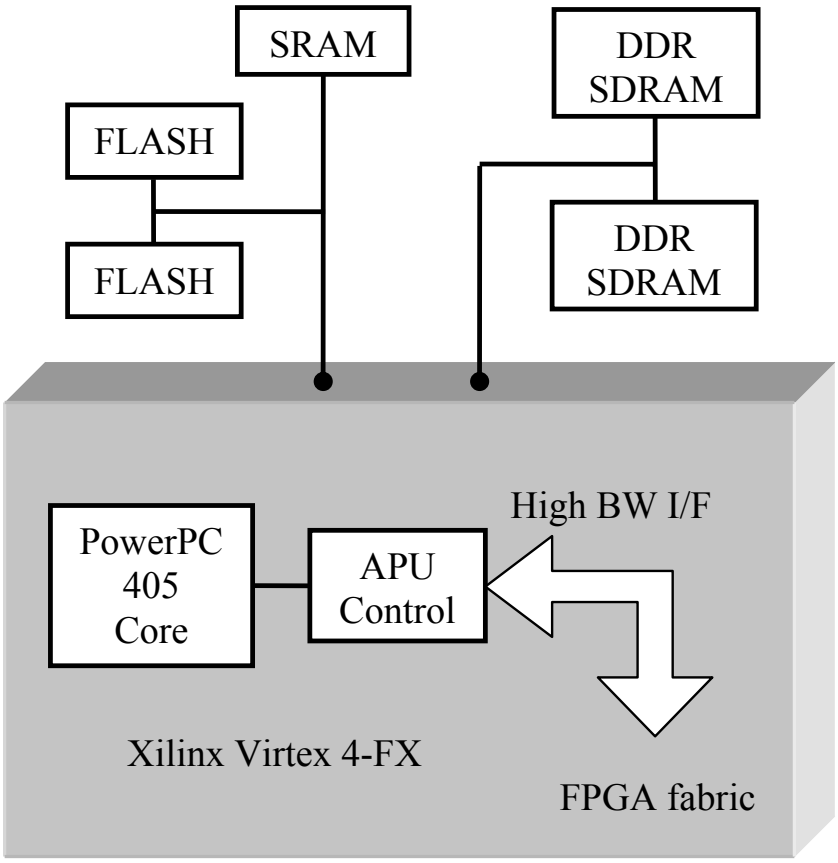

Figure 5: The hardware architecture of the demo FPGA platform.

4. Hybrid implementation of the TTS family of PKCs using the proposed framework

We also include a Gantt chart that illustrates the scheduled project progress in Figure 6.

References

[1] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys*, vol. 34, pp. 171–210, June 2002.

[2] W. Wolf, "A decade of hardware/software codesign," *IEEE Computer*, vol. 36, pp. 38–43, April 2003.

[3] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, pp. 1–19, 2006.

[4] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid FPGA-CPU computational components: A missing link," *IEEE Micro*, vol. 24, pp. 42–53, July/August 2004.

[5] S. Brown and J. Rose, "FPGA and CPLD architectures: A tutorial," *IEEE Design & Test of Computers*, vol. 13, pp. 42–57, Summer 1996.

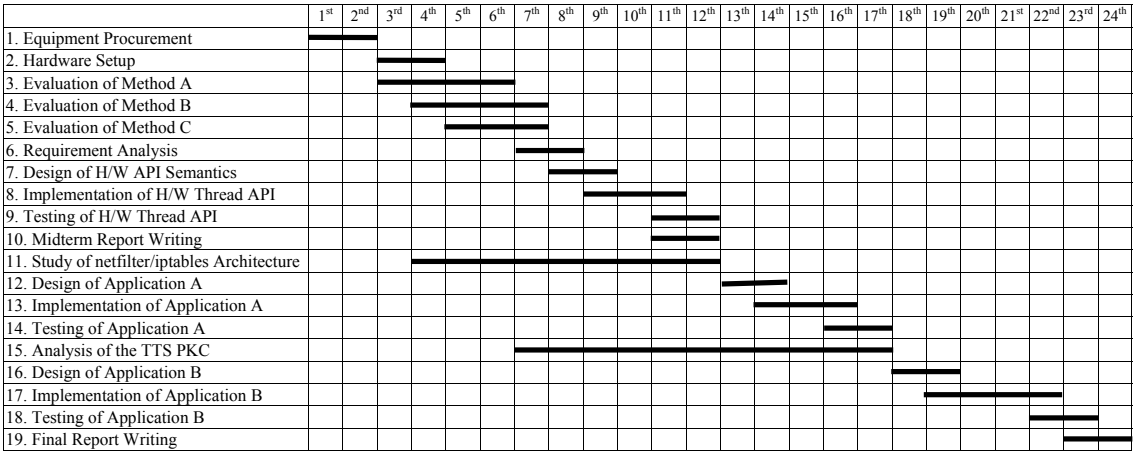

Figure 6: The scheduled project progress.

[6] D. Niehaus and D. Andrews, "Using the multi-threaded computation model as a unifying framework for hardware-software co-design and implementation," in *Proceedings of the 9th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'03F)*, (Anacapri, Italy), October 2003.

[7] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proceedings of the 41st Design Automation Conference (DAC 2004)*, (San Diego, CA, USA), June 2004.

[8] R. Jidin, *Extending the thread programming model across CPU and FPGA hybrid architectures*. PhD thesis, University of Kansas, Lawrence, KS, USA, April 2005.

[9] M. Vuletić, L. Righetti, L. Pozzi, and P. Ienne, "Operating system support for interface virtualization of reconfigurable coprocessors," in *Proceedings of the 2004 Design, Automation and Test in Europe Conference & Exhibition (DATE 04)*, (Paris, France), February 2004.

[10] M. Vuletić, L. Pozzi, and P. Ienne, "Seamless hardware-software integration in reconfigurable computing systems," *IEEE Design & Test of Computers*, vol. 22, pp. 102–113, March/April 2005.

[11] H. K.-H. So and R. W. Brodersen, "Improving usability of FPGA-based reconfigurable computers through operating system support," in *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL 2006)*, (Madrid, Spain), August 2006.

[12] H. K.-H. So, A. Tkachenko, and R. W. Brodersen, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH," in *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2006)*, (Seoul, Korea), October 2006.

[13] `http://www.gnu.org/software/gnuradio/`.

[14] B.-R. Chen, C.-M. Cheng, and T.-Y. Lin, "BoboLink: Software defined radio on personal computers," tech. rep., Harvard University, Cambridge, MA, USA, May 2005.

[15] http://www.opencores.org/.

[16] M. Šimka, V. Fischer, and M. Drutarovský, "Hardware-software codesign in embedded asymmetric crytpgraphy application—a case study," in *Proceedings of the 13th International Conference on Field Programmable Logic and Application (FPL 2003)*, (Lissabon, Portugal), September 2003.

[17] N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi, "Implementation of elliptic curve cryptosystems on a reconfigurable computer," in *Proceedings of the 2003 IEEE International Conference on Field-Programmable Technology (FPT'03)*, (Tokyo, Japan), December 2003.

[18] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede, "Hardware/software co-design for hyperelliptic curve cryptography (HECC) on the 8051 μP," in *Proceedings of the 2005 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, (Edinburgh, Scotland), August 2005.

[19] J. Großschädl, P. Ienne, L. Pozzi, S. Tillich, and A. K. Verma, "Combining algorithm exploration with instruction set design: A case study in elliptic curve cryptography," in *Proceedings of the 2006 Design, Automation and Test in Europe Conference & Exhibition (DATE 06)*, (Munich, Germany), March 2006.

[20] P. Schaumont and I. Verbauwhede, "Hardware/software codesign for stream ciphers," in *The State of the Art of Stream Ciphers, a Special Workshop hosted by the ECRYPT Network of Excellence in Cryptology (SASC 2007)*, (Bochum, Germany), January 2007.

[21] B.-Y. Yang, C.-M. Cheng, B.-R. Chen, and J.-M. Chen, "Implementing minimized multivariate public-key cryptosystems on low-resource embedded systems," in *Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC 2006)*, (York, UK), April 2006.

[22] B.-Y. Yang and J.-M. Chen, "Building secure tame-like multivariate public-key cryptosystems: The new TTS," in *Proceedings of the 10th Australasian Conference on Information Security and Privacy (ACISP'05)*, (Brisbane, Australia), July 2005.

[23] B.-Y. Yang, J.-M. Chen, and Y.-H. Chen, "TTS: High-speed signatures on a low-cost smart card," in *Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, (Boston, MA, USA), August 2004.

[24] P. Kosher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (Crypto'99)*, (Santa Barbara, CA, USA), August 1999.

[25] http://www.netfilter.org/.

[26] http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html.

[27] P. Yiannacouras, J. G. Steffan, and J. Rose, "Application-specific customization of soft processor microarchitecture," in *Proceedings of the 14th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2006)*, (Monterey, CA, USA), February 2006.

[28] J. M. Arnold, "S5: The architecture and development flow of a software configurable processor," in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology (FPT'05)*, (Singapore), December 2005.

[29] R. E. Gonzalez, "Xtensa: A configurable and extensible processor," *IEEE Micro*, vol. 20, pp. 60–70, March/April 2000.

A The Complete List of POSIX Thread API

1. `pthread_atfork`: Register Fork Handlers
2. `pthread_atfork_np`: Register Fork Handlers with Extended Options
3. `pthread_attr_destroy`: Destroy Thread Attributes Object
4. `pthread_attr_getdetachstate`: Get Thread Attributes Object Detachstate
5. `pthread_attr_getinheritsched`: Get Thread Attribute Object Inherit Scheduling Attributes
6. `pthread_attr_getschedparam`: Get Thread Attributes Object Scheduling Parameters
7. `pthread_attr_getschedpolicy`: Get Scheduling Policy
8. `pthread_attr_getscope`: Get Scheduling Scope
9. `pthread_attr_getstackaddr`: Get Stack Address
10. `pthread_attr_getstacksize`: Get Stack Size
11. `pthread_attr_init`: Initialize Thread Attributes Object
12. `pthread_attr_setdetachstate`: Set Thread Attributes Object Detachstate
13. `pthread_attr_setinheritsched`: Set Thread Attribute Inherit Scheduling Attributes
14. `pthread_attr_setschedparam`: Set Thread Attributes Object Scheduling Parameters
15. `pthread_attr_setschedpolicy`: Set Scheduling Policy
16. `pthread_attr_setscope`: Set Scheduling Scope
17. `pthread_attr_setstackaddr`: Set Stack Address
18. `pthread_attr_setstacksize`: Set Stack Size
19. `pthread_cancel`: Cancel Thread
20. `pthread_cleanup_peek_np`: Copy Cleanup Handler from Cancellation Cleanup Stack
21. `pthread_cleanup_pop`: Pop Cleanup Handler off of Cancellation Cleanup Stack
22. `pthread_cleanup_push`: Push Cleanup Handler onto Cancellation Cleanup Stack
23. `pthread_clear_exit_np`: Clear Exit Status of Thread
24. `pthread_cond_broadcast`: Broadcast Condition to All Waiting Threads
25. `pthread_cond_destroy`: Destroy Condition Variable
26. `pthread_cond_init`: Initialize Condition Variable
27. `pthread_cond_signal`: Signal Condition to One Waiting Thread
28. `pthread_cond_timedwait`: Timed Wait for Condition
29. `pthread_cond_wait`: Wait for Condition
30. `pthread_condattr_destroy`: Destroy Condition Variable Attributes Object
31. `pthread_condattr_init`: Initialize Condition Variable Attributes Object

32. pthread_condattr_getpshared: Get Process Shared Attribute from Condition Attributes Object
33. pthread_condattr_setpshared: Set Process Shared Attribute in Condition Attributes Object
34. pthread_create: Create Thread
35. pthread_delay_np: Delay Thread for Requested Interval
36. pthread_detach: Detach Thread
37. pthread_equal: Compare Two Threads
38. pthread_exit: Terminate Calling Thread
39. pthread_extendedjoin_np: Wait for Thread with Extended Options
40. pthread_get_expiration_np: Get Condition Expiration Time from Relative Time
41. pthread_getcancelstate_np: Get Cancel State
42. pthread_getconcurrency: Get Process Concurrency Level
43. pthread_getpthreadoption_np: Get Pthread Run-Time Option Data
44. pthread_getschedparam: Get Thread Scheduling Parameters
45. pthread_getspecific: Get Thread Local Storage Value by Key
46. pthread_getthreadid_np: Retrieve Unique ID for Calling Thread
47. pthread_getunique_np: Retrieve Unique ID for Target Thread
48. pthread_is_initialthread_np: Check if Running in the Initial Thread
49. pthread_is_multithreaded_np: Check Current Number of Threads
50. pthread_join: Wait for and Detach Thread
51. pthread_join_np: Wait for Thread to End
52. pthread_key_create: Create Thread Local Storage Key
53. pthread_key_delete: Delete Thread Local Storage Key
54. pthread_kill: Send Signal to Thread
55. pthread_lock_global_np: Lock Global Mutex
56. pthread_mutex_destroy: Destroy Mutex
57. pthread_mutex_getprioceiling: Get Mutex Priority Ceiling
58. pthread_mutex_init: Initialize Mutex
59. pthread_mutex_lock: Lock Mutex
60. pthread_mutex_setprioceiling: Set Mutex Priority Ceiling
61. pthread_mutex_timedlock_np: Lock Mutex with Time-Out
62. pthread_mutex_trylock: Lock Mutex with No Wait
63. pthread_mutex_unlock: Unlock Mutex
64. pthread_mutexattr_destroy: Destroy Mutex Attributes Object
65. pthread_mutexattr_getkind_np: Get Mutex Kind Attribute
66. pthread_mutexattr_getname_np: Get Name from Mutex Attributes Object
67. pthread_mutexattr_getprioceiling: Get Mutex Priority Ceiling Attribute
68. pthread_mutexattr_getprotocol: Get Mutex Protocol Attribute
69. pthread_mutexattr_getpshared: Get Process Shared Attribute from Mutex Attributes Object

70. pthread_mutexattr_gettype: Get Mutex Type Attribute
71. pthread_mutexattr_init: Initialize Mutex Attributes Object
72. pthread_mutexattr_setkind_np: Get Mutex Kind Attribute
73. pthread_mutexattr_setname_np: Set Name in Mutex Attributes Object
74. pthread_mutexattr_setprioceiling: Set Mutex Priority Ceiling Attribute
75. pthread_mutexattr_setprotocol: Set Mutex Protocol Attribute
76. pthread_mutexattr_setpshared: Set Process Shared Attribute in Mutex Attributes Object
77. pthread_mutexattr_settype: Set Mutex Type Attribute
78. pthread_once: Perform One-Time Initialization
79. pthread_rwlock_destroy: Destroy Read/Write Lock
80. pthread_rwlock_init: Initialize Read/Write Lock
81. pthread_rwlock_rdlock: Get Shared Read Lock
82. pthread_rwlock_timedrdlock_np: Get Shared Read Lock with Time-Out
83. pthread_rwlock_timedwrlock_np: Get Exclusive Write Lock with Time-Out
84. pthread_rwlock_tryrdlock: Get Shared Read Lock with No Wait
85. pthread_rwlock_trywrlock: Get Exclusive Write Lock with No Wait
86. pthread_rwlock_unlock: Unlock Exclusive Write or Shared Read Lock
87. pthread_rwlock_wrlock: Get Exclusive Write Lock
88. pthread_rwlockattr_destroy: Destroy Read/Write Lock Attribute
89. pthread_rwlockattr_getpshared: Get Pshared Read/Write Lock Attribute
90. pthread_rwlockattr_init: Initialize Read/Write Lock Attribute
91. pthread_rwlockattr_setpshared: Set Pshared Read/Write Lock Attribute
92. pthread_self: Get Pthread Handle
93. pthread_set_mutexattr_default_np: Set Default Mutex Attributes Object Kind Attribute
94. pthread_setcancelstate: Set Cancel State
95. pthread_setcanceltype: Set Cancel Type
96. pthread_setconcurrency: Set Process Concurrency Level
97. pthread_setpthreadoption_np: Set Pthread Run-Time Option Data
98. pthread_setschedparam: Set Target Thread Scheduling Parameters
99. pthread_setspecific: Set Thread Local Storage by Key
100. pthread_sigmask: Set or Get Signal Mask
101. pthread_signal_to_cancel_np: Convert Signals to Cancel Requests
102. pthread_test_exit_np: Test Thread Exit Status
103. pthread_testcancel: Create Cancellation Point
104. pthread_trace_init_np: Initialize or Reinitialize Pthread Tracing
105. PTHREAD_TRACE_NP: Macro to optionally execute code based on trace level
106. pthread_unlock_global_np: Unlock Global Mutex
107. sched_yield: Yield Processor to Another Thread