Windows CE for a Reconfigurable System-on-a-Chip Processor[*]

M. R. George, and W. F. Wong
School of Computing,
National University of Singapore,
3 Science Drive 2, Singapore 117543
Singapore
wongwf@comp.nus.edu.sg

Abstract

Reconfigurable System-on-a-Chip (RSoC) processors promise a low cost and rapid means of prototyping complex systems of integrated software and hardware, especially for embedded applications. Due to need to manage a wide diversity of resources on-chip, an embedded operating system is needed. In this paper, we will report on our experience in porting an industrial-strength embedded operating system, namely Microsoft Windows CE .NET®, on a state-of-the-art RSoC processor platform, namely the Altera Excalibur®. We will report performance evaluation using a set of kernel micro-benchmarks. We will also illustrate how the reconfigurable hardware resource, the programmable logic device *(PLD), can be used in an application running under Windows CE.*

1 Introduction

Reconfigurable System-on-a-Chip (RSoC) processors integrate an industrial standard, full instruction set architecture microprocessor, peripherals, and reprogrammable logic in a single chip. Such a sophisticated yet generic platform opens up many interesting new possibilities. This includes the rapid prototyping of complex hardware - software solutions, as well as remote upgrading and bug fixes for systems already deployed [8]. The diversity of hardware resources available on-chip necessitates an operating system. We have ported Microsoft's Windows CE .NET to Altera's most recent Excalibur board, the EPXA10. As our experience shows, the advantages of an operating system (OS) on the target board are multiple. The characteristics of the memory system are abstracted away from the programmer. Thereby application programs written on one platform is portable to the other. The nuances of configuring the reprogrammable hardware, the *programming logic device* (PLD), are delegated to a device driver. All that a programmer has to do to dynamically reconfigure the PLD, is to provide the configuration data file. Moreover the programmer is able to utilize the rich environment provided by the operating system which will significantly increase the ease of use.

The main contribution of this paper is to describe the port along with some initial performance evaluation of this port. Using examples, we will also show how the PLD can be used in support of applications. Although we just found out that there was another parallel effort in porting Windows CE to the Altera Excalibur albeit using a different board [15], this paper presents the first set of detailed evaluation of such a port along with evidence of the efficacy of using the PLD.

The rest of the paper is organized as follows. Section 2 gives a brief description of WinCE architecture, Section 3 highlights the characteristics of Excalibur family. Section 4 gives some details regarding our port. In Section 5, we evaluate the performance of the kernel by means of micro-benchmarks. Section 6 and 7 describe how we use the PLD of the Excalibur as well as evaluate its effectiveness by means of a hardware-software benchmark. This is followed by a conclusion.

2 Windows CE Architecture

Windows CE is Microsoft's solution to embedded operating systems. Unlike Windows XP embedded, which is a componentized version of Windows XP professional, Windows CE is designed from the ground up for the embedded marketplace. It was introduced in the Handheld PC range of products in November 1996.

Windows CE has a hierarchical architecture with the boot loader, OAL (OEM adaptation layer) and device drivers forming the lowest layer. The kernel, GWES (Graphics, Windowing and Events Subsystem) and the communication stacks form the next

[*]This work is funded by Microsoft Research Contract Number 2003-308.

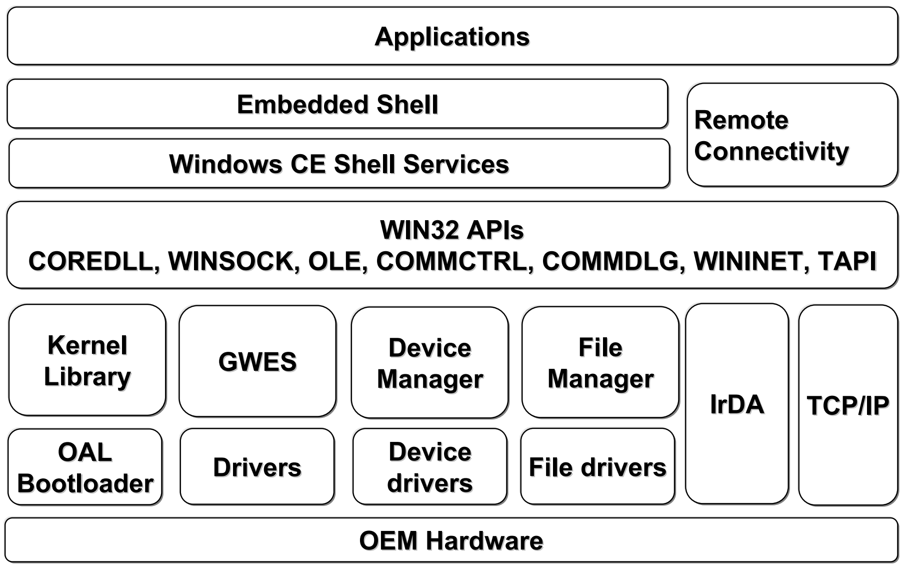

Figure 1. Microsoft Windows CE® Architecture

layer. Remote API (RAPI) capability is built on top of the communication layer. Database and file system is built on top of the kernel. RAPI enables remote applications running on desktop PC's to connect to devices running Windows CE. Application executes in its own address space and interacts with the rest of the OS via the Win32 system call interface [6, 12]. Of the layers, all except the lowest are implemented by Microsoft. The well defined OEM adaptation layer makes Windows CE easily portable to other platforms.

Other key features of Windows CE includes a subset of the Win32 API that addresses the most commonly needed services, a low overhead user level device driver model and a built-in power management.

Windows CE requires a 32 bit processor with MMU support for virtual memory management. While this requirement restricts the range of embedded processors on which it can be ported to, it has two fundamental advantages. Firstly, virtual memory simplifies the programming paradigm significantly. Secondly, it allows for portability of the code across systems supporting the same OS. The Windows CE architecture also places restrictions on processes' address spaces.

Windows CE is a fully multitasking, multithreaded operating system. A process can have any number of threads, but at any time there can be only a maximum of 32 processes and the virtual address space of each process is limited to 32 MBytes. Processes in Windows CE communicate using message passing and memory mapping. Memory mapping facilitates very fast data transfers between cooperating processes and can be used to dramatically enhance real-time performance [6, 12, 13].

Windows CE requires about 500 KBytes for a minimal kernel with some communication support. This is comparable to embedded Linux.

The key advantage of Windows CE is that it is an industrial strength operating system and a full member of the Microsoft .NET Compact framework. This is a feature rich infrastructure that enables the rapid development and deployment of web enabled and real-time applications. This compliments well the intended market of RSoC processors.

3 The Excalibur EPXA10 Development Board

The EPXA10 DDR development board, the latest and most sophisticated among the Altera Excalibur family embeds an ARM922T® processor core, memory, memory controllers and common peripherals with up to 1 million gate-equivalent of programmable logic [2].

The ARM922T processor core, with its support for MMU, facilitates deployment of numerous operating systems including Windows CE and Linux. Another advantage of ARM922T core is its support of the high performance Advanced Microcontroller Bus Architecture (AMBA) specification utilizing the advanced high-performance bus (AHB) standard.

The programmable logic device (PLD) in the Excalibur chip is configurable under processor control. A distinct advantage of Excalibur family is its integrated processor sub-system that is capable of configuring the PLD. This facilitates dynamic reconfiguration of the PLD under processor control. The EPXA10 DDR development board has around one million logic gates, which makes it the largest of all other members in the Excalibur family.

Although Altera provides a rich set of development and design tools for its Excalibur devices, an op-

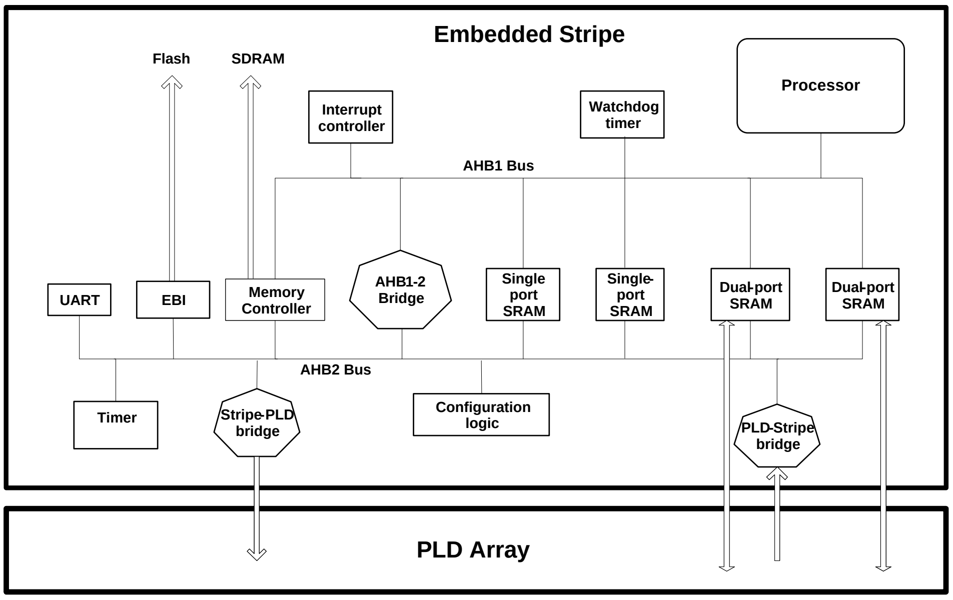

Figure 2. Architecture of Altera Excalibur®

erating system such as Windows CE that can abstract the memory characteristics and that is capable of dynamically loading and unloading device drivers can greatly enhance the usability of this platform. The next section provides a more detailed description of our experience in porting Windows CE to the Excalibur [1].

4 Porting Windows CE

To port the kernel of Windows CE, the lowest layer of its hierarchy, described in Section 2, has to be implemented. This involves implementing a boot loader, an OAL layer and the required device drivers. The boot loader consists of platform specific code, written in ARM assembly with the rest being library code supplied by Microsoft. We implemented a flash boot loader, which is a boot loader capable of booting an operating system image residing in the flash, for the Excalibur board. Altera provides a flash programming utility which is used to program the flash with the boot loader image. When the board is reset, the platform specific code of the boot loader starts executing. It performs the necessary platform set up which include setting up the Excalibur clocks which drives the embedded processor and other peripherals, setting up the memory map of peripherals on board, initializing the SDRAM etc. The boot loader copies the kernel code to RAM and passes control to the startup routine in the kernel sources.

The kernel code itself consists of Microsoft supplied libraries and a platform specific OAL layer which is to be implemented by the original equipment manufacturer. In other words, porting the kernel involves building a customized kernel for the custom platform. The set of OAL routines carry the prefix '`OEM`' to indicate that they are associated with hardware manufacturers developing custom CE platforms. Although Microsoft has documented the required OEM functions to be implemented in order to successfully develop an OAL layer, kernel documentation is scarce, and it led to some difficulties in the beginning. However, Windows CE turned out to be very easily portable, owing to its well defined and easy to implement OAL layer. Microsoft also provides a Kernel Independent Transport Layer [14] (KITL), by which the operating system kernel running on the Excalibur board can connect to the Platform Builder software, running on an x86 based host PC. Platform Builder delivers all the tools developers need in order to build Windows CE based systems quickly. The integrated development environment (IDE) enables users to configure, build and debug the OS running on the target. It also includes Microsoft embedded Visual Tools, which offers Microsoft Visual Basic and Visual C++ optimized for embedded development. The KITL layer allows the target to be connected to the PC over any available transport channel. Platform Builder's integrated development environment and KITL's connectivity provide a rich environment by which the user can interact with the kernel. It is possible to dynamically reconfigure the PLD by means of this facility.

Device driver development completes the development of a custom *board support package* (BSP). The device driver architecture of Windows CE is unique. Windows dynamic link libraries (DLL) are used for the dynamic loading of device drivers upon installation and identification of a device. All Windows CE drivers run in user mode. Although this approach incurs extra over head, it has many advantages. The most obvious advantage is that a driver crash will not affect the stability of the kernel. Another important advantage is that drivers may access all the resources available to application developers. From our experience device driver development in Windows CE is fairly simple and straight-forward. We have implemented drivers for PLD configuration, accessing peripherals like stripe bridges, dual port RAM and flash memory. All of these implementations use the stream interface model.

5 Performance Evaluation

We tested the performance of our port of Windows CE on the Excalibur platform, using the OSBENCH tool [9]. This benchmark suite is implemented by Microsoft, with OEMs required to add necessary support functions. This tool measures the performance of the kernel by conducting performance tests on the scheduler. These tests include timing basic kernel operations such as synchronization. Windows CE supports the entire set of Win32 synchronization objects.

The OSBENCH test consists of 7 basic groups, namely, (a) critical section operations, (b) events, (c) semaphore, (d) mutex, (e) voluntary yield, (f) *protected server library* (PSL) API call overhead, and (g) interlocked operations APIs (decrement, increment, test-exchange, and exchange). Protected server libraries (PSL) are kernel libraries that borrow the resources of a calling thread to minimize the amount of stack space needed as well as to increase efficiency when used in conjunction with critical sections. The calculations are based on the elapsed clock ticks for each operation. When the time taken for an operation is much smaller than the overhead incurred in calling the performance routines itself, the operation is looped for a fixed number of *iterations per sample* (IPS). An IPS of 1 means the elapsed clock ticks is taken for only one instance of the operation. For tests which completes very fast, an IPS of 1000 is chosen and individual run times are averaged out. For tests of IPS = 1, run time is calculated with both a cache flush on each sample and without a cache flush on each sample. For tests of IPS = 1000, this is not possible as the cache cannot be flushed between each iteration of the operation.

Table 1 shows the performance results we obtained on the Excalibur EPXA10 platform running ARM922T processor (200MHz) for Windows CE .NET 4.1. Tests are conducted for 100 samples and results are averaged out. Wherever possible, we took timing for thread-to-thread performance within a process as well as across processes. We should emphasize that these results were obtained without tuning the kernel much and we expect that better results will be achievable in the future. Nonetheless, the results we obtained are comparable to those obtained for Pentium processors of similar clock speeds [10].

Our results show that there is not much performance difference between the inter-process and intra-process tests. This may be explained by the nature of processes and threads supported by Windows CE. A process in Windows CE consumes no resources except the memory footprint it occupies. However threads use more system resources than a process. It uses the processor registers and requires a stack. It is threads that the Windows CE scheduler schedules. Scheduling is based on a priority scheme and is independent of the process to which the thread belongs. Also the effect of cache is significant as the results show. However it should be noted that this is very much dependent on the system state and number of threads running on the system.

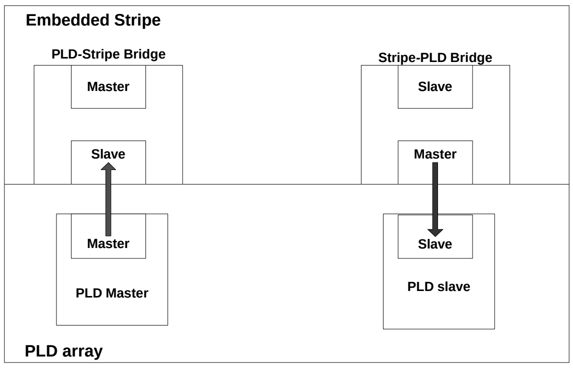

Figure 3. Excalibur Bridge Architecture

6 Using the PLD

The Excalibur has 3 interfaces to access the PLD device: the Stripe-PLD Bridge, the PLD-Stripe Bridge, and the dual port SRAM (DPSRAM) interface. The PLD-Stripe and Stripe-PLD bridges [3] follow the standard AHB interface specified in the AMBA specification [5] with slight modifications. The interface is shown in Fig. 3. The Stripe-PLD Bridge facilitates the bus masters in the Stripe to access slaves in the PLD. It appears as a AHB master to the PLD slave. Similarly, the PLD-Stripe Bridge allows PLD-masters to access resources in the embedded stripe such as the SDRAM, the expansion bus interface etc. Fig. 4 shows how the processor accesses a PLD slave. The processor resides in the AHB1 bus and gain access to the Stripe-PLD Bridge via the AHB1-2 Bridge. Both bridges include synchronization logic, allowing the master and slave interfaces to reside in a different clock domains. Although this interface is slower than the conventional DPSRAM interface described below, the PLD-Stripe Bridge gives PLD devices access to around 128 MBytes of memory in SDRAM, making it possible to implement memory intensive hardware in the PLD.

The dual port SRAM interface (DPSRAM) [4]

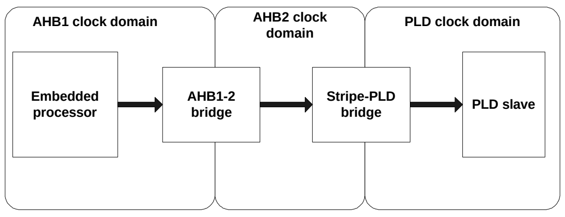

Figure 4. Bridge Transaction Paths

is a conventional interface mode with port A accessible to PLD and port B accessible to AHB1 or AHB2 buses. An arbiter determines whether AHB1 or AHB2 should be granted access. DPSRAM is a faster interface than embedded stripe bridges. The Excalibur EPXA10 board contains 128 KByte of dual port SRAM with several configuration modes.

Device drivers are implemented for both the bridges and DPSRAM device. The application programs running on top of the operating system, access these devices over user mode device driver interface. In the PLD the appropriate AHB slave/master interfaces should be implemented. A future research effort is to explore the possibility of extending the Windows CE support to make PLD programming more effortless and portable.

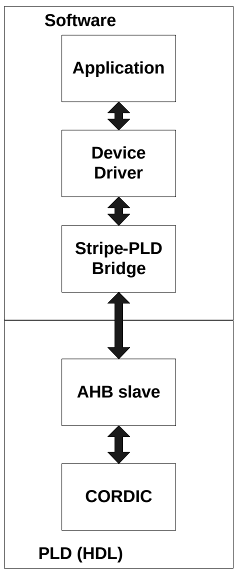

Figure 5. CORDIC Implementation

7 Evaluating a PLD-based Solution

The existence of a tightly coupled reconfigurable hardware device along with an embedded processor allows for the off-loading of functionalities in hardware.

A simple application written in Altera provided C libraries will run faster than an application running on top of an operating system like Windows CE. So the ease of programming and an operating system environment will induce some performance overhead which has to be taken into account when designing a performance sensitive PLD application.

In order to quantify the performance of a PLD application running on top of operating system, we evaluated the performance of the CORDIC algorithm implemented using pure software as well as in hardware (see Fig. 5). For the latter, the CORDIC core is implemented in the PLD, while the software controlling the application runs on top of the operating system. The hardware CORDIC core [7] is modified to interface with an AHB slave designed in Verilog. The AHB slave talks to the master port of the Stripe-PLD Bridge. The application code runs on top of the operating system layer, and interacts with the Stripe-PLD Bridge using the device driver interface. The pure software model also runs on top of Windows CE.

We also implemented the same benchmark using Altera's development tools running without any operating system. Our implementation has three clock domains: the AHB1 clock domain runs at 200 MHz, the AHB2 clock domain is clocked at 100 MHz, while the PLD runs at 48 MHz.

The results are shown in Table 2. The listed results are for a single instance calculation, or the overhead can be treated more or less the worst case analysis. Again for our experiments we have used the embedded stripe bridge as the Stripe-PLD interface.

8 Conclusion

This paper gives a brief overview of our porting experience of Windows CE 4.1 .NET onto the Excalibur EPXA10 DDR development board from Altera. We have also presented various benchmark results, and performance analysis for a PLD application implementing the standard CORDIC core. The operating system environment increases the ease of use and feature set of the Excalibur board to a large extent. This reduces the development time to a large extent. Looking forward, we intend to further our efforts and explore the possibility of extending operating system support to make PLD programming more effortless. We would also like to work on applications that can effectively use the PLD. At the time of writing of this paper, we have also just completed a port to Windows CE 4.2 .NET. We are currently testing the port with the hope releasing it shortly.

9. References

[1] Altera Inc., *Excalibur Hardware Reference Manual.* `http://www.altera.com/literature/manual/mnl_arm_hardware_ref.pdf`

[2] Altera Inc., *Excalibur EPXA10 DDR development board Hardware Reference Manual.* `http://www.altera.com/literature/manual/mnl_epxa10devbd.pdf`

[3] Altera Inc., *Excalibur Solutions- Using the Embedded Stripe Bridges.* `http://www.altera.com/literature/an/an142.pdf`

[4] Altera Inc., *Excalibur Solutions- DPSRAM Reference Design.* `http://www.altera.com/literature/lit-exc.jsp`

[5] ARM Inc., *ARM AMBA specification Rev 2.0.* `http://www.arm.com/products/solutions/AMBAHomePage.html`

[6] D. Boling, *Programming Microsoft Windows CE .NET (Third Edition)* Microsoft Press, 2003.

[7] R. Herveille. *CORDIC open core.* `http://www.opencores.org/projects.cgi/web/cordic/overview`

[8] Convergence Promotions. *Dynamic System Reconfiguration with Excalibur Devices.* `http://www.convergencepromotions.com/pdf/iss4(pg58-60).pdf`

[9] Microsoft Inc., *MSDN OSBENCH Documentation.* `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/cmconosbenchexe.asp`

[10] Microsoft Inc., *OSBENCH Results.* `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncenet/html/ rtnetdesigning.asp`

[11] Microsoft Inc., Microsoft Windows CE Website. `http://msdn.microsoft.com/embedded/ce.net/default.aspx`

[12] J.Y. Wilson, and A. Havewala, *Building Powerful Platforms with Windows CE.* Addison-Wesley Publications, 2001.

[13] Microsoft Inc., Microsoft Windows CE .NET Core OS Services. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcemain4/html/_wcesdk_Kernel_Services.asp`

[14] Microsoft Inc., *MSDN KITL Documentation.* `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wcepb40/html/_pbplatman_Ethernet_Debugging_Transport.asp`

[15] S. Moore, J. Srinivasan, and D. Wilson, *Hardware/Software Co-Design Teaching Package* `http://www.cl.cam.ac.uk/Teaching/current/ECADArch/extra/WinCE-EPXA1-19april2004/EPXA1-poster.pdf`

[16] M. Vuletic, L. Righetti, L. Pozzi, and P. Ienne, “Operating system support for interface virtualisation of reconfigurable coprocessors.” *Proc. of the 2nd Workshop on Application Specific Processors*. Dec 2003.

Benchmark Description	**Average Time (μs)**
1. Critical Section Tests	
Time from a lower priority thread calling `LeaveCS` to a higher priority thread unblocking from the `EnterCS` call (IPS = 1)	50.451 87.767 (CFlush)
Time from a lower priority thread calling `LeaveCS` to a higher priority thread unblocking from the `EnterCS` call (IPS = 1)	53.866 93.448 (CFlush)
Time required to execute an `EnterCS` where there is no contention (IPS = 1000)	0.319
Time required to execute a `LeaveCS` where there is no contention (IPS = 1000)	0.375
2. Event Handling	
Time from a thread to set an event which will wake up another thread in the same process waiting for that event (IPS = 1)	31.634 90.619 (CFlush)
Time from a thread to set an event which will wake up another thread in a different process waiting for that event (IPS = 1)	30.961 92.049 (CFlush)
3. Semaphore Signaling	
Time from a lower priority thread releasing the semaphore to a higher priority thread in the same process unblocking from a wait (IPS = 1)	35.814 89.173 (CFlush)
Time from a lower priority thread releasing the semaphore to a higher priority thread in another process unblocking from a wait (IPS = 1)	36.414 80.808 (CFlush)
4. Mutex	
Time from a lower priority thread releasing the mutex to a higher priority thread in the same process unblocking from a wait (IPS = 1)	48.770 111.4 (CFlush)
Time from a lower priority thread releasing the mutex to a higher priority thread in another process unblocking from a wait (IPS = 1)	58.923 101.948 (CFlush)
5. Yield to thread	
Time taken for a thread to start running after another thread of the same priority and of the same process voluntarily yields by calling `Sleep(0)` (IPS = 1)	17.369 31.662 (CFlush)
Time taken for a thread to start running after another thread of the same priority but of another process voluntarily yields by calling `Sleep(0)` (IPS = 1)	17.076 32.130 (CFlush)
6. Interlocked variable access	
Time required to call `InterlockedIncrement()` API function (IPS = 1000)	0.193
Time required to call `InterlockedDecrement()` API function (IPS = 1000)	0.193
Time required to perform an interlocked exchange operation (IPS = 1000)	0.148
Time required to perform an interlocked test operation (IPS = 1000)	0.153
7. Protected Server Library API call	
Time required to call a PSL routine with no parameters and returns immediately (IPS = 1000)	3.731
Time required to call a PSL routine with 7 DWORD parameters and returns immediately (IPS = 1000)	3.767
Time required to call a PSL routine with 7 PVOID parameters and returns immediately (IPS = 1000)	4.252
Time required to call a PSL routine that is in a different process with no parameters and returns immediately (IPS = 1000)	3.691
Time required to call a PSL routine that is in a different process with 7 DWORD parameters and returns immediately (IPS = 1000)	3.769
Time required to call a PSL routine that is in a different process with 7 PVOID parameters and returns immediately (IPS = 1000)	4.222
Time required to call a PSL routine that is in a different process function in `NK.exe` (kernel) which returns immediately (IPS = 1000)	3.848

Table 1. Performance on OSBENCH.

Pure SW version running on WinCE	**SW/HW version running on WinCE**	**Pure SW version using Altera lib.**	**SW/HW version using Altera lib.**
2130	348	153	30 to 35

Table 2. Performance of various versions of CORDIC Benchmark (in μs).