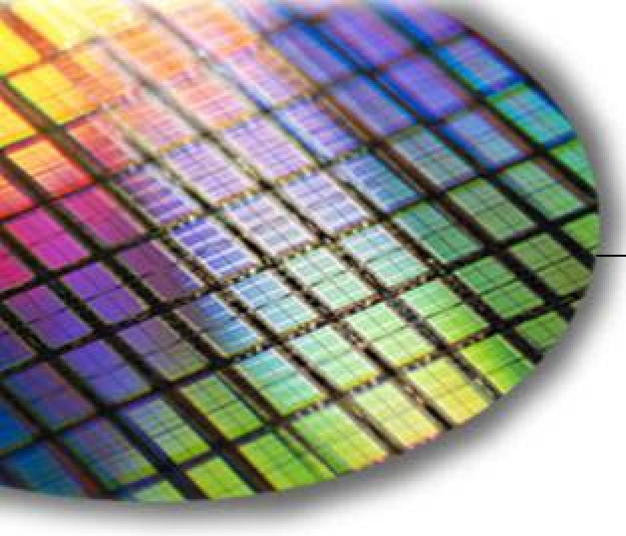


The World Leader in High Performance Signal Processing Solutions

An Introduction to the VisualDSP++ Kernel (VDK)

Presented by:
Ken Atwell
Product Line Manager

ANALOG DEVICES

About This Module

- This module discusses the VisualDSP++ kernel (VDK) concepts and capabilities.

- It is recommended that users have:
  - A working knowledge of software terminology
  - Previous experience with other commercial or home-grown operating systems

Module Outline

- Introduction
  - Operating system choices for Blackfin
  - Introducing the VDK
- Capabilities of the VDK
- On-line Demo: Building and Debugging VDK Projects
- Timings and Sizes
  - Footprints and benchmarks

OS Choices for Blackfin Processors

uClinux

µC/OS-II
The Real-Time Kernel

THREADX

Green Hills
SOFTWARE, INC.

INTEGRITY

velOSity
MICROKERNEL

ANALOG DEVICES
BLACKfin

Quadros
Systems Inc.

NUCLEUS®

unicoi
SYSTEMS

Find third party RTOS options at:
http://dspcollaborative.analog.com/developers/DSP_ThirdParty_Search_Home.asp

Introducing the VisualDSP++ Kernel (VDK)

- Small, robust kernel bundled with VisualDSP++
  - Designed for application with light-weight OS requirements
  - No additional cost or run-time licenses/fees
  - Fully supported and maintained along with the rest of VisualDSP++
- Supports all current and future Blackfin derivatives
- Complements and co-exists with System Services and its device drivers

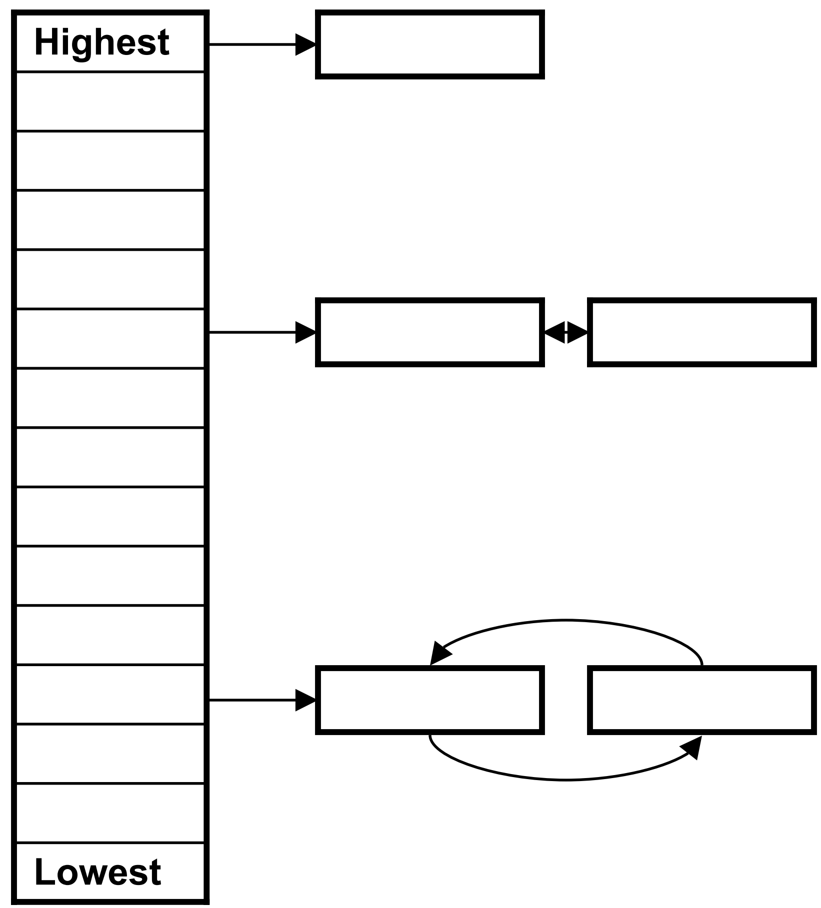
VDK Concepts

- Threads
- Priority and scheduling
  - Pre-emptive, cooperative, and time-slicing
- Critical and unscheduled regions
- Semaphores, including periodic
- Messages
- Beyond discussion today, but available in VDK
  - Events and event bits
  - Multi-processor messaging
  - Memory pools
  - Device flags
- Fully documented in VisualDSP++ help and PDF manuals

Capabilities of the VDK

Threads and Priorities (1)

- **Arbitrary number of threads running at 31 priority levels**
  - Preemptive scheduling by priority, cooperative or time sliced within a priority
  - Priority can be statically or dynamically assigned
  - A thread's life ends when the "run" function exits



ANALOG DEVICES

Threads and Priorities (2)

- Arbitrary number of threads running at 31 priority levels
  - Preemptive scheduling by priority, cooperative or time sliced within a priority
  - Priority can be statically or dynamically assigned
- Threads may be instantiated at boot time or later at run time
  - Each thread gets its own stack
- Each thread implements four functions
  - Create, run, destroy, error
  - All major execution occurs in "run", many threads never exit run
  - A thread's life ends when the "run" function exits

VDK_ClearThreadError()
VDK_CreateThread()
VDK_CreateThreadEx()
VDK_DestroyThread()
VDK_FreeDestroyedThreads()
VDK_GetLastThreadError()
VDK_GetLastThreadErrorValue()
VDK_GetPriority()
VDK_GetThreadID()
VDK_GetThreadStackUsage()
VDK_GetThreadStatus()
VDK_ResetPriority()
VDK_SetPriority()
VDK_SetThreadError()
VDK_Sleep()
VDK_Yield()

Critical and Unscheduled Regions

- Critical regions disables all interrupts and context switches
  - Use with discretion to perform actions that cannot be interrupted
  - Typical used in test-and-set or read-modify-write style operations
- Unscheduled regions are less drastic, disabling the VDK context switch only
  - Other interrupts are allowed to continue

VDK_PopCriticalRegion()
VDK_PopNestedCriticalRegions()
VDK_PopNestedUnscheduled\
Regions()
VDK_PopUnscheduledRegion()
VDK_PushCriticalRegion()
VDK_PushUnscheduledRegion()

Semaphores (1)

- A facility for coordination between threads or from an ISR to a thread
- A semaphore can be used to a control access to a shared resource in threads
  - For example, protecting a buffer from simultaneous read and write

VDK_CreateSemaphore()
VDK_DestroySemaphore()
VDK_MakePeriodic()
VDK_PendSemaphore()
VDK_PostSemaphore()
VDK_RemovePeriodic()

Semaphores (2)

- A facility for coordination between threads or from an ISR to a thread
- A semaphore can be used to a control access to a shared resource in threads
  - For example, protecting a buffer from simultaneous read and write
- Many semaphores are of a Boolean nature (yes or no), but they can also be used to allow multiple access, “counting semaphores”
- Semaphores may be periodically and automatically posted by the VDK

VDK_CreateSemaphore()
VDK_DestroySemaphore()
VDK_MakePeriodic()
VDK_PendSemaphore()
VDK_PostSemaphore()
VDK_RemovePeriodic()

Messages

- A message is a targeted transmission from one thread to another
  - Message type
  - Payload address and size
    - An arbitrary amount of information of any type may be passed between two threads
- Facilities for multi-core/processor messaging exist

VDK_CreateMessage()
VDK_DestroyMessage()
VDK_ForwardMessage()
VDK_FreeMessagePayload()
VDK_GetMessageDetails()
VDK_GetMessagePayload()
VDK_GetMessageReceiveInfo()
VDK_MessageAvailable()
VDK_PendMessage()
VDK_PostMessage()
VDK_SetMessagePayload()

On-line Demo:
Building and Debugging VDK Projects

- Project Window/VDK tab
- Thread creation template
- VDK Status Window
- VDK History Window

Sizes and Timings

VDK Static Memory Footprints

Application	Code	Data
One C-language thread, no API calls	5384	1120
Two C-language threads, no API calls	5584	1208
Two threads, usage of a static semaphore	6916	1252
Add critical regions	7068	1252
Add message passing	9260	1292
Add a history window of 512 events and full instrumentation (debugging scenario)	13304	9536

Size of VDK libraries' contributions, not entire application. Measured under VisualDSP++ 4.5 (base release) with dead code/data elimination enabled. Sizes are in bytes.

Cycle Counts for Performance-Sensitive Activities

Event	Cycles
Boot (from reset vector to first instruction of highest priority thread's run function)	15,311
Tick, no change of thread	67
Tick, change of thread	722
Post semaphore, no change of thread	76
Post semaphore, change of thread	286
Push a critical region, increment a global variable, pop	199
Create a new thread, no change of thread	2,352

Measurements with entire application in internal memory. The application contains five running threads. Measured with VisualDSP++ 4.5 (base release). Processor was an ADSP-BF533 r0.5.

Conclusion

- VDK is provided at no additional cost with VisualDSP++
  - Designed to be a robust solution for light-weight requirements
  - Many commercial RTOS's are also available for Blackfin
- Basic facilities include threads, prioritization, semaphores, messaging, and critical and unscheduled regions
  - Other facilities are available and documented as part of VisualDSP++
- VDK is well integrated into the VisualDSP++ user interface, with facilities to configure VDK, generate template thread code, and display status and history while debugging your application

For Additional Information

- Review other BOLD topics (especially System Services and drivers)
- Take a test drive of VisualDSP++ and/or get an EZ-KIT Lite
  - http://www.analog.com/processors/VisualDSP/testDrive.html
  - Examples demonstrating all major features
    - …\Blackfin\Examples\No Hardware Required\VDK
- Consult detailed documentation within VisualDSP++
  - Also available for download
    - http://www.analog.com/processors/blackfin/technicalLibrary/manuals
- Find third party RTOS options
  - http://dspcollaborative.analog.com/developers/DSP_ThirdParty_Search_Home.asp

- For questions, use the “Ask a Question” button