Embedded DSP: Visual DSP++ Kernel (VDK) for Real-Time

Motivation

- Applications require control code as support for the algorithm, often thougth of as the „real“-program, e.g.
  - Data have to be moved to and/or from peripherals (I/O-devices, memory, ADC, DAC, ...) , functions depends on the control of interrupts, often polling modes are used for reading the state of input switches or sensors.
  - Most algorithms consitst of more than one functional block.

- For some systems, the control code may be as simple as a loop, processing data which arrives to the processor at a constant rate.
- A better solution is to control more sophisticated the flow of the internal and external signals, events, interrupts in order to use the full potential of the procesor.

Solution:

- Kernel based programming environment as a small real-time operating system (RTOS)

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Benefits:

Rapid Appliaation Development

- The use of a ‚RTOS‘ allows rapid development of applications compared to integrate all of the control code by hand.
- Better and more documented programs by using the automatic code generation and file templates.
- Usage of standard programming interfaces to device drivers allows to concentrate on the algorithms (the main application) and the desired control flow rather than on the implementation details.

Debugging Control Structures

- Statistical information and logging of all significant events into a history (trace) buffer.
- The content of the buffer can be presented as a graphical trace of the execution,:
  - Including context switches.
  - Pending and posting of signals
  - Changes in thread‘s status
  - Statistics are presented for each thread and show the total amount of execution time of the thread, the number of times this thread has been activated, the signal which is currently blocked, and other specific data.

Embedded DSP: Kernel & Operating System

Visual DSP++™, Kernel (VDK™)

Users Guide, Fourth Revision 2001
(Analog Devices, Inc.
Digital Signal Processing Division)

VisualDSP++ Kernel (VDK)

- RTOS kernel from Analog Devices. VDK is a part of VisualDSP++. The kernel is integrated with the Integrated Development and Debugging Environment (IDDE) and assembler, compiler, and linker programs into the DSP development tool chain. VDK is supported on ADSP-219x, ADSP-21xxx, ADSP-TSxxx, and BLACKfin processors.

Embedded DSP: Visual DSP++ -- Kernel (VDK) Definitions

Application Programming Interface (API):

- A library of C/C++ functions and/or assembly macros that define VDK services. These services are essential for kernel-based application progamms. The services include interrupt handling, thread management, and semaphore management, among other services.

Context Switch

- A process of saving/restoring the processor’s state. The scheduler performs the context switch in response to the system change. A hardware interrupt can occur and change the state of the system at any time. Once the processor’s state has changed, the currently running thread may be swapped with a higher-priority thread. When the kernel switches threads, the entire processor’s state is saved and the processor’s state forthe thread being switched in is restored. This process is known as a context switch.

Critical Region:

- A sequence of instructions, execution of which cannot be interrupted or swapped out. To ensure that the execution of a critical region is uninterrupted, all interrupt service routines must be suspended before calling the critical region. Once the critical region routine has has been completed, ISRs are again enabled.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Event

- A signal, similar to a semaphore, used to synchronize multiple threads in a system. An event is a logical switch, having two binary states (available/true and unavailable/false) that control thread execution. When an event becomes available, all pending (waiting) threads in the wait list are set to be ready-to-run. When an event is available and a thread pends on it, the thread continues running and the event remains available.
- For error handling purposes, threads can specify a timeout period when pending on an event. A certain time period can be set for posting an event at a given time, thus allowing threads to be made ready to run periodically.
- An event is a code object of global scope, so any thread can pend on any event. Event properties include the EventBit mask, EventBit value, and combination type. Events are statically allocated and enumerated at run-time. An event cannot be destroyed, but its properties can be changed (see LoadEvent()).

EventBit

- A flag set or cleared to post the event. The Event is posted (available) when the current values of the system EventBits match the EventBit's mask and EventBits' values defined by the Event's combination type. There is one and only one EventBits word in a system. It matches the width of the processor's data bus: sixteen bits for ADSP-219x DSPs; thirty-two bits for ADSP-21xxx, and ADSP-TSxxx DSPs.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Device Driver:

- A user-written routine that abstracts the hardware implementation from the application code. User code accesses device drivers through a set of Device Driver APIs.

Interrupt

An external or internal condition detected by the hardware interrupt controller. In response to an interrupt, the kernel processes a subroutine call to a predefined Interrupt Service Routine (ISR). Interrupts have the following specifications:

- **Latency** — interrupt disable time—the period between the interrupt occurrence and the first ISR's executed instruction.
- **Response** — interrupt response time — the period between the interrupt occurrence and a context switch.
- **Recovery** — interrupt recovery time — the period needed to restore the processor's context and to start the return-from-interrupt (RTI) routine.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Interrupt Service Routine (ISR)

- A routine executed as a response to a software interrupt or hardware interrupt. VDK supports nested interrupts, meaning that the kernel recognizes other interrupts and/or services interrupts with higher priorities while executing the current ISR. VDK ISRs are written in assembly language. VDK reserves the timer and the lowest priority (idle) interrupt.

Real-Time Operating System (RTOS)

- A software executive that handles algorithms, peripherals, and control logic. The fist release of Analog Devices RTOS comprises the following components: kernel, communication manager, support library, and device drivers. RTOS enables structured, scalable, and expandable DSP application development while hiding OS complexity.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

RTOS Kernel

- The main module of any Real-Time Operating System. The kernel loads first and permanently resides in the main memory. Typical kernel manages ther modules of the Real-Time Operation System. Kernel typicals ervices include context switching and communication management between OS modules.

Pre-emptive Kernel

- VisualDSP++ Kernel is a priority-based kernel in which the currently running thread of the highest priority is pre-empted, or suspended, to give system resources to the new highest-priority thread.

Scheduler

- A kernel component responsible for scheduling of system threads and interrupt service routines. VDK is a priority-based kernel in which the highest-priority thread is executed first.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Round-Robin Scheduling

- A scheduling scheme where all threads at a given priority are given processor time automatically in fixed duration intervals. Round-Robin priorities are specified at build time.

Semaphore

- A signal (similar to an event) used to synchronize multiple threads in a system. A semaphore is a logical switch, two states of which (available/true and unavailable/false) control any thread execution. Unlike an event, whose state is automatically calculated, a semaphore is directly manipulated via PendSemaphore() or PostSemaphore(). Posting a semaphore takes a deterministic amount of time and may incur a context switch.

Signal

- A method of communicating between multiple threads. VDK supports three types of signals: semaphores, events, and device flags.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

System Configurator

- The System Configuration control is accessible from the Kernel tab on the Project Window. The Kernel Window provides a graphical representation of the data contained in the vdk.h and vdk.cpp files.

Thread

- A kernel system component that performs a predetermined function and has its own share of system resources. VDK supports multithreading, a run-time environment with concurrently executed independent threads. Threads are dynamic objects that can be created and destroyed at runtime. Thread objects can be implemented in C, C++, or assembly language. A thread's properties include an ID, priority, and current state (wait, ready, run, or interrupted). Each thread maintains its own C/C++ stack.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Unscheduled Region

- A sequence of instructions whose execution can be interrupted, but cannot be swapped out. The kernel acknowledges, controls and services interrupts when an unscheduled region routine is running.

VisualDSP++ Kernel (VDK)

- RTOS kernel from Analog Devices. VDK is a part of VisualDSP++. The kernel is integrated with the Integrated Development and Debugging Environment (IDDE) and assembler, compiler, and linker programs into the DSP development tool chain. VDK is supported on ADSP-219x, ADSP-21xxx, ADSP-TSxxx, and BLACKfin processors.

Ticks

- The system level timing mechanism. Every system tick corresponds to a timer interrupt.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Code Reuse

- The VDK supports the complete infrastructure for control flow by libraries.
- Forces and supports good coding practice and organization by partitioning large applications into maintainable and comprehensible functional blocks.
- Each thread of execution is created from a user-defined template
  - At boot time
  - Dynamically by another thread during runtime
  - Multiple threads can be created from the same template, but the state associated with each created instance is and remains unique.
  - Each thread template presents a full encapsulation of an algorithm that is unaware of other treads in the environment unless is has a direct dependancy.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Hardware Abstraction

- VDK provides a hardware abstraction layer.
- Software interfaces allow to write most of the applications in a platform-independant, high-level language (C and C++).
- The API is identical to all Analog Devices processors, allowing code to be easily ported to a different DSP processor type**.**
- VDK architecture identifies a crisp boundary around the two important subsystems areas, interrupt service routines and device drivers, and supports the traditionally difficult development wit a clear and transparent programming framework and code generation.
- Interrupt and device drivers are declared with a graphical user interface in the IDDE, which generates good commented code.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Partitioning an Application

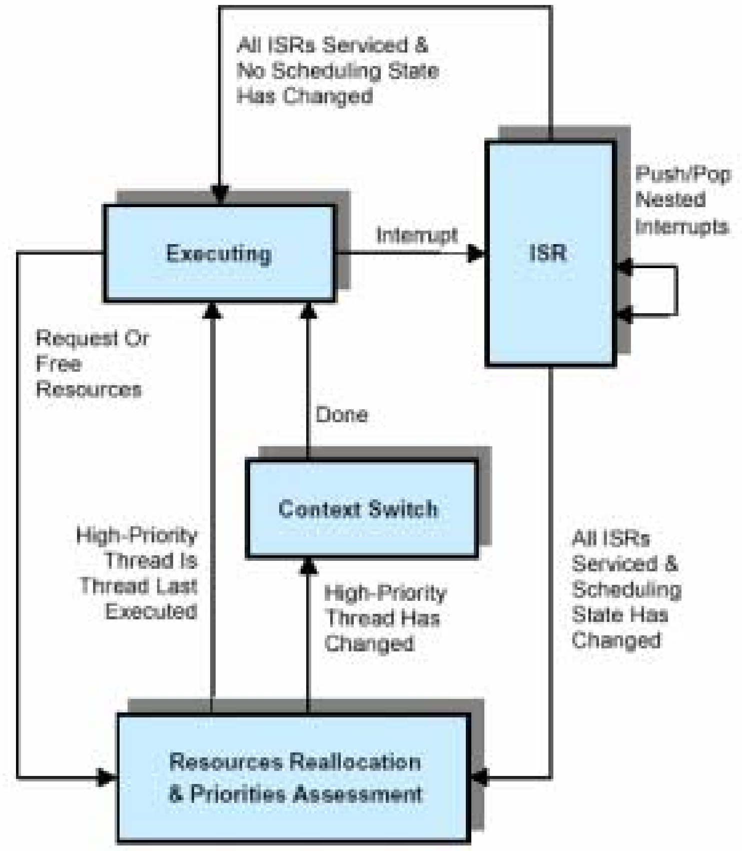
- A VDK thread is an encapsulation of an algorithm (code) and the associated data structure.
- Idea: partitioning the algorithm into smaller functional units that can be individually coded and tested.
- The building blocks then become reusable components in more robust, complex and scalable systems.
- Define the behavior of VDK threads by creating thread types.
  - Thread types are templates that define the behavior and data associated with all threads of tahta type.
  - For each thread type only one copy of the code is linked into the executable code.
  - Each thread has its own private set of variables defined for the thread type, ist own stack, and its own C run-time-context.
- Which parts should be partitioned ?
  - Identify parts of the design in which a similar algorithm is applied to multiple sets of data !
  - When data are available in sequential blocks, only one instance of the thread is required !
  - If the same operation is performed on separate sets of data simultaneously, multiple threads of the same type can work and be scheduled for prioritized execution.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Scheduling

- VDK is a pre-emptiv multitasking kernel.
- Each thread begins execution at its entry point.
  - The thread runs to coompletion or
  - Performs the main function repeatedly in an infinite loop.
- The scheduler has to pre-empt the execution of a thread and to resume ist execution when appropriate.
- Each thread has a priority to assist the scheduler in determining precedence of threads.
- The scheduler gives processor time to the thread with the highest-priority which is in the ‚ready'-state.
- A thread is in the ‚ready'-state when it is not waiting for any system resources (signals, events, shared memory) that it has requested. The state is stored in an internal kernel structure as a ready queue.

VDK State Diagram



All ISRs Serviced & No Scheduling State Has Changed
Push/Pop Nested Interrupts
Executing
Interrupt
ISR
Request Or Free Resources
Done
Context Switch
High-Priority Thread Is Thread Last Executed
High-Priority Thread Has Changed
All ISRs Serviced & Scheduling State Has Changed
Resources Reallocation & Priorities Assessment

Fraunhofer Institut Zerstörungsfreie Prüfverfahren

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Scheduling

Priorities

- Each thread is assigned to a dynamically-modifiable priority based on the default for its thread type declaration in VisualDSP++ environment's project window.
- An application is limited to 13 (14) priority levels, which depends on the processor type.
- Priority level one is the highest priority (13 or 14 is the lowest).
- The system maintains a special thread ‚IDLE' which has a lower priority than the lowest one.
- Assigning priorities is the most difficult work of designing a real-time pre-emptive system.
- Research works cover a lot of different methods, as based on deadlines (e.g. rate monotonic scheduling), ...
- Most systems are designed by considering the interrupts and signals that are triggering the execution, while balancing the deadlines which are imposed by the input and output streams of the system.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Scheduling

Pre-emption

- A running thread continues execution unless it requests a system resource using a kernel API-function.
- When a thread requests a signal (semaphore, event or device flag) and the signal is available, the thread resumes execution.
- If the signal is not available, the thread is removed from the ready queue (blocked).
- The kernel will not perform a context switch as long as the running thread maintains the highest priority in the ready queue.
- A thread can also be interrupted. When an interrupt occurs, the kernel yields to the hardware interrupt controller. After the completion of the ISR, the highest-priority thread resumes execution.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Scheduling

Protected Regions

- The VDK has two levels of protection for code that needs to execute sequentially unscheduled regions and critical regions.
- Critical regions can be entered within unscheduled regions, or unscheduled regions from within critical regions.
- Example: If qou are in an unscheduled region and call a function that pushes and pops a critical region, the system is still in an unscheduled region when the function returns.

Disabling Scheduling

- The VDK scheduler can be disabled by entering an unscheduled region. Necessary if you need to free multiple system resources without being switched out, or access global variables that are modified by other threads without preventing interrupts from being serviced by the API.
- While in an unscheduled region, interrupts are always enabled and ISRs execute !
- If a higher-priority thread becomes ready the kernel will not perform a context-switch.
- Unscheduled regions are implemented using stack style interface.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Scheduling

Disabling Interrupts

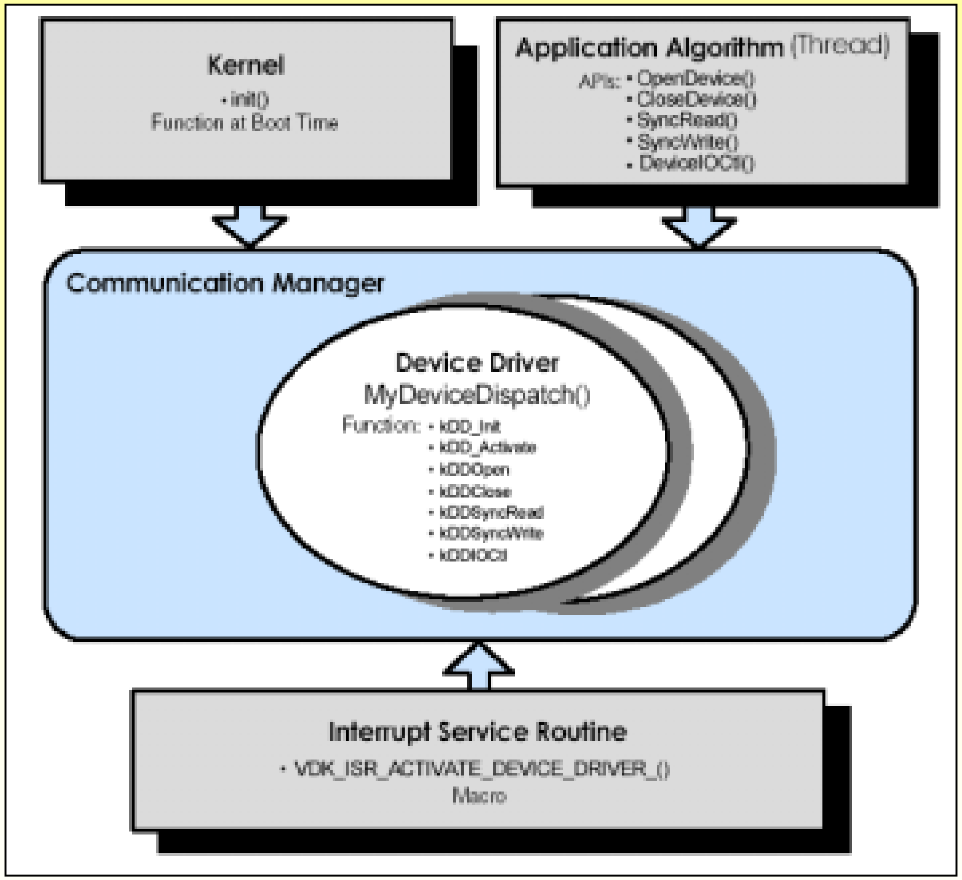
- Occasionaly disabling the scheduler does not provide enough protection to keep a block of thread code reentrant.
- A critical region disables the scheduler and interrupts.
- Critical regions are necessary when a thread modifies global variables that also may be modified by an ISR.
- Critical regions are implemented as a stack.
- It is possible to enter and exit critical regions in a function without any knowledge about the state of the critical region of the calling code.
- It is important to keep the critical region as short as possible, because it increses the interrupt latency.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Thread and Hardware Interaction

- Threads should have minimal knowledge of hardware; they should use device driver for hardware control.
- A thread is able to control and interact with a device in a portable and hardware abstracted manner by a standard set of APIs.
- Interrupts relay information to threads through signals to device drivers or directly threads.
- Using signals to connect hardware to the algorithms allows the kernel to schedule threads based on asynchronous events.
- The VDK run-time environment is a bridge between two domains, the thread and the interrupt domain (Software/Hardware).

Device Driver Entry Points



Embedded DSP: Visual DSP++ -- Kernel (VDK)

Thread Domain with Software Scheduling

- The thread domain runs under a C/C++ run-time model.
- The prioritized execution is performed by a software scheduler with full context switching.
- Threads should have little or no direct knowledge about hardware, they should request resources and then wait for them to become available.
- Threads are granted processor time based on priority and demanded resources (e.g. connect to ADC/DAC, UART, Flash-EEPROM, CODEC, LCD-Display, ...).
- Threads should minimize time spent in critical and unscheduled regions to avoid short-circuting the scheduler and the interrupt controller.

Interrupt Domain with Hardware Scheduling

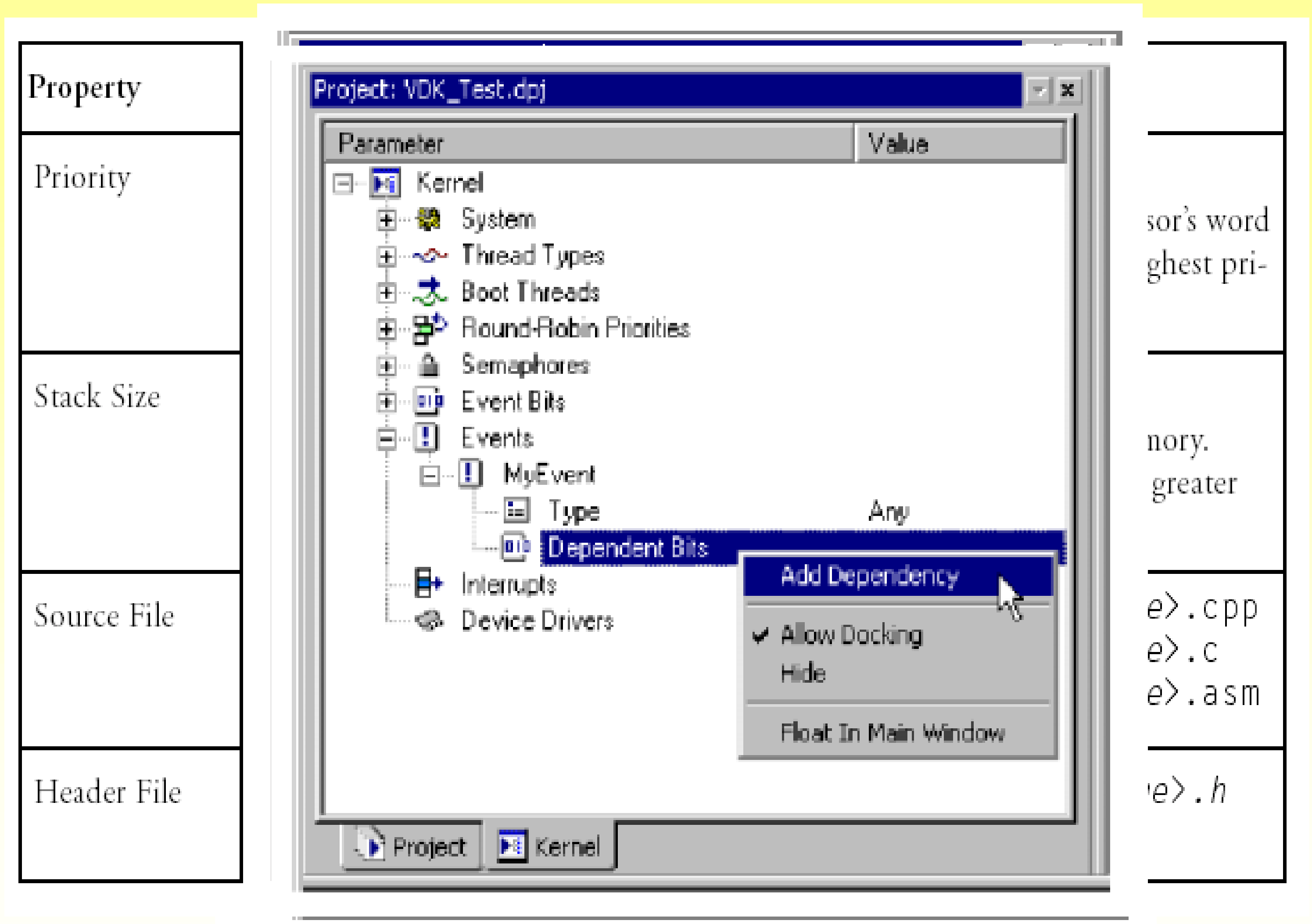
- The interrupt domain runs outside of the C/C++ run-time model.
- The prioritized execution is maintained by the hardware interrupt controller.
- ISRs should be as small as possible. (MiniProject !!)
- The job is to do as much as possible in order to acknowledge asynchronous events and to allow peripheras to continue operation in parallel with the processor.
- ISRs should only signal that more processing can occur and leave the processing to the threads.

Embedded DSP: Visual DSP++ -- Kernel (VDK)

Device Driver

- Interrupt Service Routines (ISR) are able to communicate with threads directly using signals.
- Interrupt Service Routines (ISR) and a thread can use a device driver to provide more complex device-specific functionality that is abstarcted from the algorithm.
- A device driver is a single function with multiple entry conditions and domain executions.
- (4.35 ...)

Embedded DSP: Visual DSP++ -- Kernel (VDK)

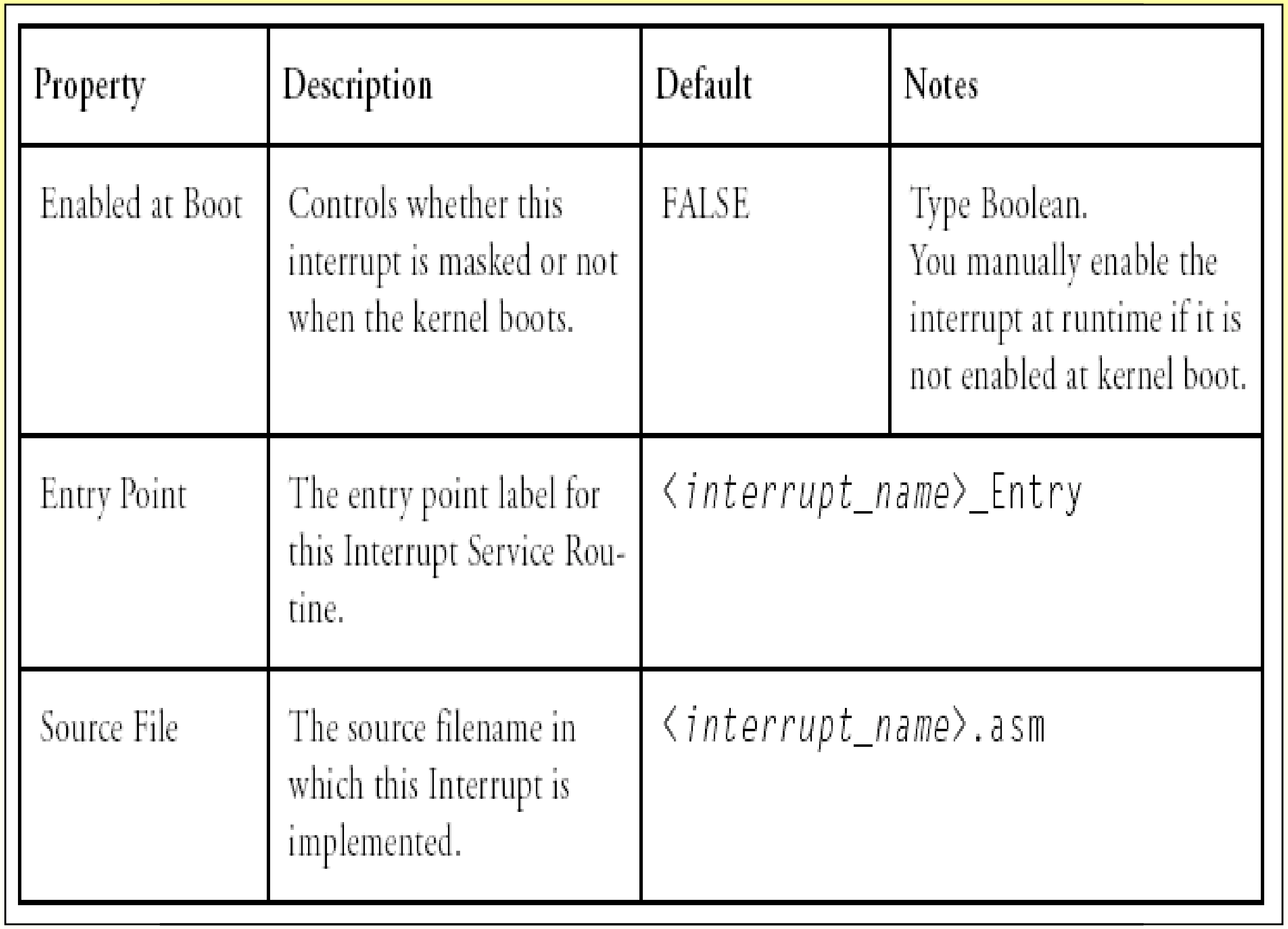


Property	
Priority	sor's word ghest pri-
Stack Size	nory. greater
Source File	e>.cpp e>.c e>.asm
Header File	e>.h

Project: VDK_Test.dpj

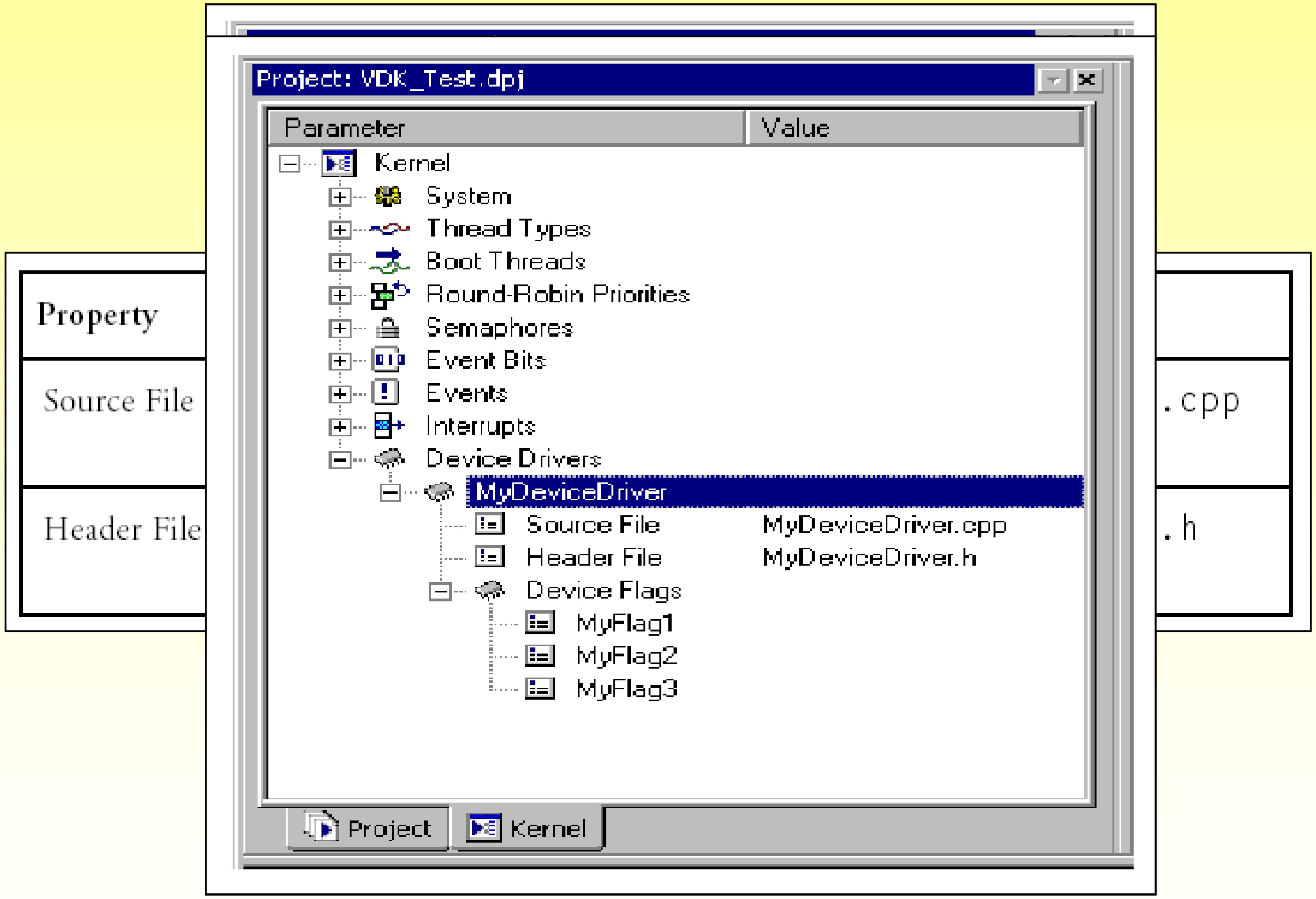
Parameter	Value
Kernel	
System	
Thread Types	
Boot Threads	
Round-Robin Priorities	
Semaphores	
Event Bits	
Events	
MyEvent	
Type	Any
Dependent Bits	
Interrupts	
Device Drivers	

Embedded DSP: Visual DSP++ -- Kernel (VDK)



Property	Description	Default	Notes
Enabled at Boot	Controls whether this interrupt is masked or not when the kernel boots.	FALSE	Type Boolean. You manually enable the interrupt at runtime if it is not enabled at kernel boot.
Entry Point	The entry point label for this Interrupt Service Routine.	`<interrupt_name>_Entry`	
Source File	The source filename in which this Interrupt is implemented.	`<interrupt_name>.asm`	

Embedded DSP: Visual DSP++ -- Kernel (VDK)

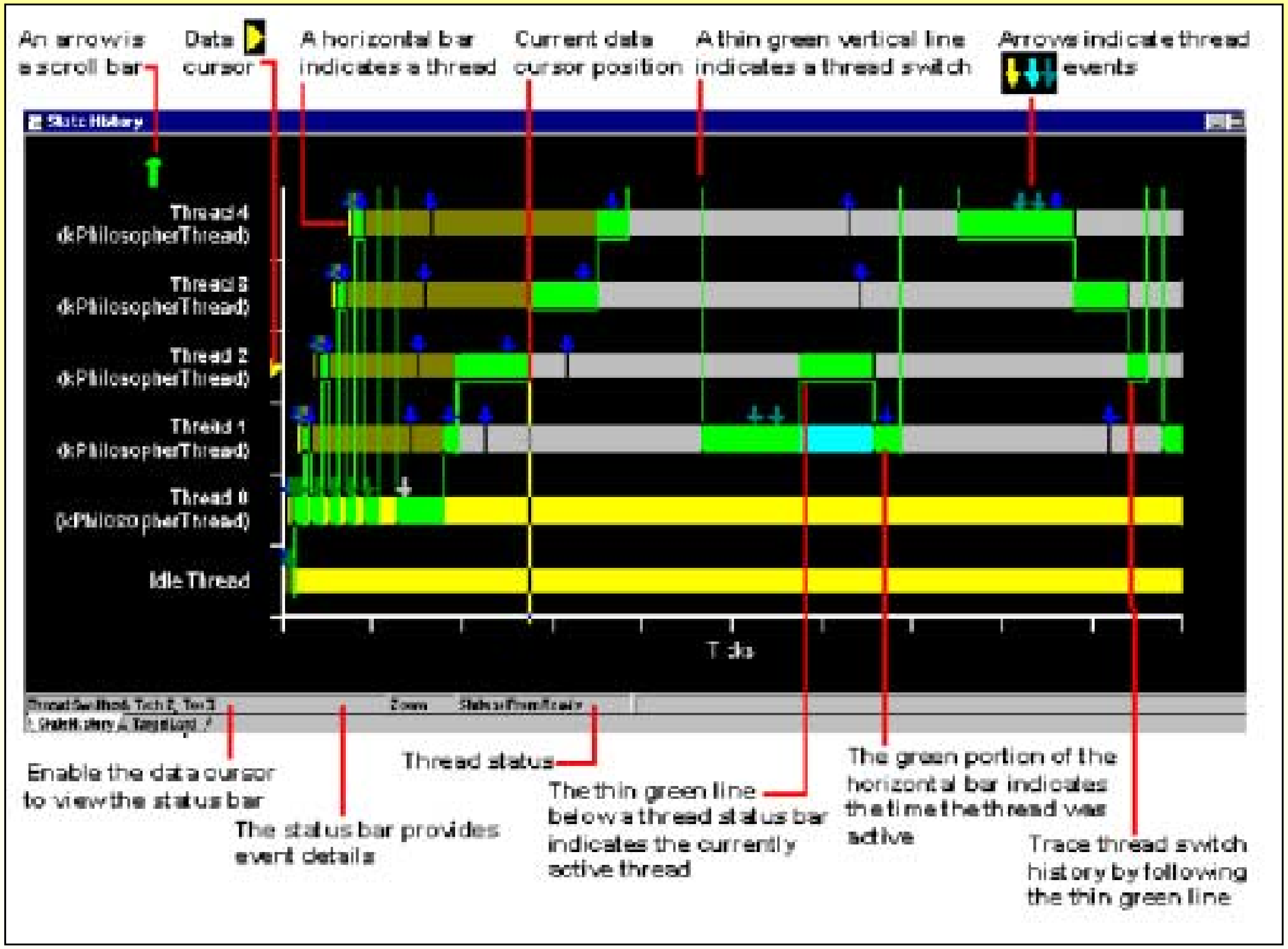


Property	
Source File	.cpp
Header File	.h

Project: VDK_Test.dpj

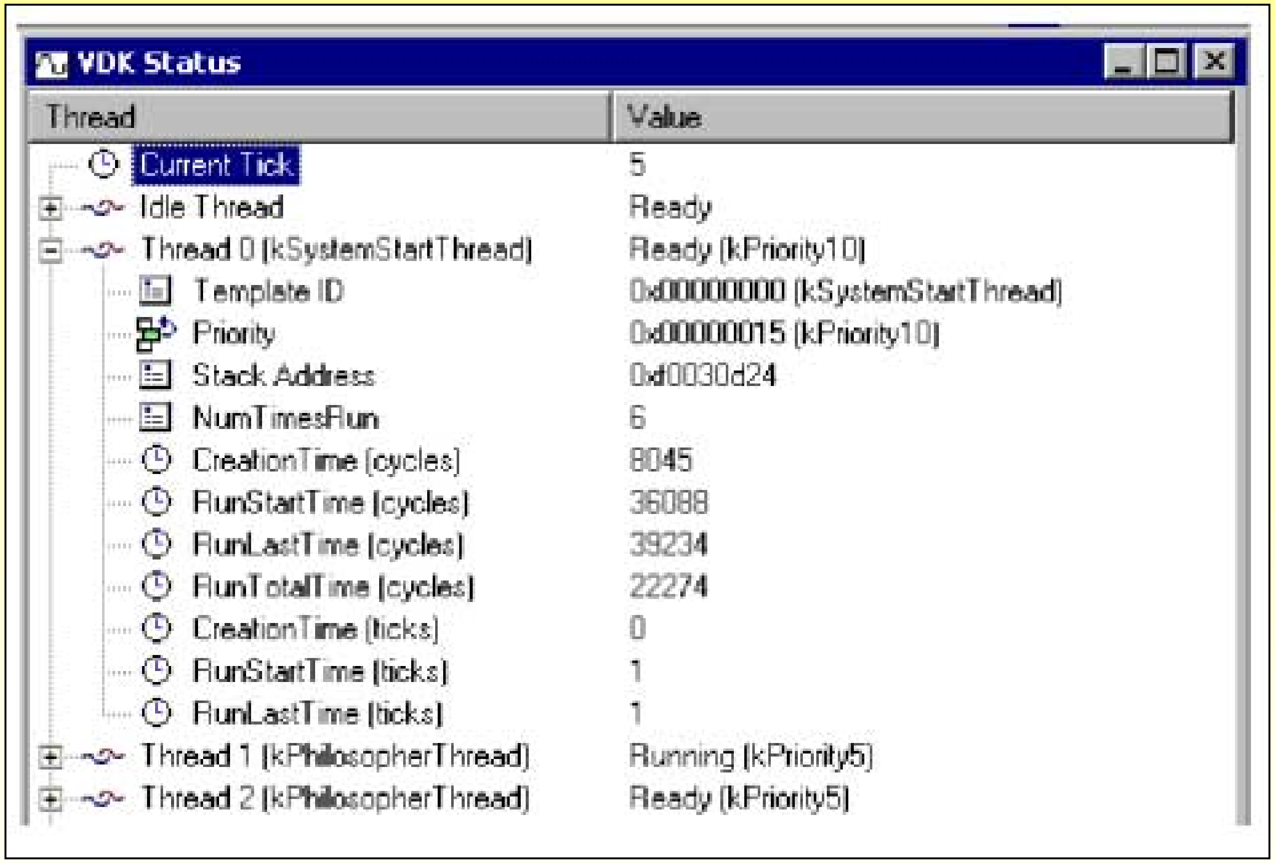
Parameter	Value
Kernel	
System	
Thread Types	
Boot Threads	
Round-Robin Priorities	
Semaphores	
Event Bits	
Events	
Interrupts	
Device Drivers	
MyDeviceDriver	
Source File	MyDeviceDriver.cpp
Header File	MyDeviceDriver.h
Device Flags	
MyFlag1	
MyFlag2	
MyFlag3	

Embedded DSP: Visual DSP++ -- Kernel (VDK)


An arrow is a scroll bar
Data cursor
A horizontal bar indicates a thread
Current data cursor position
A thin green vertical line indicates a thread switch
Arrows indicate thread events
State History
Thread 4 (kPhilosopherThread)
Thread 3 (kPhilosopherThread)
Thread 2 (kPhilosopherThread)
Thread 1 (kPhilosopherThread)
Thread 0 (kPhilosopherThread)
Idle Thread
Ticks
Enable the data cursor to view the status bar
The status bar provides event details
Thread status
The thin green line below a thread status bar indicates the currently active thread
The green portion of the horizontal bar indicates the time the thread was active
Trace thread switch history by following the thin green line

Fraunhofer Institut Zerstörungsfreie Prüfverfahren

Embedded DSP: Visual DSP++ -- Kernel (VDK)



VDK Status

Thread	Value
Current Tick	5
Idle Thread	Ready
Thread 0 (kSystemStartThread)	Ready (kPriority10)
Template ID	0x00000000 (kSystemStartThread)
Priority	0x00000015 (kPriority10)
Stack Address	0xd0030d24
NumTimesRun	6
CreationTime (cycles)	8045
RunStartTime (cycles)	36088
RunLastTime (cycles)	39234
RunTotalTime (cycles)	22274
CreationTime (ticks)	0
RunStartTime (ticks)	1
RunLastTime (ticks)	1
Thread 1 (kPhilosopherThread)	Running (kPriority5)
Thread 2 (kPhilosopherThread)	Ready (kPriority5)

Embedded DSP: Visual DSP++ -- Using the VDK

Thread Types:
- Divide an application into threads which each operate independant from the others.
- Definiton of thread TYPES.
- A thread is an instance of a thread type (C-Structure) and every variable of the structure is a thread.
- Multiple instantiations of a thread is possible.
- Each thread has ist own stack, state, priority and other local variables.

Thread Parameters:
- Each thread is identified by:
    - ThreadID (Access by GetThreadID())
    - Valid for the life of the thread.
    - When destroyed then it becomes invalid.

Thread Size:
- System allocates space in the heap when created for the thread-specific parameters.
- Hold information for the kernel and the thread type specifications by the user.

Thread Priority:
- Each thread has a specific priority at the start.
- Threads can change their and other priorities by: SetPriority(), ResetPriority(), ...

Embedded DSP: Visual DSP++ -- Using the VDK

Run Functions:
- Each thread is required to have 5 functions declared and implemented.

- **Run Function**: entry point (for many thread types the only necessary function)
    - in C: RunFunction();   (equivalent to main())
    - When started the thread is moved to the queue of threads waiting to free resources.

- **Error Function**: Called by the kernel when an error occurs in the API call made by the thread.

- **Create Function**: in C: CreateThread() is similar to the constructor.
    - The first function for creating a thread for spawning a new process, performs the space allocations of the structure, return the pointer the thread.
    - Can also be created as a boot thread !

- **Init Function/Constructor**: in C: InitFunction();
    - Provide a place for the thread to allocate system resources during the dynamic thread creation.
    - A thread use maloc() for allocating the local variables of the thread.

- **Destructor**: DestroyThread():
    - Is called when the thread should be destroyed.

Embedded DSP: Visual DSP++ -- Using the VDK

Global Variables:

- VDK applications can use global variables as normal variables.
- Global variables are declared in one file and are extern on other files.
- Critical and/or unscheduled regions should be used to protect operations on independant variables that can leave the system in an undefined state if not completeld atomic.

Error Handling:

- When an error occurs, the system calls the user-implemented ErrorFunction().
- GetLastThreadErrorValue()
- GetLastThreadError(); manages intelligently the value from the above function.

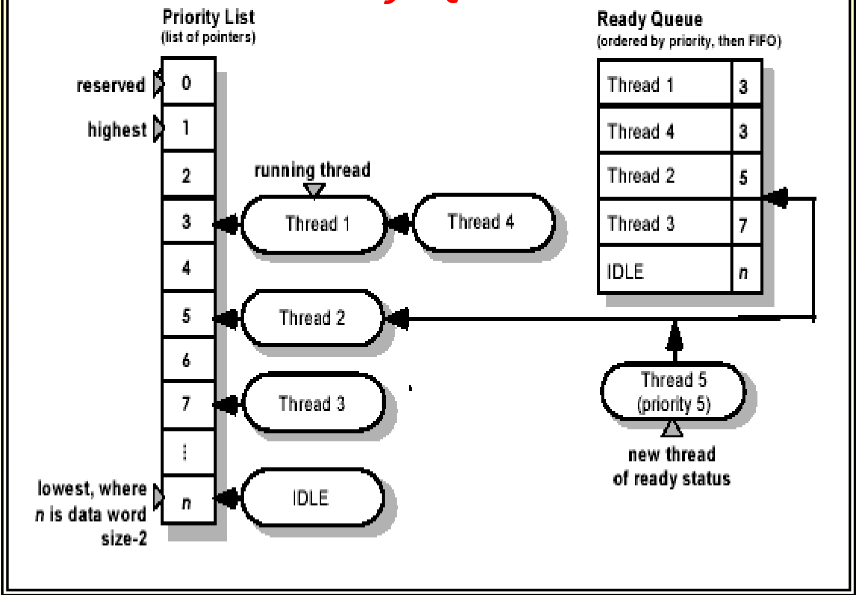
Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Scheduling:

- Ensures that the highest-priority ready thread is allowed to run at the earliest possible time.
- Is activated whenever a kernel API called from either a thread or an ISR changes the highest priority.
- The scheduler is not invoked during critical or unscheduled regions but can be invoked immediately at the close of either type of protected region.
- Scheduler works on a data structure ready queue.
- The queue hold reference to all threads that are not sleeping or blocked.
- Arranged as a prioritized FIFO buffer, when a thread is moved to the ready queue, it is added as the last entry at its priority.

Works as a pre-epmtive kernel with additional possibilities for scheduling !

Ready Queue



Fraunhofer Institut Zerstörungsfreie Prüfverfahren

IZFP

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Additional Scheduling Methodoligies:

Cooperative Scheduling:

• Multiple threads may be created at the same priority level.
• Simplest solution: All threads in the system are given the same priority, and each of the threads has access to the processor until it manually yields control.

$\Rightarrow$ ***Cooperative Multiprocessing***

• When a thread is ready to move to the next thread in the FIFO, the thread can do this by calling the function yield(), which places the curently running thread to the end of the list.

• Example: If a thread pends on a signal which momentarily is not available, the next thread in the queue with the same priority starts executing.

Embedded DSP: Visual DSP++ -- Kernel (VDK)
Using the VDK

Additional Scheduling Methodoligies:

Round-Robin Scheduling:

- Also called time slicing, allows multiple threads with the same priority to have processor time automatically in fixed time slots (depends on the mode at build time and the periods specified in system ticks).

- Threads at the given priority should to be run for that duration as measured by the number of timer interrupts.

- If the thread is pre-empted by a thread with a higher priority for a time tx, the time is not subtracted from the time slice.

- When the period of a thread completes, it is moved to the end of the list of threads according to the priority in the ready queue.

Problems: Leads to jitter when threads at that priority are pre-empted.

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Additional Scheduling Methodoligies:

Pre-Emptive Scheduling:

• Full pre-emptive scheduling means, that a thread gets processor time as soon as it is placed in the ready queue and it has a higher priority than the running thread.

• Provides more power and felxibility than simple cooperative or round-robin scheduling.

VDK allows all three methods without any model configuration.

Example: A multiple non-time critical thread can be set to low-priority with round-robin method, in order to secure that each thread gets processor time without interfering with time-critical threads.

Addiotionally a thread can yield the processor at any time, allowing another thread to run.

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Disabling Scheduling:

- Unscheduled areas (similar to critical ...) are sections of code that execute without being pre-empted by a higher-priority thread.

- Interrupts are always serviced in an unscheduled region, but the same thread runs after return to the thread domain.

- Unscheduled regions are activated by a call to:
    - Entry: PushUnscheduledRegion()
    - Leave: PopUnscheduledRegion()

- Unscheduled regions are implemented with a stack.

Entering the Scheduler from API Calls:

- The scheduler needs to be called only when a higher-priority thrad becomes ready.

- By the interaction with API calls the time for ready threads changes is well defined.

- A thread invokes the scheduler only then when a thread changes the highest-priority ready thread or exits an unscheduled section and the highest-priority ready threads has changed.

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Scheduler and Interrupts:

The interrupt service routines should be written in assembler to decrease the number of the context switches and the interrupt latency time.

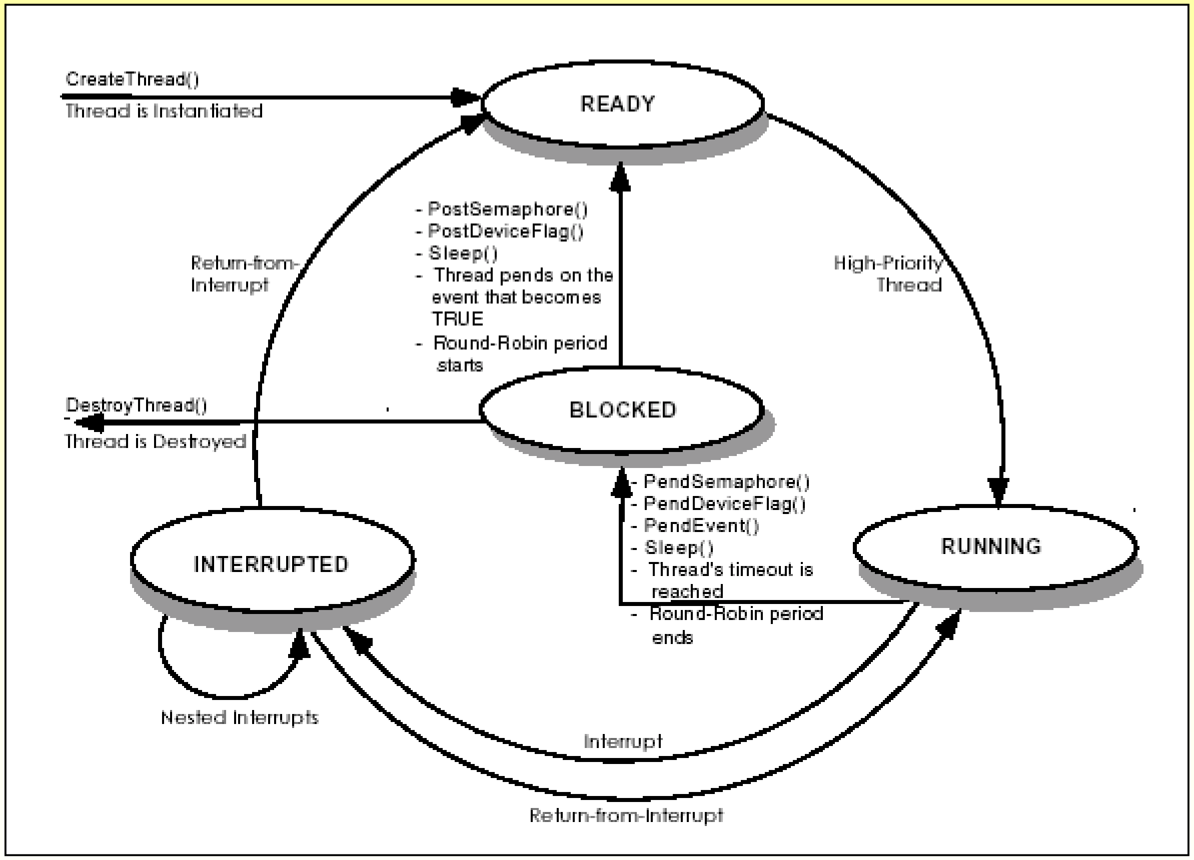
Depending on the system state, an ISR API call may demand the scheduler being executed.

Therefore VDK reserves the lowest-priority interrupt to handle the reschedule process.

Idle Thread:

- The idle thread is a predefined automatically created thread with a priority lower than the priority of any user threads.
- When no user threads are in the ready queue the idle thread is runs.
- The idle threads handles destruction of threads that were passed to the function DestroyThread().

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK



Fraunhofer Institut Zerstörungsfreie Prüfverfahren

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Signals

- Threads use three different methods for communication and synchronisation:
  - Semaphores
  - Events
  - Device Flags
- If a signal is not available, the thread blocks until the signal becomes available.
- Optionally a timeout is reached.

Semaphores

- Semaphores are protocol methods offered by most operating systems. Semaphores are used to:
  - Control and manage access to a shared resource.
  - Signal a special system occurence.
  - Allow two threads to communicate.
  - Schedule periodic execution of threads.

  - The number and predefined state of semaphores is set up when your project is build.

Embedded DSP: Visual DSP++ -- Kernel (VDK) Using the VDK

Behavior of Semaphores

- A Semaphore is a token that a thread can acquire in order to continue execution.
- If a semaphore is accessed by another thread the pending on a semaphore thread is blocked until the semaphore is available.
- If the semaphore is not available in the specified time, the thread continues execution with its error function.
- Semaphores are global structures that are accessible to all threads.
- When a semaphore is posted, the thread with the highest-priority that has been waiting the for longest time is moved to the ready queue.
- Semaphores are not owned in VDK (unlike many operating systems).
- Any thread is allowed to post a semaphore (make it avaliable).

- Normally operates a semaphore as a flag between threads.
- Besides a semaphore can be set up to have a periodic behavior.
    - A periodic semaphore is available every n ticks, with n is the period of the semaphore.
    - Usable to secure that a thread is run at regular time intervals.

Next Lecture:

- VDK-SHARC Example
- OSEK