Chapter 11

Real-time Analysis and Scheduling

Introduction

To develop and analyze embedded real-time software, TI includes DSP/BIOS with CCS. The main elements of DSP/BIOS are:

- A small firmware real-time library
- DSP/BIOS application programming interface (API) for using real-time library services
  - The APIs are modular
- Easy to use tools for configuration, real-time tracing, and analysis

Features

- All DSP/BIOS objects can be created with the *Configuration Tool*, with definitions saved in a file `*.cdb`
  - This tool generates all code required to declare objects used within the program, including the linker command file `*.cmd` and `vectors.asm`
- Only the API modules used need to be bound into the program

- A significant portion of the modules are in assembly
- Communication between the target and host is performed with a *background idle loop*
  - Logging and statistics for BIOS objects are available at run time without additional programming
  - BIOS analysis tools allow real-time monitoring of program behavior
- Thread types are provided for:
  - Hardware interrupts, software interrupts
  - Tasks
  - Idle functions, periodic functions
- Priorities can be controlled as well as blocking characteristics
- Structures are provided that support communication and synchronization between threads
  - Semaphores, mail boxes, and resource locks
- Two I/O models are available:
  - Pipes for target/host communication and reading and writing from threads
  - Streams can be used for more complex I/O and to support device drivers
- The *Chip Support Library* allows for easier device programming, e.g., register level programming, and is portable across different DSP platforms

The DSP/BIOS API Modules

Instrumentation/Real-Time Analysis

LOG	Message Log manger
STS	Statistics accumulator manager
TRC	Trace manager
RTDX	Real-Time Data Exchange manager

Thread Types

HWI	Hardware interrupt manager
SWI	Software interrupt manager
TSK	Multitasking manager
IDL	Idle function & processing loop manager

Clock and Periodic Functions

CLK	System clock manager
PRD	Periodic function manger

Chip Support Library

CSL	Easier device (register) programming

Comm/Synch between threads

SEM	Semaphores manager
MBX	Mailboxes manager
LCK	Resource lock manager

BIOS API Modules (cont.)

Input/Output

PIP	Data pipe manager
HST	Host input/output manager
SIO	Stream I/O manager
DEV	Device driver interface

Memory and Low-level Primitives

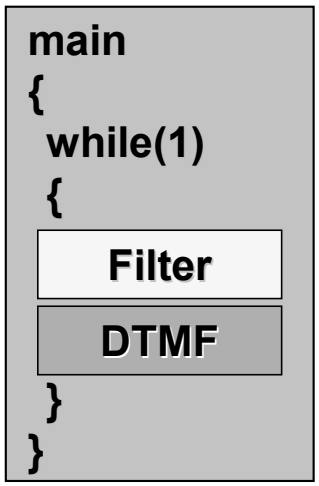
MEM	Memory manager
SYS	System services manager
QUE	Queue manager
ATM	Atomic functions
GBL	Global setting manager

A Case Study: Audio Player with DTMF[1]

- An audio DSP application that filters an audio stream is being enhanced to include a DTMF generator with keypad entry
- Design issues that need to be considered are:
  - Do we have enough bandwidth (MIPS)?
  - Will one routine conflict with the other?
  - How do we create the compound system?

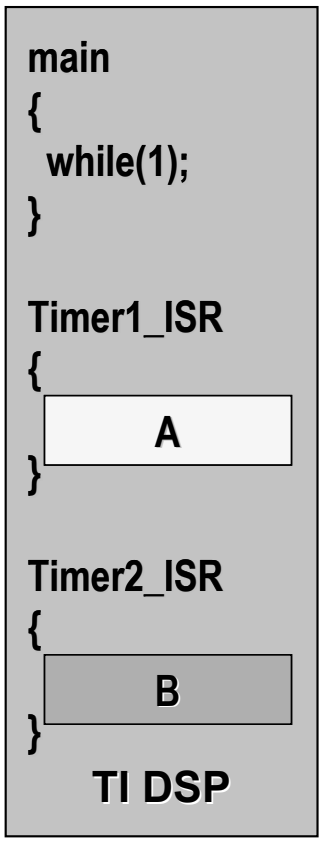
1. This example is taken from TI DSP/BIOS lecture material

Run them together under `main()`**:**

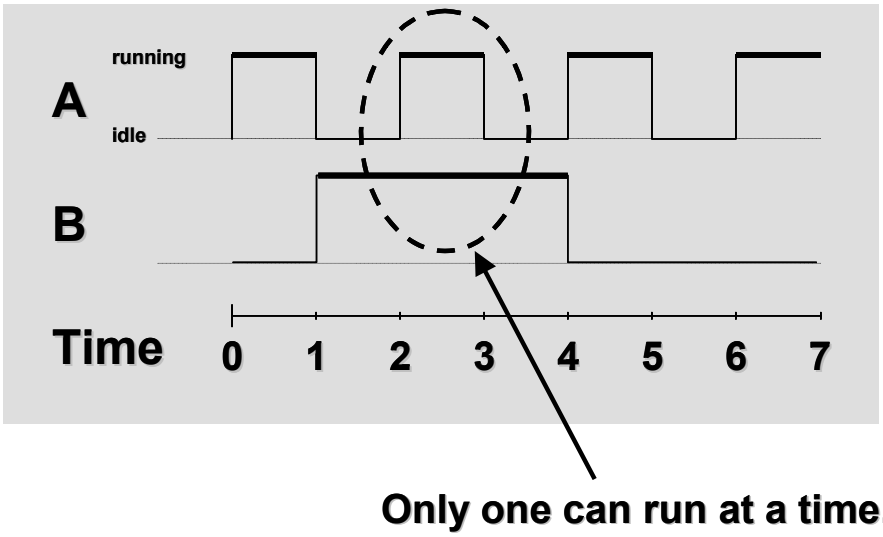


- What if algorithms run at differing rates? (e.g.: our filter runs ~ 44 KHz and the DTMF algorithm ~ 8 KHz)
- What if one algorithm overshadows another, starving it for recognition or delaying it's response beyond the limits of the system?

A second solution is to use two interrupts under `main()`**:**



	Period	Compute	CPU Usage
Routine A:	22 µs	11 µs	(50%)
Routine B:	125 µs	33 µs	(26%)
			76%

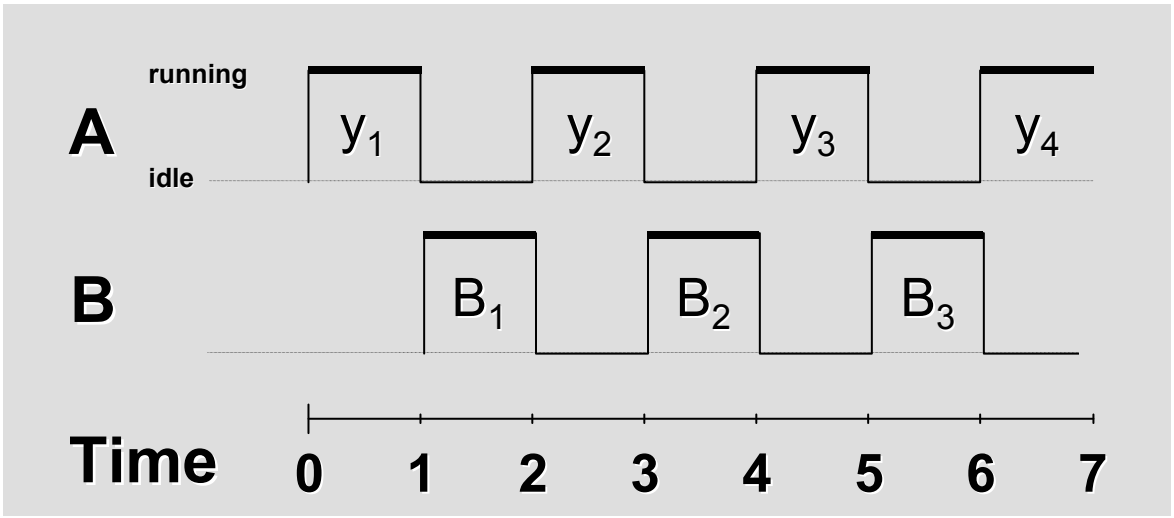


- We need to consider both average and instantaneous CPU loading

Interrupt driven state machine

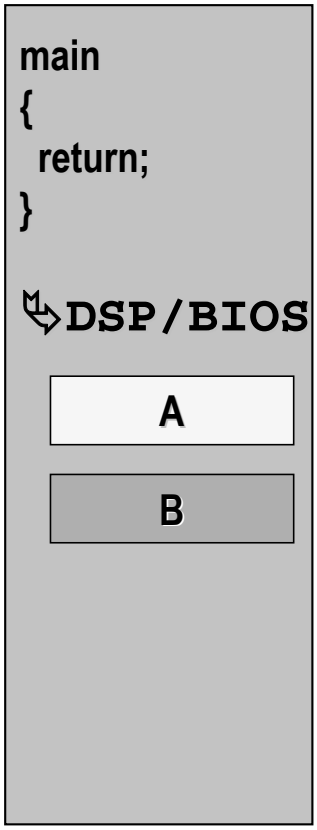
```
main
{
 if tick>set1
  if …
     A
  else if
   if …
    B (part 1)
   else if ...
    B (part 2)
   else …
    B (part 3)
...
}
```

- To solve this scheduling problem, consider building a state-machine in the `main()` routine
  - Difficult and tedious to write; Need to keep track of various execution times and paths through software
  - Difficult to maintain; Code is too tightly coupled to allow any changes or updates
  - Can be slow and large; Conditional statements lead to branching operations and disruptions in normal software flow



- The use of C main() background functions has the problems of:
  - No Guarantee of Concurrency
  - Non-deterministic timing
  - No Software Preemption
  - Ad Hoc Analysis

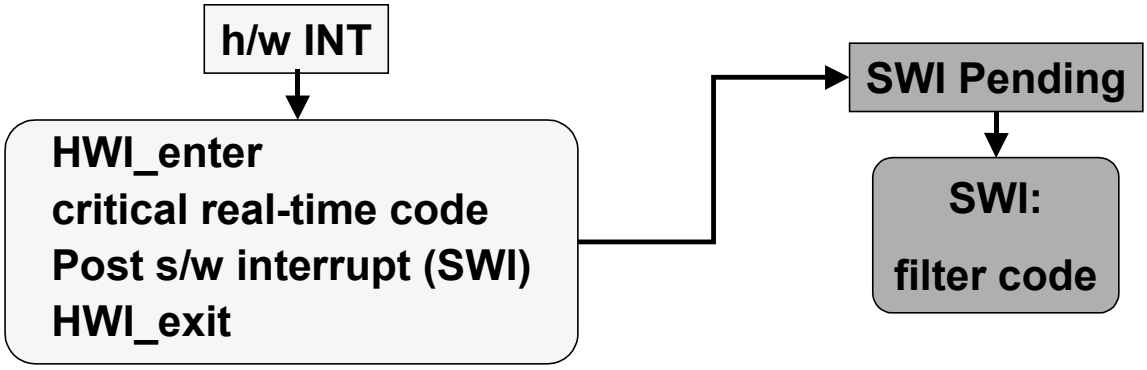
The DSP/BIOS Solution



- DSP/BIOS provides scheduling:
  - You needn't build a custom (inflexible) state-machine for each DSP design
  - Easy to write - Modules written independently
  - Easy to maintain - Module interaction minimized
  - Built-in Scheduling - Managed by DSP/BIOS

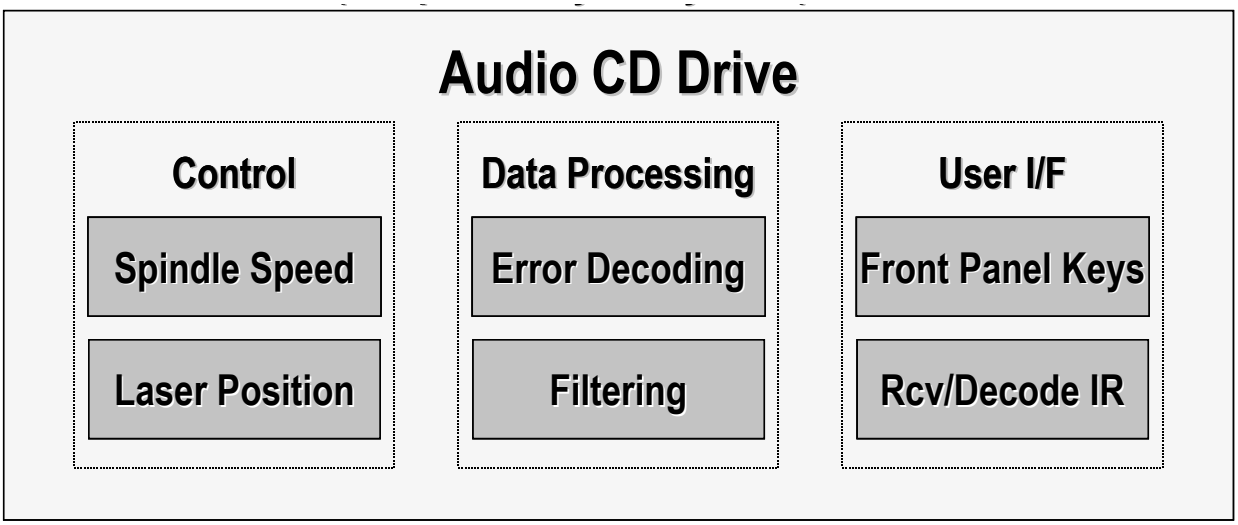
- DSP/BIOS allows both hardware (HWIs) and software interrupts (SWIs)
  - HWIs implement 'urgent' portion of real-time event
  - SWIs perform 'follow-up' activity

  - SWIs are ‘posted’ to run by HWIs or other SWIs
- The DSP/BIOS scheduler provides both HWI and SWI management

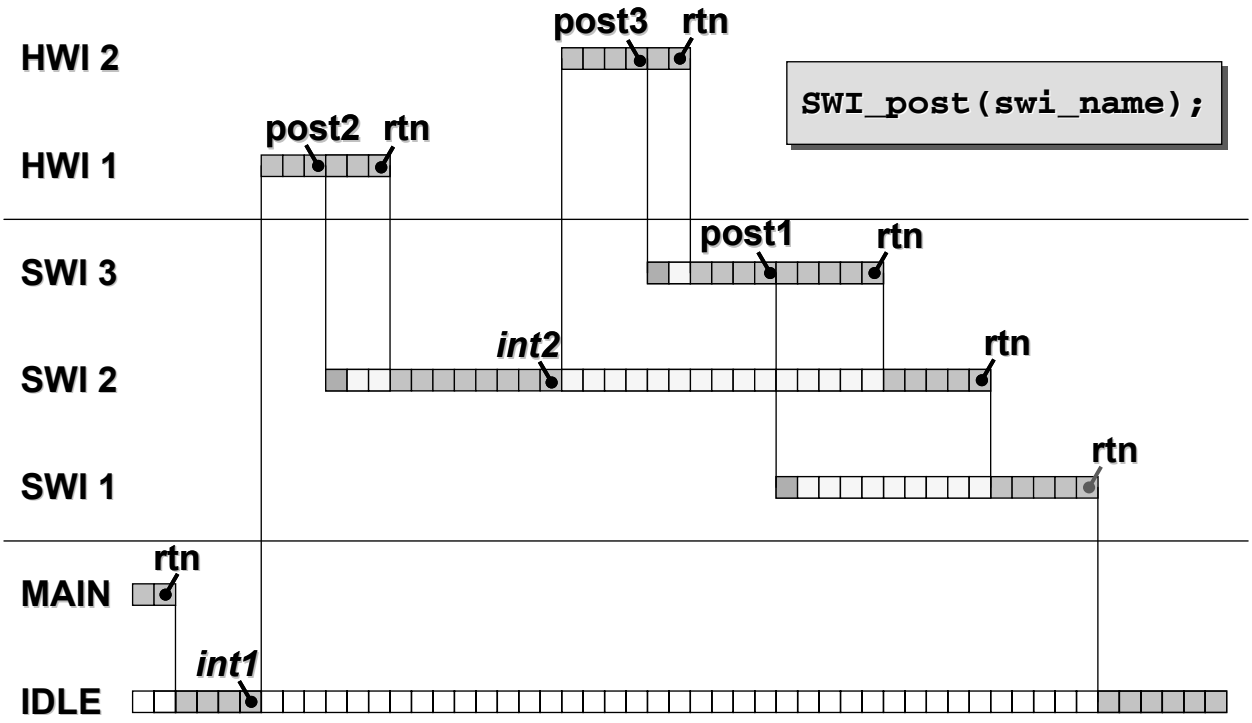


- HWI features:
  - Fast response to interrupts
  - Minimal context switching
  - High priority for CPU
  - Can post SWI
  - Danger of missing an interrupt while executing ISR
- SWI features:
  - Latency in response time
  - Context switch performed
  - Selectable priority levels
  - Can post another SWI
  - Execution managed by scheduler

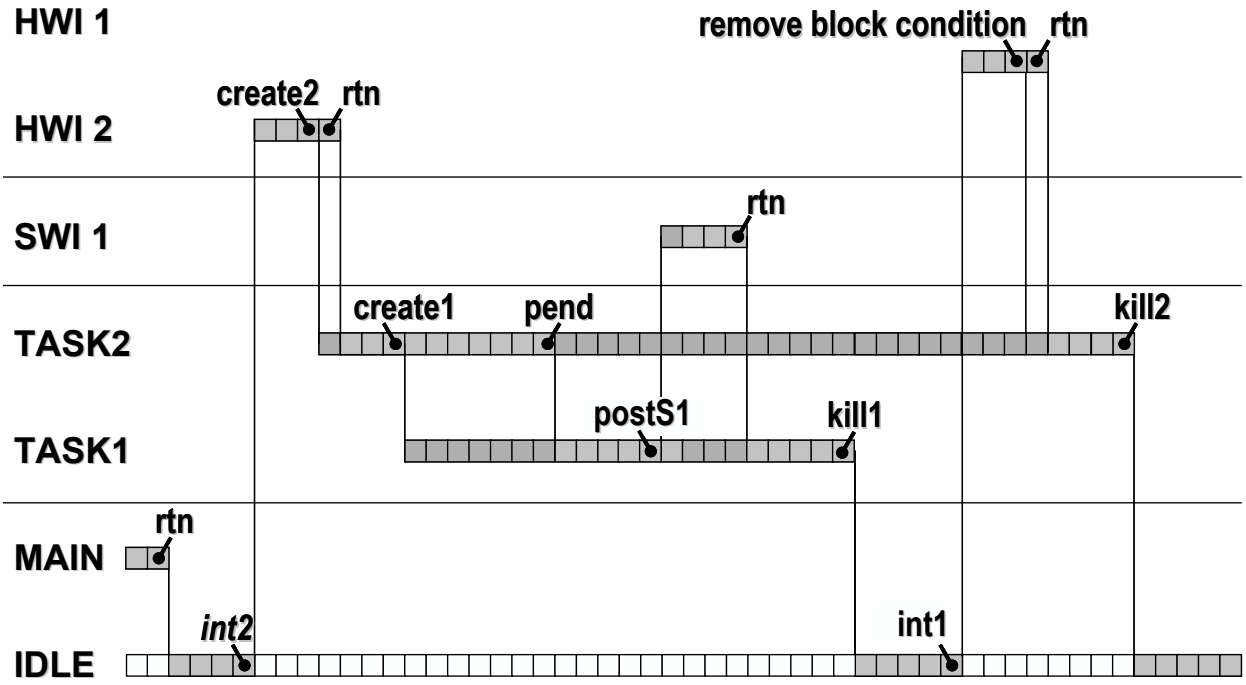
- Typical routines required in an audio CD drive:



- A graphical view of scheduling



- Another graphical example showing tasks (TSK) which was added in DSP/BIOS II, and included in CCS 2



Getting Started with DSP/BIOS

- To get started with DSP/BIOS we will consider the instrumentation/real-time analysis module, which in particular includes
  - LOG, the message log manger
  - STS, the statistics accumulator manager
-