Blackfin System Services

Presented By:
David Lannigan
About this Module

This module discusses the System Services software available for the Blackfin family of processors.

It is recommended that users should have some understanding of the Blackfin architecture, a basic knowledge of software terminology and experience in embedded systems.
Module Outline

- **Overview**
  - What are system services?
  - Benefits of using system services
- **Highlight functionality of each service**
  - Dynamic Power Management
  - External Bus Interface Unit (EBIU)
  - Interrupt Manager
  - Deferred Callback Service
  - DMA Manager
  - Flag Control
  - Timer Control
  - Port Control
- **Simple examples using the services**
What are System Services?

- **Software library**
  - Provides functionality common to embedded systems
    - Simple, efficient access into
      - PLL, DMA, interrupt controllers, timers, flags etc.
    - Improved interrupt performance
      - Deferred callbacks
  - Callable from ‘C’ or assembly

- **Common APIs across Blackfin processors**
  - ADSP-BF531, BF532, BF533, BF534, BF536, BF537
  - ADSP-BF561

- **Leveraged by applications, device drivers etc.**
  - Standalone environment
  - VDK environment
Benefits To Using System Services

- Faster time to market
  - Tested and proven software
  - Shorter learning curve
  - Less re-invention
- Modular software
  - Better compatibility
  - Simplifies integration efforts
- Portability
  - APIs identical across Blackfin processors
    - Both single-core and multi-core processors
  - Leverage processor roadmap
    - Transition quickly to new processors
- Access to device driver portfolio
  - ADI device drivers built on top of system services
- Full source code provided
System Architecture

Application

RTOS (optional)

Device Driver

Device Driver

Device Driver

System Services
System Services

- EBIU (SDRAM)
- Dynamic Power
- Interrupt Manager
- DMA Manager
- Deferred Callback
- Flag Control
- Timer Control
- Port Control
Dynamic Power Management Service

- Controls Phase-Locked Loop (PLL) and internal voltage regulator
- **Single function call to**
  - Change operating modes
    - Full-on, active, sleep, deep sleep, and hibernate
  - Change core and system clock frequencies (CCLK and SCLK)
    - Clock frequency priority
    - Voltage level automatically adjusted accordingly
      - Lowered whenever possible to maximize power savings
      - Raised when performance requires
    - Maximize CCLK and SCLK for given voltage level
      - Voltage level priority
      - Clocks raised to max safe frequency
- Automatically controls EBIU service
External Bus Interface Unit (EBIU)

- Initializes SDRAM settings
  - Configures SDRAM controller
- Logic to calculate new values
  - Changing SCLK frequencies
- Works in concert with Power Management Service
  - Automatically adjusts settings for SCLK frequency changes
Using EBIU and Power Management

- Application should
  - Initialize the EBIU service
  - Initialize the Power Management service
  - Call Power Management functions as needed
- Such as
  - adi_pwr_SetFreq();
  - adi_pwr_MaxFreqForVolt();
  - adi_pwr_SetPowerMode();
Dynamic Power Management API

Initialization/Termination

ADI_PWR_RESULT adi_pwr_Init(); // Initializes the power service
ADI_PWR_RESULT adi_pwr_Terminate(); // Terminates the power service

Frequency and Voltage Control

ADI_PWR_RESULT adi_pwr_Control(); // Sets/queries a configuration parameter
ADI_PWR_RESULT adi_pwr_SaveConfig(); // Saves the power configuration
ADI_PWR_RESULT adi_pwr_LoadConfig(); // Loads a power configuration

ADI_PWR_RESULT adi_pwr_SetVoltageRegulator(); // Adjusts the internal voltage regulator
ADI_PWR_RESULT adi_pwr_SetMaxFreqforVolt(); // Set the max clock freqs for voltage level
ADI_PWR_RESULT adi_pwr_SetFreq(); // Sets the core and system clock frequencies
ADI_PWR_RESULT adi_pwr_AdjustFreq(); // Adjusts the core and system clock frequencies
ADI_PWR_RESULT adi_pwr_GetFreq(); // Gets the core and system clock frequencies

Operating Modes

ADI_PWR_RESULT adi_pwr_SetPowerMode(); // Place processor in specified operating mode
ADI_PWR_RESULT adi_pwr_GetPowerMode(); // Gets the current operating mode
EBIU Service API

Initialization/Termination

ADI_EBIU_RESULT adi_ebiu_Init();  // Initializes the EBIU service
ADI_EBIU_RESULT adi_ebiu_Terminate(); // Terminates the EBIU service

EBIU Control

ADI_EBIU_RESULT adi_ebiu_Reset();  // Resets the EBIU module to power-up settings
ADI_EBIU_RESULT adi_ebiu_Control();  // Sets/queries module settings
ADI_EBIU_RESULT adi_ebiu_AdjustSDRAM(); // Recalculates and apply settings for SCLK changes

EBIU Configuration

ADI_EBIU_RESULT adi_ebiu_LoadConfig();  // Loads a set of controller settings
ADI_EBIU_RESULT adi_ebiu_SaveConfig() // Saves a set of controller settings
Dynamic Power Example

- Change frequencies, voltage, operating mode
- Dynamic Power Service
  - Phase Locked Loop (PLL)
  - Voltage Regulator
  - EBIU Service
    - SDRAM
    - Adjust timings
Interrupt Manager Service

- **Core Event Controller (CEC)**
  - Hook/Unhook interrupt handlers into Interrupt Vector Groups (IVG)
  - Supports handler chaining
- **System Interrupt Controller (SIC)**
  - Mapping of peripheral interrupts to IVG
  - Enable/disable passing to core event controller
  - Enable/disable wakeup of core event controller
- **Utility functions**
  - Critical region protection
  - Interrupt Mask Register (IMASK) control
Interrupt Manager API

Initialization/Termination

ADI_INT_RESULT adi_int_Init(); // Initializes the interrupt manager
ADI_INT_RESULT adi_int_Terminate(); // Terminates the interrupt manager

Core Event Controller Functions

ADI_INT_RESULT adi_int_CECHook(); // Hooks a handler into an IVG chain
ADI_INT_RESULT adi_int_CECUnhook(); // Unhooks a handler from an IVG chain

System Interrupt Controller Functions

ADI_INT_RESULT adi_int_SICEnable(); // Allows interrupt to be passed to the CEC
ADI_INT_RESULT adi_int_SICDisable(); // Disallows interrupt to be passed to the CEC
ADI_INT_RESULT adi_int_SICSetIVG(); // Sets the IVG to which a peripheral is mapped
ADI_INT_RESULT adi_int_SICGetIVG(); // Gets the IVG to which a peripheral is mapped
ADI_INT_RESULT adi_int_SICWakeup(); // Allows the interrupt to wakeup the processor
ADI_INT_RESULT adi_int_SICInterruptAsserted(); // Tests if an interrupt is asserted

Utility Functions

void *adi_int_EnterCriticalRegion(); // Enters a critical region of code
void adi_int_ExitCriticalRegion(); // Exits a critical region of code
void adi_int_SetIMaskBits(); // Sets bits in IMASK register
void adi_int_ClearIMaskBits(); // Clears bits in IMASK register
Interrupt Manager Example

Main program

Interrupt Handler

Interrupt Manager

RTC

Minute Interrupt

System Interrupt Controller (SIC)

Core Event Controller (CEC)
What’s a Callback?

- Call made to a function outside the normal flow of program execution
  - Response to an asynchronous event (hardware interrupt)
- Callback Function
  - Regular ‘C’ callable function
  - Takes some action based on the event
- Types of Callbacks
  - **Live**
    - Call to the function is made immediately
    - Callback function typically executes at hardware interrupt time
    - Negative impact to performance (higher interrupt latency)
  - **Deferred**
    - Call to the function is deferred to some later point in time
    - Callback function executes at software interrupt time
    - Positive impact to performance (lower interrupt latency)
Deferred Callback Manager

- Reduce time spent in hardware Interrupt Service Routines (ISR)
  - Service hardware, queue the callback and exit
- Map callback services to different IVG levels
  - User specifies level(s)
    - Typically lower than hardware levels
- Prioritization within each level
  - Urgent callbacks processed first
- Operating environments
  - Standalone systems
    - Callbacks execute before “normal” user code
  - VDK based systems
    - Callbacks run at software interrupt thread
Deferred Callback Service API

Initialization/Termination

ADI_DCB_RESULT adi_dcb_Init(); // Initializes the deferred callback service
ADI_DCB_RESULT adi_dcb_Terminate(); // Terminates the deferred callback service

Callback Queue Server Control

ADI_DCB_RESULT adi_dcb_Open(); // Opens a callback queue
ADI_DCB_RESULT adi_dcb_Close(); // Closes a callback queue
ADI_DCB_RESULT adi_dcb_Control(); // Changes settings of a callback queue
ADI_DCB_RESULT adi_dcb_Post(); // Posts a callback to a queue
ADI_DCB_RESULT adi_dcb_Remove(); // Removes callbacks from a queue
DMA Manager

- Controls and schedules DMA
  - Supports both peripheral and memory DMA
  - User control of DMA channel mappings/priority, traffic control
- Comprehensive support for DMA modes
  - Descriptor chaining (large, small)
    - Queues descriptor jobs
  - Autobuffering (called circular buffers)
- MemCopy functions
  - DMA transfers rather than core access
  - One dimensional and two dimensional
- Optional callbacks on completion
  - Live or deferred
DMA Manager API

Initialization/Termination

ADI_DMA_RESULT adi_dma_Init(); // Initializes the DMA manager
ADI_DMA_RESULT adi_dma_Terminate(); // Terminates the DMA manager

Channel Control

ADI_DMA_RESULT adi_dma_Open(); // Opens a channel for use
ADI_DMA_RESULT adi_dma_Close(); // Closes a channel
ADI_DMA_RESULT adi_dma_Control(); // Configures a channel
ADI_DMA_RESULT adi_dma_Queue(); // Posts a chain of descriptors to a channel
ADI_DMA_RESULT adi_dma_Buffer(); // Provides a one-shot or circular job to a channel

Memory Stream Control

ADI_DMA_RESULT adi_dma_MemoryOpen(); // Opens a memory stream
ADI_DMA_RESULT adi_dma_MemoryClose(); // Closes a memory stream
ADI_DMA_RESULT adi_dma_MemoryCopy(); // Performs a one-dimensional memory transfer
ADI_DMA_RESULT adi_dma_MemoryCopy2D(); // Performs a two-dimensional memory transfer

Channel Mappings

ADI_DMA_RESULT adi_dma_SetMapping(); // Sets the mapping of a channel to a peripheral
ADI_DMA_RESULT adi_dma_GetMapping(); // Gets the mapping of a channel to a peripheral
Memory DMA Example

Main program

Open memory stream and copy data

Source

DMA Manager

Callback Function

Callback upon completion

Destination

Do another copy
Flag Control Service

- **Controls general purpose programmable flags (GPIO)**
  - All hardware capabilities exposed
    - Set direction
    - Set/clear/toggle level
    - Sense level
- **Provides callback capability**
  - Callback function invoked upon trigger condition
    - Level sensitive
      - High/low
    - Edge sensitive
      - Rising/falling/either
  - Live or deferred
Flag Control API

Initialization/Termination

```c
ADI_FLAG_RESULT adi_flag_Init(); // Initializes the flag service
ADI_FLAG_RESULT adi_flag_Terminate(); // Terminates the flag service
```

Flag Control

```c
ADI_FLAG_RESULT adi_flag_Open(); // Opens a flag for use
ADI_FLAG_RESULT adi_flag_Close(); // Closes a flag
ADI_FLAG_RESULT adi_flag_SetDirection(); // Configures the flag for input or output
ADI_FLAG_RESULT adi_flag_Set(); // Sets a flag to logical 1
ADI_FLAG_RESULT adi_flag_Clear(); // Sets a flag to logical 0
ADI_FLAG_RESULT adi_flag_Toggle(); // Toggles the current value of a flag
ADI_FLAG_RESULT adi_flag_Sense(); // Senses the value of a flag
```

Flag Callback Control

```c
ADI_FLAG_RESULT adi_flag_InstallCallback(); // Installs a callback for sensing flag changes
ADI_FLAG_RESULT adi_flag_RemoveCallback(); // Removes a callback from a flag
ADI_FLAG_RESULT adi_flag_SetTrigger(); // Sets the trigger condition for a flag callback
ADI_FLAG_RESULT adi_flag_SuspendCallbacks(); // Temporarily suspend callbacks for a flag
ADI_FLAG_RESULT adi_flag_ResumeCallbacks(); // Resume callbacks for a flag
```
Flag Control Example

Configure flag as an input and install callback

Main program

Flag Control

Callback Function

Callback upon trigger condition

Programmable Flag

Pushbutton on EZ-Kit
Timer Control Service

◆ Controls operation of timers
  ● Full access into all modes and features
    ◆ Core timer
      ● Count, period, scale, auto-reload
    ◆ Watchdog timer
      ● Select timeout event, reset counter
  ◆ General purpose timers
    ● PWM, WidthCap
    ● Simultaneous enable/disable

◆ Provides callback capability
  ● Callback function upon timer expiration
  ● Live or deferred
Timer Control API

Initialization/Termination

ADI_TMR_RESULT adi_tmr_Init(); // Initializes the timer service
ADI_TMR_RESULT adi_tmr_Terminate(); // Terminates the timer service

Timer Control

ADI_TMR_RESULT adi_tmr_Open(); // Opens a timer for use
ADI_TMR_RESULT adi_tmr_Close(); // Closes a timer
ADI_TMR_RESULT adi_tmr_Reset(); // Resets a timer to power-up settings
ADI_TMR_RESULT adi_tmr_GetPeripheralID(); // Gets the peripheral ID for a timer

ADI_TMR_RESULT adi_tmr_CoreControl(); // Controls the core timer
ADI_TMR_RESULT adi_tmr_WatchdogControl(); // Controls the watchdog timer
ADI_TMR_RESULT adi_tmr_GPControl(); // Controls a general purpose timer
ADI_TMR_RESULT adi_tmr_GPGroupEnable(); // Simultaneously enables/disables a group of timers

Timer Callback Control

ADI_TMR_RESULT adi_tmr_InstallCallback(); // Installs a callback for a timer
ADI_TMR_RESULT adi_tmr_RemoveCallback(); // Removes a callback from a timer
Port Control Service

- Controls assignment of muxed pins
  - Applicable to ADSP-BF534, ADSP-BF536, ADSP-BF537 only
- Operation largely transparent to applications
  - Application need only initialize port control
  - No other application involvement required
- Automatically accessed by drivers and other services
  - Examples
    - PPI driver
      - Data width, frame sync pins etc.
    - Timer service
      - Input clocks/output signals
    - Flag service
      - Configures as appropriate for flag pins
Port Control API

Initialization/Termination

ADI_PORTS_RESULT adi_ports_Init(); // Initializes the port control service
ADI_PORTS_RESULT adi_ports_Terminate(); // Terminates the port control service

Peripheral Based Control

ADI_PORTS_RESULT adi_portsEnablePPI(); // Configures pins for PPI operation
ADI_PORTS_RESULT adi_portsEnableSPI(); // Configures pins for SPI operation
ADI_PORTS_RESULT adi_portsEnableSPORT(); // Configures pins for SPORT operation
ADI_PORTS_RESULT adi_portsEnableUART(); // Configures pins for UART operation
ADI_PORTS_RESULT adi_portsEnableCAN(); // Configures pins for CAN operation
ADI_PORTS_RESULT adi_portsEnableTimer(); // Configures pins for timer operation
ADI_PORTS_RESULT adi_portsEnableGPIO(); // Configures pins for flag operation

Profile Based Control

ADI_PORTS_RESULT adi_ports_SetProfile(); // Sets a muxing profile
ADI_PORTS_RESULT adi_ports_GetProfile(); // Gets a muxing profile
Finding the System Services

◆ Include files
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\include\services

◆ Source files
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\lib\src\services

◆ Libraries
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\lib

◆ Examples
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF533\Services
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF537\Services
  ● C:\Program Files\Analog Devices\VisualDSP 4.0\Blackfin\EZ-KITs\ADSP-BF561\Services

◆ Documentation
  ● Device Driver and System Services User Manual
    ■ Blackfin Technical Library at www.analog.com
  ● Device Driver and System Services User Manual Addendum (Sept 2005)
Conclusion

- **System services provide:**
  - **Faster development**
    - Stable software base for application development
      - Fewer variables
    - Less re-invention
      - Don’t need to create everything from scratch
  - **Modular software**
    - Better compatibility
      - Resource control is managed by the system services
    - Easier integration
      - Multiple software components working concurrently
  - **Portability**
    - Code portable to other Blackfin processors
Additional Information

- **Documentation**
  - Device Drivers and System Services Manual for Blackfin Processors
  - Device Drivers and System Services Addendum (Sept 2005)

- For questions, click “Ask A Question” button