



www.kaztek.com

## WS\_CCESBF7

### System Development with CCES and the ADSP-BF70x Blackfin Processor Family

<b>Course Name:</b>	System Development with CrossCore Embedded Studio (CCES) and the ADSP-BF70x Blackfin Processor Family
<b>Course Code:</b>	WS_CCESBF7
<b>Course Description:</b>	<p>This is a practical and interactive course that is designed to systematically teach how to use the ADSP-BF70x Blackfin Processor Family to its fullest potential. Emphasis is placed on understanding the steps required to create an efficient Blackfin CPU based system in the way that ADI had intended the processor to be used. Several hands on exercises provide an opportunity for the instructor to work one on one with the attendee. Throughout the workshop, attendees are encouraged to ask questions.</p> <p>The CrossCore Embedded Studio (CCES) IDE is covered in detail, including topics on navigating through the IDE, projects and project configuration, the build process, and debug features. Tools based optimizations including compiler and linker optimization are covered.</p> <p>An understanding of the Blackfin architecture will enable getting the best performance out of the processor. Architecture topics covered include the new Blackfin+ core features, security and safety features, system fabric, and an overview of the peripherals. A section on booting covers what happens during the boot process, creating boot image files, and discussing how to get them into the target system. Hardware considerations such as operation of the power management and reset are also discussed. Hardware development tools, such as evaluation boards and ICE's are also covered, including setting up hardware debug sessions and other hardware debug topics. An introduction into Micrium's uC/OS is also covered in the workshop.</p> <p>Throughout the course, a number of hands on exercises will take the attendee through the various aspects of the software development process. Topics covered through exercises include setting up and building projects, C language programming, various optimizations, code debugging (simulation and hardware), and software module support (i.e. System Services and Device Drivers).</p>
<b>Goals and Objectives:</b>	The main course objective is to understand the Blackfin+ architecture (with a focus on the ADSP-BF70x Blackfin Processor Family members) sufficiently to enable DSP system designers to resolve hardware/software issues with their applications. Additional goals include gaining a thorough understanding of Blackfin code development using the CCES tool chain.
<b>Pre-requisites:</b>	Previous embedded microprocessor experience would be an asset (hardware and/or software).
<b>Target Audience:</b>	System Designers needing to make informed decisions on design tradeoffs, Hardware Designers needing to develop external interfaces and low level code, and Code Developers needing to know how to get the highest performance from their algorithms
<b>Duration:</b>	3.5 days



- 1 Introduction**
  - 1.1 Workshop outline**
  - 1.2 CCES Highlights**
  - 1.3 ADSP-BF70x Architecture Highlights**
    - 1.3.1 Blackfin+ Core**
    - 1.3.2 System Fabric Overview**
    - 1.3.3 Fault Management**
    - 1.3.4 System Security Features**
    - 1.3.5 Clock Domains**
  
- 2 Introduction to CCES**
  - 2.1 CCES IDE Overview**
    - 2.1.1 IDE Concepts**
    - 2.1.2 Navigating the IDE**
    - 2.1.3 Help**
  - 2.2 IDE and Projects**
    - 2.2.1 Creating projects and the Project Wizard**
    - 2.2.2 Project Options and Build Configurations**
    - 2.2.3 Project Structure**
    - 2.2.4 Invoking the Build Tools**
  - 2.3 Exercise – “Hello World”**
  - 2.4 Debug Sessions**
    - 2.4.1 Types of Debug Sessions**
    - 2.4.2 Creating/Launching Debug Sessions**
    - 2.4.3 Configuring a Debug Session**
  - 2.5 Debugger Features**
    - 2.5.1 Debug control**
    - 2.5.2 Debug windows**
  - 2.6 Exercise**
    - 2.6.1 Compiler Optimization topics**
    - 2.6.2 Performance assessment topics**



- 
- 3 Blackfin+ Core Review**
    - 3.1 Overview**
      - 3.1.1 Differences from original Blackfin Core**
    - 3.2 Highlights**
      - 3.2.1 Registers**
        - 3.2.1.1 Accessing registers in C**
      - 3.2.2 Computational Units**
      - 3.2.3 Addressing Unit**
      - 3.2.4 Sequencer**
        - 3.2.4.1 Program Flow Control**
        - 3.2.4.2 Core Interrupt Controller**
    - 3.3 Assembly Code Example**
      - 3.3.1 Source code structure**
      - 3.3.2 Issuing Parallel Instructions**
    - 3.4 Exercise**
  
  - 4 Memory Architecture**
    - 4.1 Features Introduction**
      - 4.1.1 Memory Hierarchy**
      - 4.1.2 Memory Maps**
      - 4.1.3 Bus Architecture**
    - 4.2 L1 Memory**
      - 4.2.1 Internal Memory Architecture**
      - 4.2.2 Instruction Memory and Control Registers**
      - 4.2.3 Data Memory and Control Registers**
      - 4.2.4 MMU (Memory Management Unit)**
    - 4.3 L2 Memory**
      - 4.3.1 Features**
    - 4.4 Memory Protection Overview**
      - 4.4.1 Parity/ECC for L1/L2 memory**
      - 4.4.2 CRC engines**
      - 4.4.3 SMPU (System memory Protection Unit)**



- 
- 5 Linker Operations**
    - 5.1 Converting C and Assembly files to Object files**
      - 5.1.1 Features and Overview**
      - 5.1.2 Assembler Expressions and Directives**
      - 5.1.3 Object Sections**
    - 5.2 The Link Process Linker Description File (LDF)**
      - 5.2.1 Overview**
      - 5.2.2 Linker Description File (LDF)**
        - 5.2.2.1 Example LDF**
      - 5.2.3 Linker Optimizations**
      - 5.2.4 Customizing and Auto Generation**
    - 5.3 L2 Utility ROM**
      - 5.3.1 Contents**
      - 5.3.2 Usage**
  
  - 6 Hardware Tools**
    - 6.1 EZKITs**
      - 6.1.1 Overview**
      - 6.1.2 EZKIT Extender Boards**
      - 6.1.3 Board Support Package**
        - 6.1.3.1 Application examples**
    - 6.2 In Circuit Emulators (ICE)**
      - 6.2.1 Configuration of a Debug Target**
    - 6.3 EZKIT/Emulator Debug Sessions**
      - 6.3.1 CCES Features for HW Debug**
    - 6.4 CPU Hardware Debug Features**
      - 6.4.1 System Debug and Trace Unit**
      - 6.4.2 System Watchpoint Unit**
  
  - 7 System Booting**
    - 7.1 Boot process**
    - 7.2 Boot Loader Stream**
      - 7.2.1 Format**
      - 7.2.2 Creating**
    - 7.3 Booting Methods**
      - 7.3.1 Boot Modes**
      - 7.3.2 Hardware Configuration**
    - 7.4 Command Line Device Programmer (CLDP) Utility**
      - 7.4.1 Demonstration – Boot stream create/flash**



## **8 System Services and Device Drivers**

### **8.1 Overview**

### **8.2 System Services API**

### **8.3 Device drivers**

#### **8.3.1 API**

#### **8.3.2 Buffers**

#### **8.3.3 Callback considerations**

### **8.4 Creating CCES projects that use SS/DD**

#### **8.4.1 Finding the files**

### **8.5 Walkthrough UART echo example**

#### **8.5.1 Demonstrate API and SSL considerations**

## **9 System Event Handling**

### **9.1 CEC (Core Event Controller) and SEC (System Event Controller)**

#### **9.1.1 Fault Management**

#### **9.1.2 Interrupt processing flow**

#### **9.1.3 Programming model**

### **9.2 Trigger Routing Unit (TRU)**

#### **9.2.1 Description**

#### **9.2.2 Comparison to traditional Event Handling**

## **10 PORTs and GPIO**

### **10.1 PORTs Overview**

#### **10.1.1 Multiplexing GPIO with peripheral functions**

#### **10.1.2 Pin Mux Control**

### **10.2 General Purpose I/O (GPIO)**

#### **10.2.1 Features**

#### **10.2.2 Configuration**

#### **10.2.3 Use in Event management**

### **10.3 Exercise**



---

## **11 Timers and Counters**

- 11.1 Overview**
- 11.2 Core Timer**
- 11.3 Real Time Clock**
- 11.4 Watch Dog Timer**
- 11.5 GP Timers**
  - 11.5.1 Pin Interrupt Capture Mode**
  - 11.5.2 Windowed Watchdog Mode**
  - 11.5.3 PWM Mode**
  - 11.5.4 Pulse Capture Mode**
  - 11.5.5 External Event Mode**
- 11.6 GP Counter**
  - 11.6.1 Features**
- 11.7 Exercise – Periodic Interrupts**

## **12 Direct Memory Access (DMA)**

- 12.1 Overview**
  - 12.1.1 Types of DMA**
- 12.2 Setting up DMAs**
  - 12.2.1 Register based DMA**
  - 12.2.2 Descriptor based DMA**
  - 12.2.3 System Services MemDMA Support**
- 12.3 DMA control**
  - 12.3.1 Interrupts**
  - 12.3.2 Trigger Support**
  - 12.3.3 Priority (via System Crossbars)**
  - 12.3.4 CRC (Cyclic Redundancy Check)**
  - 12.3.5 Bandwidth Limiting/Monitoring**
- 12.4 Exercise - MemDMA**



---

## **13 External Memory Interfaces**

- 13.1 Features and Overview**
  - 13.1.1 System Crossbar considerations**
- 13.2 Static Memory Controller**
  - 13.2.1 Interface Description**
  - 13.2.2 Programming Model**
- 13.3 Dynamic Memory Controller**
  - 13.3.1 Interface Description**
  - 13.3.2 Programming Model**
- 13.4 Performance**

## **14 Serial Communications**

- 14.1 Serial Port (SPORT)**
  - 14.1.1 Features**
  - 14.1.2 Pin Descriptions**
  - 14.1.3 Modes of Operation**
  - 14.1.4 Configuration**
- 14.2 Serial Peripheral Interface (SPI)**
  - 14.2.1 Features and Setup**
  - 14.2.2 SPI Host Port**
- 14.3 UART**
  - 14.3.1 Features and Setup**
- 14.4 Two Wire Interface (TWI)**
  - 14.4.1 Overview**
- 14.5 Controller Area Network (CAN)**
  - 14.5.1 Overview**

## **15 Enhanced PPI (EPPI)**

- 15.1 Overview**
- 15.2 Operating Modes**
- 15.3 Programming Model**

## **16 Security Features**

- 16.1 Overview**
- 16.2 System Protection Unit (SPU)**
- 16.3 System Memory Protection Unit (SMPU)**
- 16.4 Code Example**



---

**17 C Run Time Environment**

- 17.1 Compiler Overview**
- 17.2 Function Call and Return**
- 17.3 C/Assembly Language Interface**
- 17.4 Code example**

**18 Platform Design Topics**

- 18.1 Operating modes**
- 18.2 Dynamic Power Management**
  - 18.2.1 Clock generation / PLL**
  - 18.2.2 Power-down modes**
- 18.3 Housekeeping ADC (HADC)**
  - 18.3.1 Description**
  - 18.3.2 Programming**

**19 uC/OS-III**

- 19.1 Overview**
  - 19.1.1 VDK comparison**
- 19.2 Adding uC/OS support to a project**
- 19.3 RTOS Features**
  - 19.3.1 Threads and scheduling**
  - 19.3.2 Signaling and synchronization**
  - 19.3.3 Error handling**
  - 19.3.4 Debug assistance**
- 19.4 Demonstrate main API functions via example**