



www.kaztek.com

WS\_SHDEV

## System Development and Programming with the 3<sup>rd</sup> and 4<sup>th</sup> Generation SHARC Processors

Course Name:	System Development and Programming with the 3rd and 4th Generation SHARC Processors
Course Code:	WS_SHDEV
Course Description:	<p>This is a practical course with 'hands on' training using the latest VisualDSP++ software development tools. The course starts with getting a good working understanding of the VDSP++ development environment, including the development of a simple 'C' application. Next, the core elements of the SHARC, which includes the Computational Units, the Data Address Generators, and the Program Sequencer, are examined in detail along with a brief overview of the relevant assembly code instructions. The core section is common to all members of the SHARC family. A number of simulator labs help in understanding operation of the individual elements. Memory configuration (both internal and external) is discussed next. Advanced instructions and SIMD operation are presented with a follow on lab on code optimization. The I/O peripherals, which include the DAI and DPI, are discussed in detail along with DMA operation between these peripherals and internal memory. This section also deals with system booting. Hardware development tools, such as evaluation boards and ICE's are introduced with a follow on instructor led demonstration of an interrupt driven application for a hardware target. Throughout the course, the various aspects of the software development process using the latest tools are discussed including setting up and building projects, 'C' and assembly language programming, code debugging, simulation, and VDK.</p>
Goals and Objectives:	<p>The main course objective is to understand the SHARC architecture (with a focus on the ADSP-2136x/37x/4xx Sharc 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 SHARC code development (using both assembly and 'C') using the latest software tools.</p>
Pre-requisites:	<p>Previous embedded microprocessor experience (hardware and/or software) would be an asset.</p>
Target Audience:	<p>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</p>
Duration:	<p>3.5 days</p>



- 1 Introduction**
  - 1.1 Goal of Workshop**
  - 1.2 Course Overview**
  - 1.3 Course Logistics**
  - 1.4 Kaztek Systems Overview**
  - 1.5 Sharc Architecture Overview**
  
- 2 Introduction to Software Tools**
  - 2.1 VisualDSP++ Overview**
    - 2.1.1 Software Development Process**
  - 2.2 IDDE and Projects**
    - 2.2.1 Creating projects and the Project Wizard**
    - 2.2.2 Project Options and Build Configurations**
    - 2.2.3 ISA vs VISA Project Considerations**
  - 2.3 Invoking the Tools**
  - 2.4 Debug Sessions**
  - 2.5 Exercise – “Hello World”**
  - 2.6 Debugger Features**
    - 2.6.1 Debug control**
    - 2.6.2 Debug windows**
    - 2.6.3 Profiling**
  - 2.7 Help**
  - 2.8 Simulator Exercise – “FIR Filter”**
    - 2.8.1 Explore various debug features**
  
- 3 The ADSP-21K Family Core Architecture**
  - 3.1 The ADSP-21000 Family Core Internal Architecture**
    - 3.1.1 Registers and Data Types**
    - 3.1.2 Native data types and data word alignment**
      - 3.1.2.1 Fixed point - 32 bit integer, fractional**
      - 3.1.2.2 Floating point - 32 bit single precision, 40 bit extended single precision**
    - 3.1.3 Register types: UREG, SREG, DREG - overview**
    - 3.1.4 Register File**
    - 3.1.5 Optional Simulator Exercise: registers exercise, basic simulator operation**
  - 3.2 Sharc Assembly Language Syntax**



---

### **3.3 Compute Units**

#### **3.3.1 ALU**

##### **3.3.1.1 Features**

##### **3.3.1.2 Instructions**

##### **3.3.1.3 Flags**

##### **3.3.1.4 Optional Simulator Exercise: ALU operation**

#### **3.4 Multiplier/MAC**

##### **3.4.1 Features**

##### **3.4.2 Instructions**

##### **3.4.3 Flags**

##### **3.4.4 Fractional and integer math**

##### **3.4.5 Optional Simulator Exercise: MAC operation**

#### **3.5 Shifter**

##### **3.5.1 Features**

##### **3.5.2 Instructions**

##### **3.5.3 Flags**

##### **3.5.4 Optional Simulator Exercise: Shifter operation**

## **4 Memory**

### **4.1 SHARC Memory**

#### **4.1.1 SHARC Memory Basics**

#### **4.1.2 SHARC Memory Map**

#### **4.1.3 SHARC Internal Architecture**

### **4.2 SHARC Internal SRAM**

#### **4.2.1 Internal SRAM Architecture**

#### **4.2.2 Memory Overhead Considerations**

#### **4.2.3 Internal Memory Maps**

#### **4.2.4 Configuring Internal Memory**

#### **4.2.5 Example LDF Memory Section**

### **4.3 Fetching Data from Memory**

#### **4.3.1 Data Address Generators (DAG) Features**

#### **4.3.2 Data Move Examples**

#### **4.3.3 Address Pointer Manipulation**

#### **4.3.4 Stack Operations**

#### **4.3.5 DSP Addressing**

##### **4.3.5.1 Circular Buffering**

##### **4.3.5.2 Bit Reversal**



- 
- 5 Program Sequencer and Core Timer**
    - 5.1 Program Sequencer**
      - 5.1.1 Features**
        - 5.1.1.1 ISA vs VISA**
        - 5.1.1.2 Instructions**
        - 5.1.1.3 Instruction pipeline**
        - 5.1.1.4 Branching, Delayed branching**
        - 5.1.1.5 Zero overhead looping**
        - 5.1.1.6 Hardware Stacks**
      - 5.1.2 Interrupts**
        - 5.1.2.1 Programmable Interrupts**
        - 5.1.2.2 Interrupt Vector Table**
        - 5.1.2.3 Handling Interrupts in C**
      - 5.1.3 Instruction cache, PM data access**
    - 5.2 Core Timer**
      - 5.2.1 Features**
    - 5.3 System and Memory Mapped registers**
      - 5.3.1 Status and Mode registers / USTAT registers**
      - 5.3.2 System register bit operations**
      - 5.3.3 FLAG Bits**
        - 5.3.3.1 FLAGS register**
      - 5.3.4 SYSCTL register**
      - 5.3.5 Accessing System and Memory Mapped registers in C**
    - 5.4 Simulator Exercise – Core Timer and Interrupts**
  - 6 Linker Operations**
    - 6.1 Converting C and Assembly files to Object files**
      - 6.1.1 Features and Overview**
      - 6.1.2 Assembler Expressions and Directives**
      - 6.1.3 Object Sections**
      - 6.1.4 Def21xxx.h Files**
    - 6.2 Linker / Linker Description File (LDF)**
      - 6.2.1 Features and Overview**
      - 6.2.2 LDF Commands**
      - 6.2.3 Linker Operation**
      - 6.2.4 Example LDF**
      - 6.2.5 Linker Optimizations**
      - 6.2.6 Expert Linker**
    - 6.3 The LDF and the C Run Time Environment**



- 
- 7 Advanced Instruction Types and SIMD Operation**
    - 7.1 Parallel Instruction Types and Multifunction Computations**
      - 7.1.1 Conditional register swap example
      - 7.1.2 Data registers usage for multifunction computes
      - 7.1.3 Reciprocal and Divide
      - 7.1.4 Reciprocal Square Root and Square Root
    - 7.2 SIMD Single Instruction Multiple Data**
      - 7.2.1 Terminology and Features
      - 7.2.2 SISD vs SIMD
      - 7.2.3 Data Access
      - 7.2.4 SIMD Programming Models
      - 7.2.5 Status Flags
      - 7.2.6 Optional Simulator Exercise: code optimization
  
  - 8 Hardware Tools**
    - 8.1 Hardware Tool Overview with part numbers**
    - 8.2 Sharc EZ-Kit Lite**
      - 8.2.1 Overview with Part Numbers
      - 8.2.2 EZKIT Extender Boards
      - 8.2.3 EZKIT Debug Sessions
      - 8.2.4 Application examples
    - 8.3 JTAG In Circuit Emulators (ICE)**
      - 8.3.1 USB ICE overview
      - 8.3.2 Configuration of a Debug Target
      - 8.3.3 Emulator Debug Sessions
    - 8.4 Set up Sharc EZKIT with HPUSB JTAG ICE**
  
  - 9 IOP Introduction**
    - 9.1 Overview of IOP features including DMA, External Port, DAI/DPI**
  
  - 10 Direct Memory Access (DMA)**
    - 10.1 DMA Architecture
    - 10.2 DMA Features
    - 10.3 DMA Channel Prioritization
    - 10.4 DMA Transfer Types
      - 10.4.1 External Port DMA
      - 10.4.2 Peripheral DMA
      - 10.4.3 Memory to Memory DMA
    - 10.5 Transfer Control Blocks (TCB)
    - 10.6 DMA Control and Status registers
    - 10.7 DMA Chaining and Interrupts
    - 10.8 DMA programming examples
    - 10.9 Hardware Exercise - DMA



- 
- 11 External Port**
    - 11.1 EP Configuration**
    - 11.2 Asynchronous Memory Interface**
    - 11.3 SDRAM Controller**
      - 11.3.1 DDR Controller for ADSP-21469**
    - 11.4 Shared Memory Interface (ADSP-31368 only)**
  
  - 12 Digital Audio Interface (DAI)**
    - 12.1 DAI Overview**
    - 12.2 DAI Peripherals**
      - 12.2.1 SPORT**
      - 12.2.2 SPDI/F**
      - 12.2.3 Precision Clock Generator (PGC)**
      - 12.2.4 Sample Rate Converter (SRC)**
      - 12.2.5 Input Data Port (IDP)**
    - 12.3 Signal Routing Unit (SRU)**
      - 12.3.1 Terminology and Naming Convention**
      - 12.3.2 Configuring the SRU**
      - 12.3.3 Configuration examples**
    - 12.4 DAI Interrupts**
  
  - 13 Digital Peripheral Interface (DPI)**
    - 13.1 DPI Overview**
    - 13.2 DPI Peripherals**
      - 13.2.1 SPI**
      - 13.2.2 Two Wire Interface (TWI)**
      - 13.2.3 UART**
      - 13.2.4 General Purpose Timers**
    - 13.3 DPI Interrupts**
    - 13.4 Hardware Exercise – SRU Configuration**
  
  - 14 System Booting**
    - 14.1 Booting Methods**
      - 14.1.1 Boot time hardware configuration**
    - 14.2 VisualDSP Loader Utility**
      - 14.2.1 Creating boot image files**
      - 14.2.2 Loader file formats**
    - 14.3 Flash Programmer Utility**



- 
- 15 System Design Considerations**
    - 15.1 Clock generation**
    - 15.2 Power Management**
    - 15.3 Resetting the DSP**
    - 15.4 JTAG Overview**
    - 15.5 Board Design and Layout**
  
  - 16 Optimization Topics**
    - 16.1 Overview**
    - 16.2 General Approach to Optimization**
    - 16.3 Algorithmic considerations**
    - 16.4 Getting to know the Compiler**
      - 16.4.1 Optimization Switches and pragma's**
      - 16.4.2 Built-in functions**
      - 16.4.3 Example compiler output**
      - 16.4.4 Summary**
    - 16.5 Where to start optimizing?**
      - 16.5.1 Using the profiler effectively**
    - 16.6 C/Assembly Language Interfacing**
      - 16.6.1 Register Usage**
      - 16.6.2 Parameter Passing**
      - 16.6.3 Stack Usage**
      - 16.6.4 Example**
  
  - 17 VDK Introduction**
    - 17.1 Overview**
    - 17.2 VDK projects**
    - 17.3 Threads and scheduling**
      - 17.3.1 Signaling and synchronization**
      - 17.3.2 Memory pools and messaging**
    - 17.4 Multiple heaps**
    - 17.5 VDK Conventions**
    - 17.6 Error handling**
    - 17.7 Debug assistance**