StarterWare

ADAS Driver Team
8th Jan 2015

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

What is Starterware

- C based NO-OS code
- Can be used stand-alone
- Provides peripheral programming interface -Easy to use SW interface
- Provides software portability across devices for a given peripheral
- Provides stand-alone examples
- Not a middleware complete solution
- Product is designed to scale for each SoC. Single package is used to support more than one SoC. The product packaging/folder takes care of scalability for new IPs of new SoCs.
- Supports all four SoCs in single software package: TDA1Mx, TDA2xx, TDA2ex and TDA3xx.

Starterware Scope

IS

- C based NO-OS code
- Can be used stand-alone or with an RTOS
- Provides peripheral programming interface
  - Easy to use SW interface
- Provides software portability across devices for a given peripheral
- Tool-chain agnostic
- Provides stand-alone examples and example drivers with an RTOS
- Optimal software for peripherals in terms of performance, size

IS-NOT

- Depend upon any OS services (e.g. thread, locks, memory mgmt)
- Full featured software for peripherals (async, queuing)
- Peripheral “IP version specific” software
- Code with OS dependency for examples, demos
- Middleware complete solution

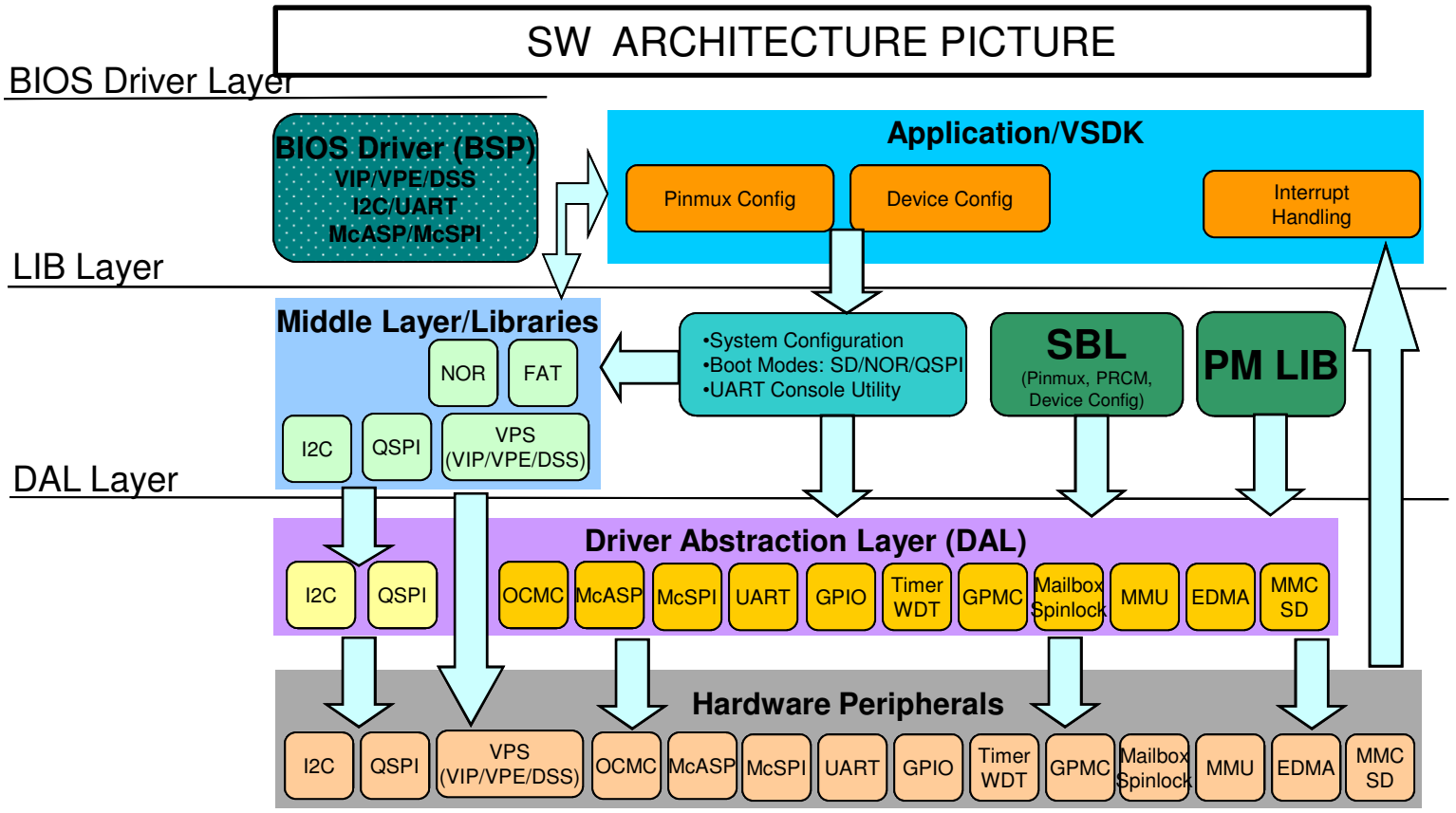
Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

Enabling customers to differentiate

- Starter-ware to start development on TI devices
- Enable customer to understand our IPs along with the TRM

HAL Libraries:
UART, I2C GPIO, Mailbox, Spinlock, EDMA, GPMC, McASP, McSPI, OCMC, QSPI, SBL, MMU, Timer, MMCSD, WDTIMER, PCIe
Driver libraries: I2C, QSPI, FAT, NOR, VPS (VIP/DSS/VPE)
SD, NOR and QSPI bootmodes
UART console utility



Software Partition(1/6)

- Driver Abstraction Layer
- Libraries
- System Configuration Code
- Utilities
- Platform Code
- Tools
- Boards
- Devices
- Secondary Boot loader (SBL)
- Peripheral Examples

Software Partition(2/6)

- Driver Abstraction Layer
  - APIs to configure and access peripherals
  - Simple, consistent and intuitive to use APIs
  - Logical group of register reads and writes to get a functional layer for a given IP
  - Stateless functions; blocking and runs to completion

- Libraries
  - Provide higher level API for certain complex peripherals
  - Uses driver abstraction layer
  - Provide buffer management and manage state of the device

Software Partition(3/6)

- System Configuration Code
  - Provides functions specific to SoC level System Configuration.
  - Provides the following functionalities:
    - Start-up Code
    - Interrupt-vector initialization
    - Low level CPU specific code
  - May involve assembly code and hence can be tools specific

- Utilities
  - Quick start utilities for external devices and interfaces on the board.
  - Build on underlying DAL or driver library
  - E.g. UartConsole contains wrapper functions which use UART APIs to interface with the user through the serial console.

Software Partition(4/6)

- Platform Code
  - Provides functions specific to an EVM to enable peripheral operations.
  - Normally provides the following functionalities:
    - Peripheral pin-muxing
    - Peripheral clock setting
    - EVM profile setting
    - I/O Expander settings

- Tools
  - Contains various SW tools like flash tools
  - E.g. Nor flash Writer is used to write data or erase the NOR flash

Software Partition(5/6)

- Boards
  - Contains list of on-board devices and daughter cards like Vision App Board, Custom Board, etc
  - Provides routines to auto-detect the board/daughter card type
  - Provides special programming like I2C configuration for selecting on-board device.

- Devices
  - Contains APIs to control IO expanders
  - Used to read/write into video on-board devices such as Sii9022a, TVP7002, etc.

Software Partition(6/6)

- Secondary Boot loader (SBL)
  - Secondary boot loader does the following:
    - Initialize the SoC
    - Configure the peripheral I/O
    - Initialize DDR
    - Load multicore RPRC image and brings the slave cores out of reset
  - Boot Modes: QSPI, QSPI_SD, NOR and MMCSD
  - Various boot modes are supported depending upon SoC.

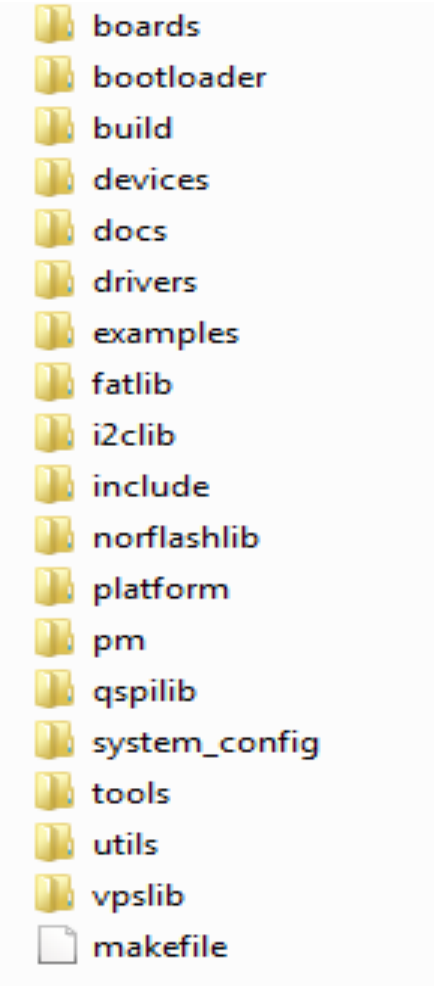
- Peripheral Examples
  - Sample applications for different peripherals
  - Demonstrate some of the capabilities of IP
  - Uses driver abstraction layer or library

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

Directory Structure

- Uses simple and intuitive directory structure
- Provides scope to accommodate new SoCs in the same directory structure
- Build folder contains all the make rules for building starterware
- Separate directories for different libraries like i2clib, vpslib, etc.
- Include folder contains all the DAL header files
  - Contains register layer files which contain peripheral register offset macros and register field token fields



Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

Starterware Build

- Dependencies:
  - TI Compiler TMS470
  - TI Compiler C6000
  - TI Compiler ARP32
  - Linaro GCC tool chain for A15

- Build:
  - Makefile based build
  - Makerules for different cores are specified in build/makerules
  - Environment variables are specified in build/makerules/env.mk
  - *gmake all PLATFORM=<PLATFORM> builds all the components*
  - *gmake libs PLATFORM=<PLATFORM> builds libs*

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

MPU Subsystem

- MPU Subsystem contains
  - Two Cortex A15 cores : MPU_C0 and MPU_C1
  - L1 and L2 cache (32 KB and 2 MB respectively)
  - Interrupt Controller
- Supports continuous fetch and decoding of three instructions per clock cycle.
- Can issue two simple instructions in a cycle.
- Can issue a load and store instruction in a cycle.
- Symmetric Multi Processing (SMP) Architecture
- Starterware does not support Cache.

Generic Interrupt Controller

- Generic Interrupt Controller (GIC), also referred to as MPU_INTC.
- Supports 160 shared peripheral interrupts.
- Supports 16 software generated interrupts.
- Individual priority of each interrupt.
- Non-secure interrupt generates an IRQ interrupt request to target processor.
- Secure interrupt can signal either IRQ or FIQ interrupt request to target processor.

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

IPU Subsystem

- IPU Subsystem contains
  - Two Cortex M4 cores : IPUx_C0 and IPUx_C1
  - Common L1 cache (called Unicache): 32 KB
  - Integrated Nested Vector Interrupt Controller
- Internal MMU(called AMMU)
  - 16-entry region-based address translation
  - Read/write control and access type control
  - Execute never(XN) MMU protection policy
- L2 MMU: 32 entries with table walking logic
- On-chip ROM (IPUx_ROM) and banked RAM (IPUx_RAM) memory.

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

Interrupt Controller- StarterWare

- The Interrupt System exports a set of APIs to enable/disable the core interrupts and to configure and use the INTC.
- The API functions are exported in \starterware_xx_xx_xx_xx\include\<arch>\interrupt.h
- It might be required to configure IRQ Crossbar before using the system interrupt in order to connect the interrupt source with the IRQ line.

Programming

- Following sequence can be used to setup INTC for system interrupt:

1. In privilege mode of core restore the processor IRQ only status by calling Intc_IntEnable() API.

Interrupt Controller- StarterWare

2. INTC initialization shall be done before any interrupt processing is enabled, by calling Intc_Init() API.

3. Register the interrupt handler for the system interrupt using Intc_IntRegister().

4. The priority is set through Intc_IntPrioritySet() API.

5. Enable the system interrupt using the API Intc_SystemEnable().

- After the configuration and setting up of INTC, the application shall enable the interrupt processing at the peripheral.

Agenda

- Starterware Overview
- Starterware Software Architecture
- Directory Structure
- Starterware Build
- MPU Subsystem
- IPU Subsystem
- Interrupt Controller
- Running Examples
- Q&A

Tools Required

- Code Composer Studio (v 5.4.0.00091 or above)
- Chip Support Package (CSP)
- Development Board
  - TDA2xx CPU Board (Vision High EVM)
  - TDA3xx CPU Board (Vision Low EVM)
- Emulator
  - XDS560v2
  - XDS200
- Flash tools:
  - NOR Flash Writer
  - QSPI Flash Writer
- SD Card

Running Examples

- Depending on the boot mode, set the SYSBOOT switch as per the SBL user guide.
- For SD boot, copy the MLO file from bootloader\prebuild_binaries\tda2xx_images\sd to sd card.
- For QSPI, QSPI_SD and NOR boot, write the SBL image from bootloader\prebuild_binaries\<tda2xx/tda3xx>_images\<qspi/nor/qspi_sd> into flash using flash writer.
- Power on the CPU Board.
- Use CCS to connect to EVM using the target configuration.
- Load and run the binaries on the A15/M4 (or any other appropriate) core.
- Prints can be observed on CCS console or Tera Term can be used for viewing UART prints.

Questions ???