Secondary Boot Loader

ADAS Driver Team
8th Jan 2015

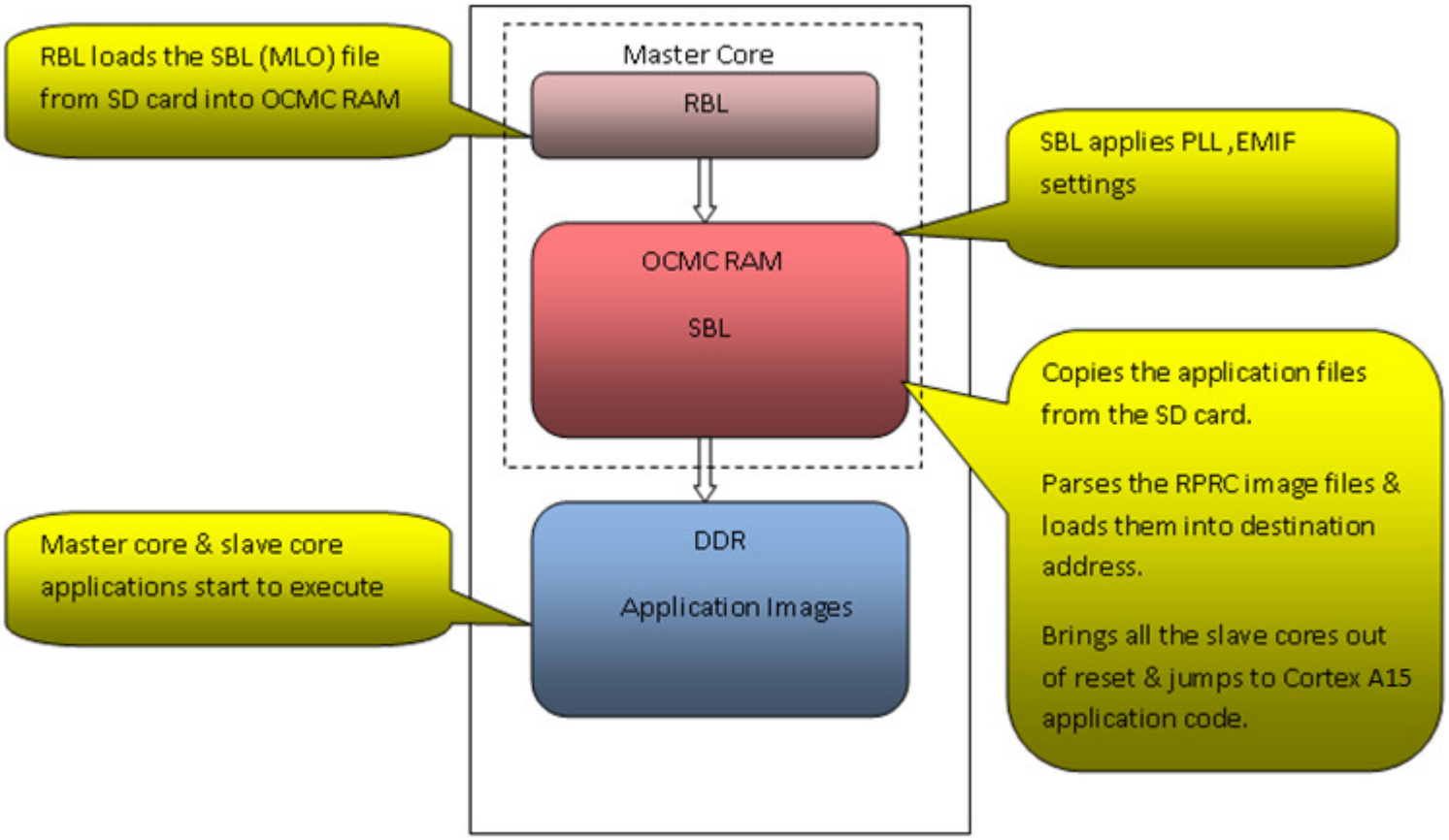
Agenda

- SBL Overview
- Directory Structure
- TDA2x SBL
- TDA3x SBL
- SBL Build
- Q&A

SBL Overview

- SBL initializes the execution environment for multi-core application
- SBL Functionality:
  - Configure the Voltage rails (AVS Class 0)
  - Initialize the Interrupt Controller
  - Setup the ADPLL values
  - Power on I/O peripherals
  - Power on slave cores
  - Configure required PADs
  - Initialize DDR
  - Load the multicore application image into DDR
  - Bring slave cores out of Reset
- SBL supports multiple boot modes depending on the SoC

TDA2x SBL: Boot Sequence



SBL Flow

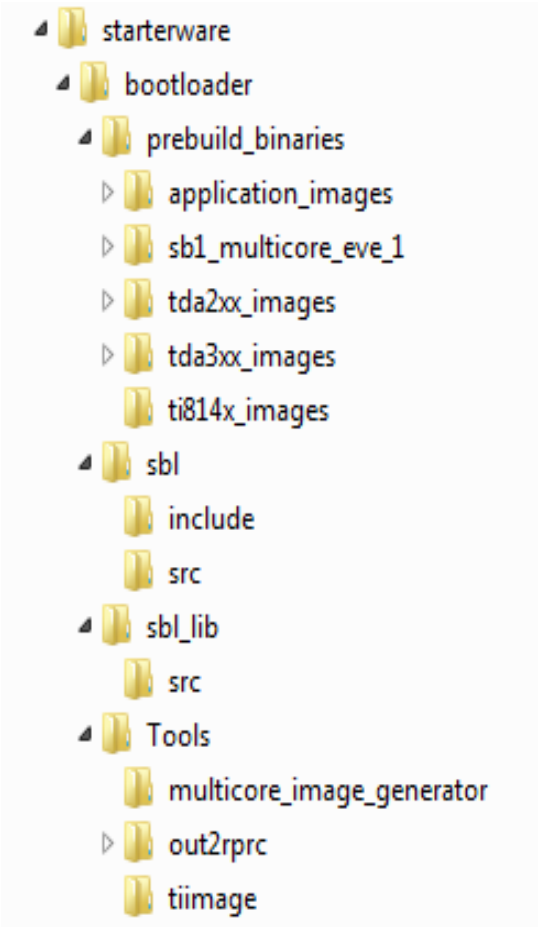
- During boot, SBL executes on the Master core. RBL loads the SBL image file into internal memory from the boot media depending on SYSBOOT settings.
- SBL first programs the PMIC to configure the Voltage Rails.
- Then SBL configures the different DPLL and clock domains.
- Then SBL enables various IO peripherals.
- Then SBL does the pad mux configuration and configures the DDR.
- Then SBL powers on the slave cores.
- After SoC configuration, the SBL loads the multicore Application image from the boot media into DDR.
- Then SBL brings the slave cores out of reset & jumps to the respective entry points.
- SBL finally jumps to the Master Core's binary.

Agenda

- SBL Overview
- Directory Structure
- TDA2x SBL
- TDA3x SBL
- SBL Build
- Q&A

Directory Structure

- Uses simple and intuitive directory structure
- Prebuild binaries folder contains the SBL images and App Images for various SoCs.
- SBL Lib folder contains the library APIs which are used by the SBL application.
- SBL folder contains the source code for SBL application.
- Tools folder contains following SBL tools:
  - Multicore image generator for creating multicore image from individual RPRC images
  - Outrprc tool to convert from ELF format to rprc format
  - Tiimage tool to convert SBL binary ELF format to tiimage format



Agenda

- SBL Overview
- Directory Structure
- TDA2x SBL
- TDA3x SBL
- SBL Build
- Q&A

TDA2x SBL Features

- SBL Build Mode:
  - SBL supports two build modes: Dev (Development) and Prod (Production)
  - In production build mode, if a valid App Image for a given CPU is not found, then the SBL puts corresponding core in power down mode.
  - In development build mode, CPU is not put to power down mode when valid App image is not there.
  - Can be specified at build time using SBL_BUILD_MODE=< build_mode> where build mode can be ‘prod’ or ‘dev’
- EMIF Mode:
  - SBL supports three different LISA configurations for tda2xx device:
    - DUAL_EMIF_2X512MB
    - DUAL_EMIF_1GB_512MB
    - SINGLE_EMIF_256MB
  - Different modes can be set with a build option EMIFMODE=<option>

Agenda

- SBL Overview
- Directory Structure
- TDA2x SBL
- TDA3x SBL
- SBL Build
- Q&A

TDA3x SBL Features

- Enabling functional safety at the boot time is critical for ADAS systems to meet the ASIL-A/B compliance standards.
- TDA3x SBL provides fast boot with functional safety.
- It does a self test (Logic/Memory) of various sub-systems of the SoC for integrity.
- It verifies integrity of application image by performing CRC.
- It enables ECC on different memories in the SoC (internal and external).
- It provides first CAN response to the main ECU in < 100 ms (including minimum safety tests).
- It boot the main application (including full SoC safety tests) with Video on Display in < 500 ms.
- It boots the rest of the application (including analytics application on DSP/EVE) in < 1s.

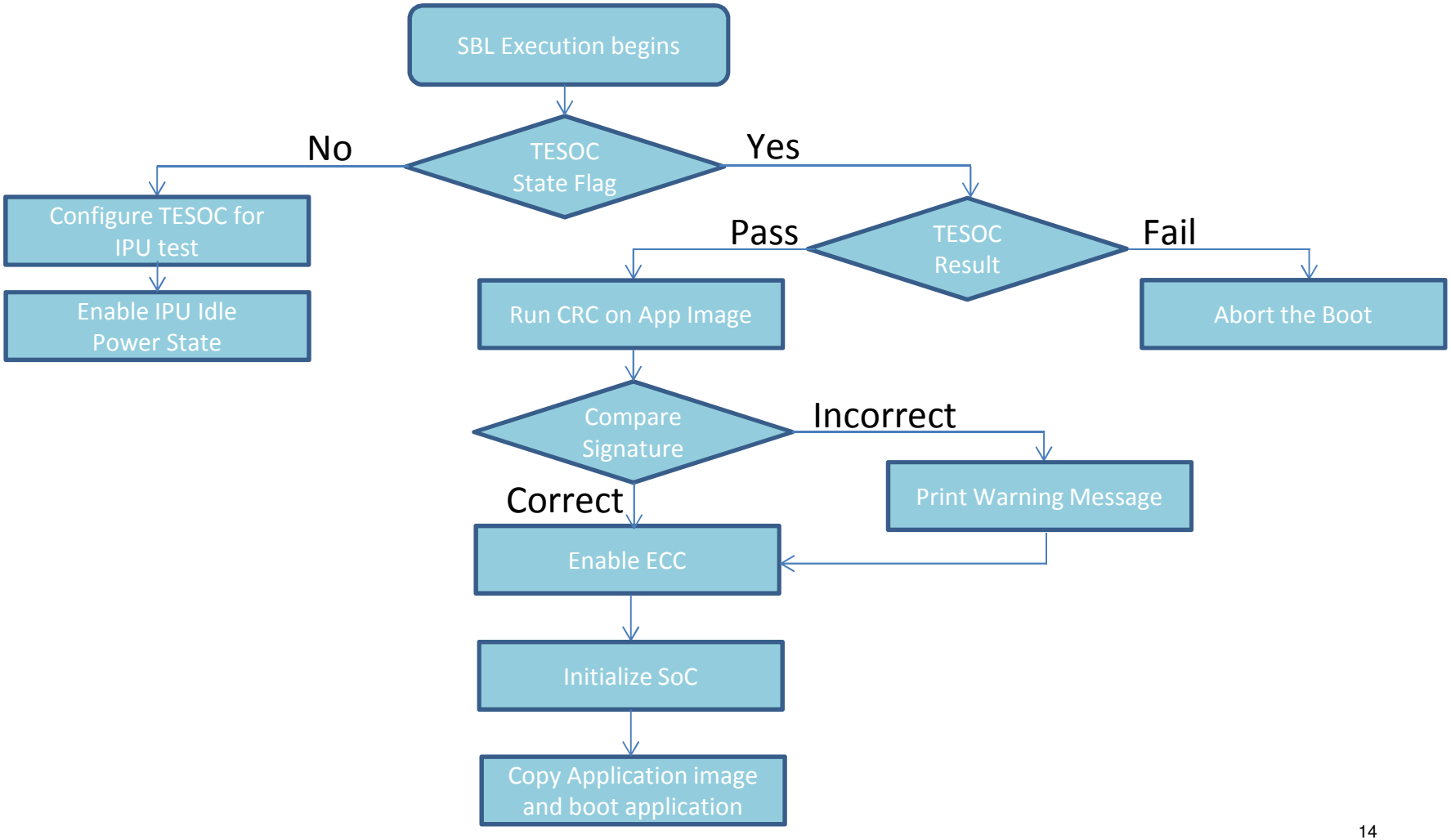
Safety IPs(1/2)

- MCRC
  - Memory Cyclic Redundancy Check (MCRC) performs checks to verify the integrity of a segment of memory. MCRC reads memory content and generates signature representing the segment of memory.
  - In semi-CPU mode, it uses EDMA for data transfer and generates completion interrupt to CPU for comparison against pre-determined good signature value.
- ECC
  - Error Correction Code (ECC), when enabled on a memory, detects 2-bit errors and correct 1-bit errors, thus making it immune to single bit errors
  - For enabling ECC, memory needs to be pre-filled with a known data pattern else ECC will erroneously report a ECC fault.
  - SBL enables ECC only for the memory region which is required by M4 Application responsible for DCAN response and Video Capture-Display application.

Safety IPs(2/2)

- TESOC
  - Tester On Chip (TESOC) supports Logical “Build-In Self-Test” (LBIST) on various CPUs and memory test (PBIST) of critical memories.
  - TESOC generates reset on the domain on which it is run. Ex, when TESOC is run on IPU subsystem it will reset the M4 sub-system, therefore RBL and SBL will be run twice.
  - Also, TESOC can’t be run on multiple domains like DSP/EVE/IPU in one go.
  - Therefore, instead of doing LBIST on all domains like IPU, DSP, EVE, etc. SBL does it only on IPU domain & let Application later perform LBIST/PBIST on remaining domains before use.
  - RBL comprehends the TESOC existence so that it does not copy SBL again
  - SBL does minimum configuration before doing LBIST on IPU and comprehends TESOC’s existence so that same configuration is not done again after TESOC reset.

Safety through SBL



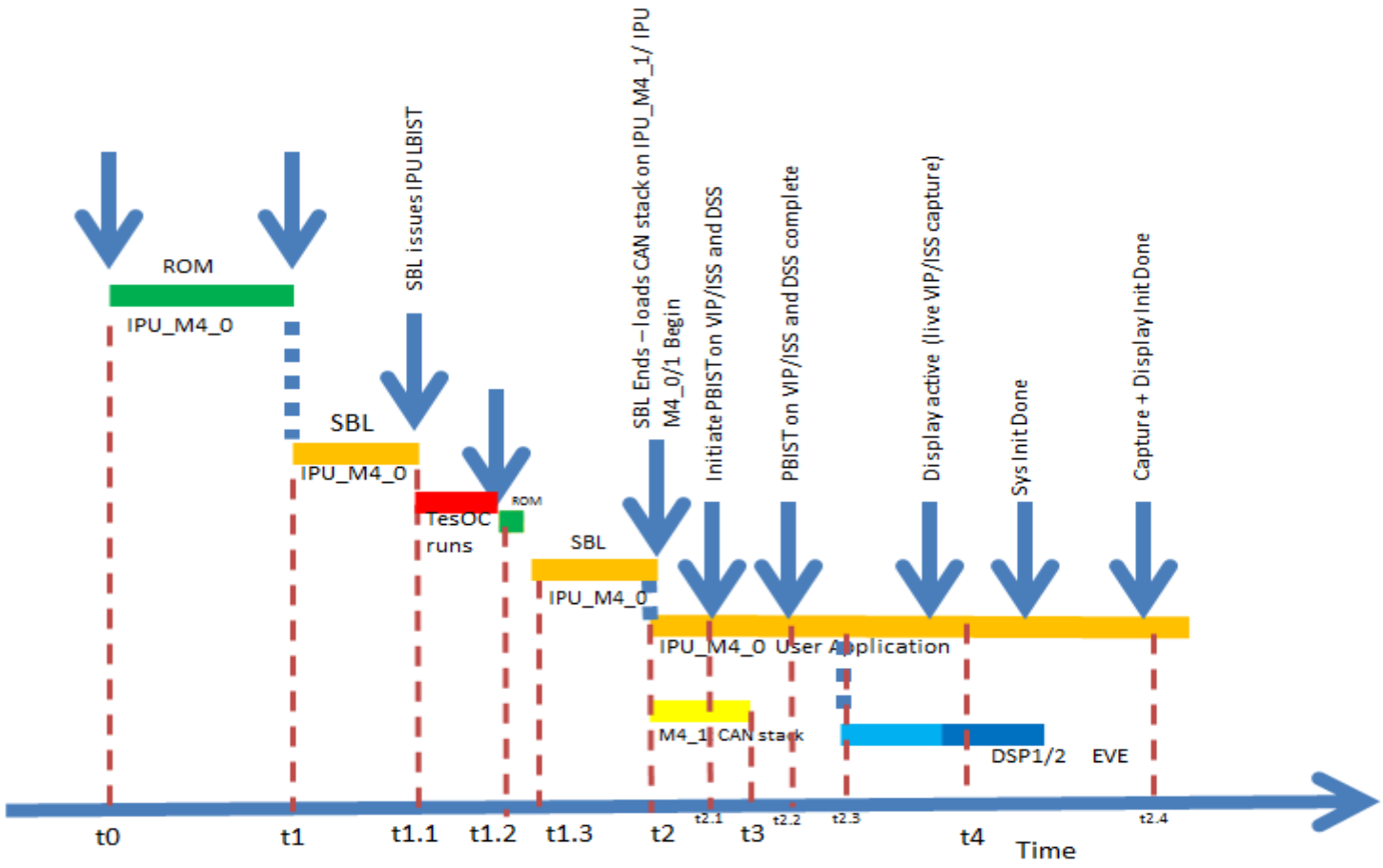
Fast Boot Strategy

- Statically Configurable SBL:
  - Static configuration feature is added so that SBL can be modified to enable only the required modules and clock domains. Hence RBL takes less time to copy the SBL code from Flash Memory to OCMC RAM.

- Boot Strategy:
  - SBL loads only the single M4 application on IPU Core-1 and thus is ready for quick CAN response.
  - Meanwhile, SBL loads the other M4 (IPU Core-0) for Video Capture-Display.
  - The application on IPU Core-0 first starts Capture-Display and then it boots DSP and EVE.

Fast Boot through SBL

- SBL Fast Boot timing sequence



Agenda

- SBL Overview
- Directory Structure
- TDA2x SBL
- TDA3x SBL
- SBL Build
- Q&A

Building SBL

- For specific boot mode SBL can be built using command:

  `gmake -s sbl PLATFORM=<platform> BOOTMODE=<boot_mode>`

  - PLATFORM: tda2xx, tda2ex or tda3xx
  - BOOTMODE: qspi, qspi_sd, sd or nor depending on PLATFORM

- Command to build SBL for all boot modes:

  `gmake -s sbl_all PLATFORM=<platform>`

- SBL can initialize the SoC for different operating points: OPP Low, OPP Nom, OPP OD and OPP High. In order to build SBL for a particular OPP build using the below command:

  `gmake -s sbl_all PLATFORM=<platform> OPPMODE=<opp_mode>`

  - OPPMODE: opp_low, opp_nom, opp_od or opp_high

- Command to build SBL for all boot modes for all OPPs:

  `gmake -s sbl_all_opps PLATFORM=<platform>`

Questions ???