


Upstream Zephyr Support on BeagleBone AI-64 R5 - Suraj Sonawane


BeagleBoard.org
May 16, 2024

Table of contents

Chapter 1

Introduction

The goal of the project is to upsteam Zephyr RTOS support to run on Cortex R5 processor cores loaded from A72 core running Linux through remoteproc. Some work has already been done by a previous year Gsoc contributor (VIM , GPIO), but not all the PR's have been merged. Therefore, I will be adding peripheral support for those PR's that were not merged (J721E, Board support, DIM Timer support) and also adding peripheral support for ADC, I2C, SPI, TI-SCI on TDA4VM SoC. The Cortex R5 processor cores are designed to provide deeply embedded real-time and safety-critical systems, so adding Zephyr RTOS support for R5 core in TDA4VM will be very helpful for users.

1.1 Summary links

- **Contributor:** Suraj Sonawane
- **Mentors:** Nishanth Menon, Dhruva Gole, Andrew Davis.
- **Code:** TBD
- **Documentation:** TBD
- **GSoC:** TBD

1.2 Status

This project is currently just a proposal.

1.3 Proposal

- Created accounts across OpenBeagle , Discord and Beagle Forum.
- The PR for Cross Compilation task: #192.

1.4 About

- **Forum:** u/suraj_sonawane (Suraj Sonawane)
- **OpenBeagle:** SurajS0215 (Suraj Sonawane)
- **Github:** SurajSonawane2415
- **School:** Veermata Jijabai Technological Institute, Mumbai

- **Country:** India
- **Primary language:** English, Hindi, Marathi
- **Typical work hours:** 10AM-7PM Indian Standard Time
- **Previous GSoC participation:** This is my first time participating in GSoC.

Chapter 2

Project

Project name: Upstream Zephyr Support on BeagleBone AI-64 R5.

2.1 Description

2.2 Overview

Zephyr is a small, yet scalable, full-featured OS with an architecture that allows developers to focus on applications requiring a real-time OS.

BeagleBone AI-64, is built on a proven open source Linux approach, bringing a massive amount of computing power to the hands of developers in an easy to use single board computer. Leveraging the Texas Instruments TDA4VM SOC with dual 64-bit Arm® Cortex®-A72, C7x DSP along with deep learning, vision and multimedia accelerators, developers have access to faster analytics, more data storage options, more high speed interfaces including all the connectors you'll need on board to build applications such as Autonomous Robotics, Vision and Video Analytics, Hi-End Drones, Media Servers, and Smart Buildings.

2.2.1 TI-SCI

- Texas Instruments' System Control Interface (TISCI) defines the communication protocol between various processing entities to the System Control Entity on TI SoCs. This is a set of message formats and sequence of operations required to communicate and get system services processed from the System Control Entity in the SoC.
- TISCI protocol is used to talk to the system software.

Power and Clock Management Public APIs are provided to perform:

- 'Device Management <https://software-dl.ti.com/tisci/esd/latest/2_tisci_msgs/pm/devices.html>'_: Enable and release a module This configures both power and clock details for the module and keeps track of its usage.
- 'Clock Management <https://software-dl.ti.com/tisci/esd/latest/2_tisci_msgs/pm/clocks.html>'_: Control the frequency of the clock to a module.

Drivers needed to implement TI-SCI:

- Mailbox controller driver
- **SCIClient Driver:**
  The SCIClient is an interface to the TI-SCI protocol for RTOS and non-OS based applications. It exposes the core message details, valid module/clock IDs to the higher level software and abstracts

the communication with the firmware based on the TI-SCI protocol. These APIs can be called by power, resource and security RTOS drivers or any other non-OS or RTOS based higher level software to be able to communicate with DMSC for its services. The higher level software is expected to populate the necessary message core parameters. The SCIClient would wrap the core message with the necessary protocol header and forward this to the DMSC. The SCIClient relies on the CSL-FL layer to program and interact with the Secure Proxy Threads. The SCIClient's goal is to ensure there is no duplication of the interface to the DMSC from different software components which need to interact with the DMSC or other System Control Entities in future devices.

Device Management and Security Control (DMSC)

The Device Management and Security Control (DMSC) subsystem is a processing entity on Jacinto 7 family of devices which performs centralized boot, power, security, and resource management functions, and System Firmware is the firmware which executes on the subsystem. System Firmware implements the Texas Instruments System Control Interface (TISCI) which client software can use to request various services related to device power, resources, and security functions.

J721E Device ID:

- The device IDs represent SoC subsystems that can be modified via DMSC TISCI message APIs.
- Some Secure, Power, and Resource Management DMSC subsystem TISCI message APIs define a device ID as a parameter allowing a user to specify management of a particular SoC subsystem.

J721E Clock ID:

- TISCI message Power Management APIs define a device ID and clock ID as parameters allowing a user to specify granular control of clocks for a particular SoC subsystem.

2.2.2 DM Timer Controller

In TDA4VM, there are thirty timer modules in the device. MCU_TIMER0 to MCU_TIMER9 are in MCU domain and TIMER0 to TIMER19 in MAIN domain. All timers include specific functions to generate accurate tick interrupts to the operating system.

Timers Features

- Interrupts generated on overflow, compare, and capture.
- Programmable divider clock source (2n, where n = [0-8])
- Dedicated input trigger for capture mode and dedicated output trigger/PWM signal
- Generates a 1-ms tick clock when functional clock is 32.768 kHz

For each timer implemented in the device, there are two possible clock sources: • 32-kHz clock • System clock

Each timer supports three functional modes: • Timer mode • Capture mode • Compare mode The capture and compare modes are disabled by default after core reset.

2.2.3 ADC

Analog-to-Digital Converter (MCU_ADC) module contains a single 12-bit ADC which can be multiplexed to any 1 of 8 analog inputs (channels). Integrated in the MCU domain are two instances, each with the following main features:

- 4 MSPS rate with a 60 MHz sample clock
- Single-ended or differential input options

- Each ADC module can be configured and transformed into digital test inputs
- Programmable 16 steps Finite State Machine (FSM) sequencer

2.2.4 I2C

Device MAIN domain contains seven multi-master Inter-Integrated Circuit (I2C) interfaces, each with the following main features:

- Compliancy to the Philips I2C-bus specification version 2.1
- Support of standard mode (up to 100 Kbps) and fast mode (up to 400 Kbps)
- Support of 7-bit and 10-bit device addressing modes
- Support of multi-master transmitter/slave receiver and receiver/slave transmitter modes
- Built-in FIFOs with programmable size of 8 to 64 bytes for buffered read or write
- 8-bit-wide data access
- Support of Auto Idle, Idle Request/Idle Acknowledge handshake, and Asynchronous Wakeup mechanisms
- Low power consumption

2.2.5 SPI

Integrated in MAIN domain eight Multi-channel Serial Peripheral Interface (MCSPI) modules have the following main features:

- Serial clock with programmable frequency, polarity, and phase for each channel
- Wide selection of MCSPI word lengths, ranging from 4 to 32 bits
- Up to four master channels, or single channel in slave mode
- Support of different master multichannel modes
- Single interrupt line for multiple interrupt source events
- Support of start-bit write command
- Support of start-bit pause and break sequence

2.3 Implementation Overview

2.3.1 Initial configuration:

1. Write Kconfig and device tree filels (Already done by previous year gsoc contributor) - Write Kconfig and device tree filels: Already done by previous year gsoc contributor in this PR.

2. Add Flash and debug support: Zephyr supports Building, Flashing and Debugging via west extension commands. To add west flash and west debug support, we need to create a board.cmake file in board directory. This file's job is to configure a "runner" for your board. (There's nothing special you need to do to get west build support for board.) West is "pluggable": you can add your own commands to west without editing its source code. These are called west extension commands, or just "extensions" for short.

- Adding a West Extension:

1. Write the code implementing the command.
2. Add information about it to a west-commands.yml file.
3. Make sure the west-commands.yml file is referenced in the west manifest.

2.3.2 Driver Development:

1. Devicetree bindings: A devicetree binding declares requirements on the contents of nodes, and provides semantic information about the contents of valid nodes. Zephyr devicetree bindings are YAML files in a custom format (Zephyr does not use the dt-schema tools used by the Linux kernel).

2. API Definitions: Available under "include/drivers/". Provides a general way of unteracting with driver instances.

2.3.3 Add basic support for Device Manager using TISCI:

- Device Manager enable and release a module.
- This configures both **power and clock** details for the module and keeps track of its usage.
- Add support driver to allow communication with system controller entity within the SoC using the mailbox client.
- **There are some existing APIs by using this API's we can add basic support for Device Manager using TISCI:**
  - **Device configuration and control APIs**

TISCI Message ID	Message Name
0x0200	TISCI_MSG_SET_DEVICE
0x0201	TISCI_MSG_GET_DEVICE
0x0202	TISCI_MSG_SET_DEVICE_RESETS

1. TISCI_MSG_SET_DEVICE : Request for a device state to be set. This is used to set or release various resets of the hardware block. This is typically used for hardware blocks which require special handling prior to specific resets being released. Typical example is when starting up a processing entity like ICSS/DSP, the device must be requested with resets asserted, required firmware loaded and the required resets released in appropriate order for operation of the device.
2. TISCI_MSG_GET_DEVICE : Retrieve the hardware block state. This requests information regarding the state of a device including the device's programmed state, current state, number of resets, and the number of times a device lost context.
3. TISCI_MSG_SET_DEVICE_RESETS : Set the state of device reset state. This is used to set or release various resets of the hardware block. This is typically used for hardware blocks which require special handling prior to specific resets being released. Typical example is when starting up a processing entity like ICSS/DSP, the device must be requested with resets asserted, required firmware loaded and the required resets released in appropriate order for operation of the device.

2.4 Software

C, RTOS

2.5 Hardware

BeagleBone AI-64

Chapter 3

Timeline

3.1 Timeline summary

Date	Activity
February 26 - March 28	Connect with possible mentors, review past year's work(PR's), overview the Zephyr codebase, read documentation on adding board support, and create a proposal.
March 28 - April 2	Proposal review and Submission.
April 3 - April 30	Start working on studying the unmerged PR's . Getting familiar with Hardware and looking deeper in Zephyr code base. Also looking into TI-SCI TI-SCI implementation idea.
May 1 - May 10	Start bonding
May 11 - May 31	Focus on College Exams
June 1 - June 2	Start coding and introductory video
June 3	Release introductory video and complete milestone #1
June 10	Complete milestone #2
June 17	Complete milestone #3
June 24	Complete milestone #4
July 1	Complete milestone #5
July 8	Submit midterm evaluations
July 15	Complete milestone #6
July 22	Complete milestone #7
July 29	Complete milestone #8
August 5	Complete milestone #9
August 12	Complete milestone #10
August 19	Submit final project video, submit final work to GSoC site and complete final mentor evaluation

3.2 Timeline detailed

3.2.1 Community Bonding Period (May 1st - May 15th)

- Add west flash support.
- Work on the unmerged pull request from a previous year's Google Summer of Code contributor titled "Add TI J721e R5 and BeagleBone AI64 R5 initial support #59191" in order to merge it.
- Work on the unmerged pull request from a previous year's Google Summer of Code contributor titled "Add TI J721e DM TIMER support #61020" in order to merge it.

3.2.2 Coding begins (June 1st)

- Start making an introductory Video.

- Finalise TI-SCI implementation idea by discussion with mentors.

3.2.3 Milestone #1, Introductory YouTube video (June 3rd)

- Introductory Video.
- Start adding basic support for TI-SCI protocol.

3.2.4 Milestone #2 (June 10th)

- Complete basic support for TI-SCI protocol.
- Start adding basic support for Device Manager using TI-SCI.

3.2.5 Milestone #3 (June 17th)

- **Adding basic support for Device Manager using TI-SCI.**
  - Implement Mailbox controller driver to provide support for Device Manager using TI-SCI.

3.2.6 Milestone #4 (June 24th)

- **Adding basic support for Device Manager using TI-SCI.**
  - Implement SCIClient Driver: to provide support for Device Manager using TI-SCI.

3.2.7 Milestone #5 (July 1st)

- Complete basic support for Device Manager using TI-SCI.
- Run basic IPC examples with the Linux cores, to control clock via device manager.

3.2.8 Submit midterm evaluations (July 8th)

Important: **July 12 - 18:00 UTC:** Midterm evaluation deadline (standard coding period)

- Submit work and documentation for the features developed.

3.2.9 Milestone #6 (July 15th)

- Understanding I2C module IN TDA4VM.
- Add I2C Support in Zephyr.

3.2.10 Milestone #7 (July 22nd)

- Understanding SPI module IN TDA4VM.
- Add SPI Support in Zephyr.

3.2.11 Milestone #8 (July 29th)

- Understanding ADC module IN TDA4VM.
- Add ADC Support in Zephyr.

3.2.12 Milestone #9 (Aug 5th)

- Add basic support for Clock control using TI-SCI protocol.

3.2.13 Milestone #10 (Aug 12th)

- Complete basic support for Clock control using TI-SCI protocol.

3.2.14 Final YouTube video (Aug 19th)

- Submit final project video, submit final work to GSoC site and complete final mentor evaluation

3.2.15 Final Submission (Aug 24nd)

Important: August 19 - 26 - 18:00 UTC: Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

August 26 - September 2 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

3.2.16 Initial results (September 3)

Important: September 3 - November 4: GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with extended deadline

---

Chapter 4

Experience and approch

- Worked on Multi Firmware ESP project, developing bootloader for ESP32. As a result, I have a good understanding of low-level codes in the C language.
- Implemented Dijkstra algorithm in C for Maze-Solving bot project, gaining knowledge of data structures in c and RTOS.
- Contributed to Autonomous Vehicle project, utilizing embedded system, C++, and understanding sensor interfaces with I2C, SPI, ADC, UART protocols.
- I have been exploring Zephyr Driver Development and have gained sufficient knowledge to start the implementation.
- In all of the above projects, I designed many things from scratch and rapidly learned many things in embedded systems. Even though I faced many errors, I stayed committed and figured them out. I have relevant experience (C, RTOS) for this project, and as a passionate Open Source enthusiast, I will wholeheartedly work towards completing the project idea within the allotted time.
- **Contribution in openbeagle.org/gsoc**
  - Improved Documentation - #40 (merged)

4.1 Contingency

I believe that if I get stuck on my project and my mentor isn't around, I will use the resources that are available to me. Some of those information portals are listed below.

- Zephyr Project Documentation
- J721E /TDA4VM Technical Reference Manual
- remoteproc
- Tutorial: Mastering Zephyr Driver Development
- I will ask on the Beagleboard and Zephyr forums or Discord.

4.2 Benefit

- With Zephyr support, developers gain access to a rich ecosystem of Zephyr-compatible libraries, drivers, and tools, opening up new possibilities for creating innovative projects and applications on the BeagleBone AI-64 R5 platform.
- Zephyr's lightweight and real-time capabilities potentially improve the BeagleBone AI-64 R5's performance in many future projects and applications.

4.3 Misc

The PR for Cross Compilation task: #192.