Improve the Raspberry Pi 4 BSP

Google Summer of Code Program 2023 Project Proposal

Utkarsh Verma
utkarsh@bitbanged.com
Indian Institute of Information Technology, Design and Manufacturing - Kancheepuram
Chennai, India

Project Abstract

The existing support for Raspberry Pi SBCs in RTEMS is operational, but it lacks essential functionalities such as graphics support, USB, SD card, UART, I2C and SPI functionality, which are necessary for basic user requirements. To address this gap, this project aims to incorporate these controllers into the Raspberry Pi 4 (Model B) board-specific package within RTEMS.

Project Scope

Large (approx 350 hours)

Project Description

The objective of this project is to improve the "raspberrypi4b" BSP's functionality in RTEMS by including support for critical features such as:

- SD card R/W access
- GPIO, UART, SPI, and I2C drivers for interfacing with sensors or other peripherals
- Graphics framebuffer support without using libbsd

The current BSP's inability to support these basic features limits its effectiveness in the embedded field. The successful implementation of these drivers would significantly expand the capabilities of RTEMS on the Raspberry Pi 4B and open numerous doors for its use in the embedded industry.

Project Deliverables

- **May 29 (Coding begins)**:
  I intend to first implement in-system programming support for the Raspberry Pi and optimize my hardware and software development workflow.

- **July 10-14 (Midterm evaluation)**:
  By the midterm evaluation, I will have fleshed-out support for I2C, SPI and SD card R/W access in the BSP. I2C and SPI will be tested with a ping-pong using a slave device with the RPi as the master. For the SD card, the test can be as simple as saving some data to a sector and accessing it later.
- **August 21 - 28 (Final evaluation)**:
  By the final evaluation, I plan to finalize support for the graphics frame buffer in the BSP. It will be tested by having the RPi display a sample image on an HDMI screen.
- **Post GSoC**:
  While my time with RTEMS may be reduced after GSoC due to higher education plans, I would still like to remain a part of the community and participate in discussions on the forums. I am passionate about firmware development and intend to contribute as much as possible in the future.

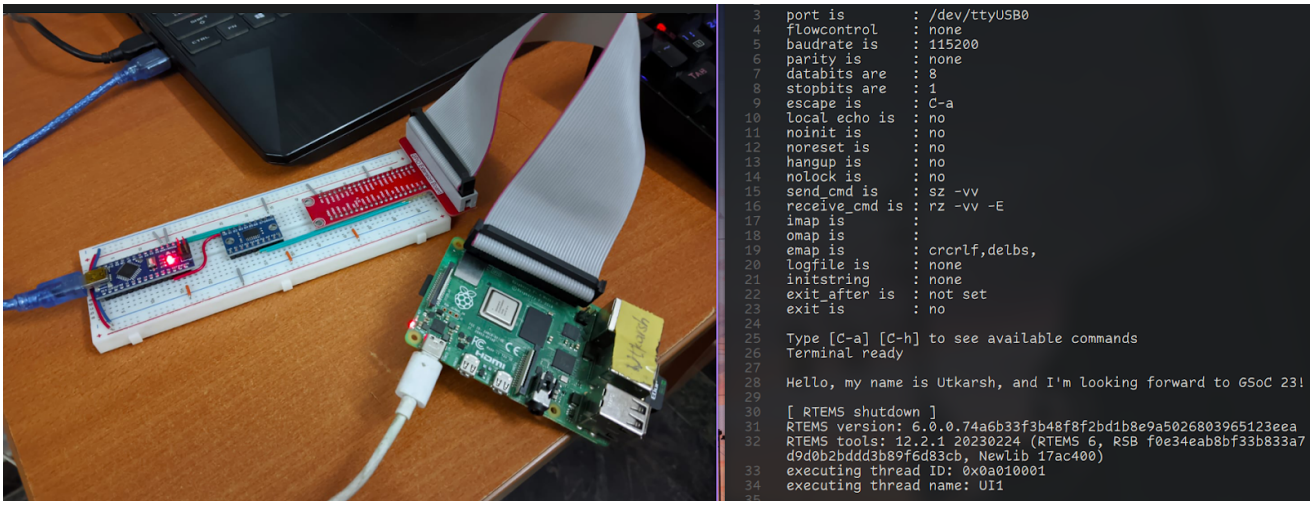
Proposed Schedule

Application period (Mar 20 - Apr 4):

- Gain familiarity with the codebase and clarity about the project.
- Explore the current status of the BSP on the RPi.
- Iron out the proposal and get feedback from prospective mentors about the project.

Acceptance waiting period (Apr 4 - May 4):

Currently, I have been able to run RTEMS on my RPi 4 board with the following setup. I've used Arduino Nano as a serial adapter for now with a level shifter.



However, I would like to build upon the existing debugging workflow and implement in-system programming (ISP) on the RPi. This will likely be a collaborative effort with another RPi BSP developer.

During this period I would solely focus on researching possible approaches, a few of which are listed below:

- One option is to go with U-Boot. It will expose a shell over UART which we could potentially utilize for uploading the RTEMS kernel. The baud rate would be 115.2 Kbps which is reasonable enough for the small-sized RTEMS kernel.
- The other option is to use JTag through the SWD. The FT232H might be a good choice since it has UART, SPI, and I2C. This method was used in last year's GSoC project at RTEMS. Still, research needs to be done on various JTag debugger options based on cost and speed.

Community bonding period (May 4 - May 28):

- Based on my findings, I will start working towards the ISP implementation for the board.
- I will explore the current status of Arasan's sdhci implementation in RTEMS-libbsd with respect to the RPi 4 BSP.
- I will also look into the existing frame buffer implementations within RTEMS.

First half (May 29 - Jul 14):

I will be majorly focusing on adding support for basic peripherals to the BSP and then finally the SD card read/write access, which would be taking a major chunk of time. For specifying the peripheral hierarchy, I will be using flattened device trees (FDT). This would be a more readable approach than the current method of defining addresses as macros.

- GPIO:
    - Define the GPIO interfaces provided by the board as `rtems_gpio_pin_conf` structs.
    - Write basic GPIO utility functions to access/modify the pin state.
- UART:
  The BSP already supports transmitting over the PL011-based UART0 or "/dev/ttyS0" port. However, the additional four UART ports on the RPi 4 are not supported yet. Here is how I will implement support for these:
    - Explore the arm-pl011 driver in the RTEMS tree and implement Rx support for UART0.
    - Write a driver for miniUART to allow accessing UART1. This would involve figuring out which addresses in memory its registers are mapped to and defining functions to access/modify them.
    - Generalize existing console implementation to fetch hardware metadata from the device tree. This would allow users to easily swap between different UART ports, with the same code.
- SPI:
    - Add support for interrupts to allow handling of SPI interrupts raised by the board.
    - Leverage the RTEMS port of Linux's spi-bus API and define functions to configure the SPI controller. For example, configuring the clock, master/slave operation etc.

    - Test bi-directional transfers and look into support for write-only devices which the RPi 1 BSP doesn't support.
- I2C:
    - Leverage the RTEMS port of Linux's i2c-bus API and define functions to configure the peripheral, just like SPI.
    - Write interrupt handlers for the I2C communications.
    - Test 10-bit addressing with an Arduino slave with bit-banged I2C.
- SD card driver:
  RTEMS already has support for the sdhci driver, and the RPi 1 BSP has initial support for it, but it does not implement SD card interrupts. Here is how I will approach the development:
    - Gain familiarity with the SD/MMC protocol and the host controller hardware and software interface.
    - Determine the block size, transfer speed, and command set.
    - Implement the command/response functions for the driver based on the SD card specification.
    - Implement the block transfer functions for the driver, which should read and write blocks of data to/from the SD card.
    - Refer to other SD card drivers (e.g. Linux's sdhci) and make improvements. The SD card spec is not fully open. Hence, this step will be crucial for ensuring compatibility.
    - Test and debug.

Second half (Jul 14 - Aug 28):
The second half would be completely dedicated to writing a non-libbsd frame buffer driver for the RPi 4 BSP. There are implementations in the RTEMS tree and one in the ARM BSP for Raspberry Pi 1, both of which would serve as great references. Here is how I plan to develop the driver:

- Develop a driver for the mailbox peripheral to get IO access to the VideoCore multimedia processor on the RPi.
- Fix a target display size of 1024x768 with 32-bit depth, and allocate memory accordingly for a double-buffered frame buffer.
- Using the SD card driver, get preprocessed data from it, and render it onto the display. This can also be done for playing a video.
- Implement basic font rendering using a cached bitmap of the glyphs.

Future improvements

A driver for the VL805 USB controller on the RPi could be ported to RTEMS. This would enable implementing support for RTEMS' console over HDMI. It would be a nice extension of the frame buffer support.

Apart from that, DMA transfers for I2C, SPI, and SD card drivers would be a nice addition as well.

This BSP could also potentially be extended to support Raspberry Pi Compute 4 module, which is pretty similar. With the newly-added support for peripheral bus, I could open the doors of industrial usage for RTEMS.

Continued involvement

While I will try to put my best efforts into pushing bug-free drivers upstream, in reality, some bugs do get through. Therefore, I will take charge of maintaining the drivers that I develop even after the internship ends.

Conflict of interest or commitment

I will not be having any commitments during the internship period which will allow me to direct my complete attention to this project.

Eligibility

Yes

Major challenges foreseen

Implementing a robust driver for the SD card could turn out to be a bit challenging because the specification is not fully open. Hence it usually boils down to some guess-work or reverse engineering existing implementations of the well supported drivers.

Apart from that, this would be my first time developing drivers for the RPi boards. Hence, being new to the board, I might end up facing unexpected delays. But, I'm pretty confident that it will be something I can take care of with my mentor's guidance and my prior experience in firmware development.

References

RTEMS

https://devel.rtems.org/wiki/GSoC/2015/RaspberryPi_peripherals_and_SD_card
https://git.rtems.org/rtems/tree/bsps/arm/raspberrypi
https://git.rtems.org/rtems/tree/cpukit/include/rtems/framebuffer.h

Others

linux/mmc_spi.c at master · torvalds/linux · GitHub

Kingston microSDXC Memory Card Flash Storage Media
https://github.com/raspberrypi/firmware/wiki/Mailbox-property-interface
ARM: dts: bcm2711: Use bcm2711 compatible for sdhci - Patchwork

Blogs

Setup OpenOCD with JTAG + UART on raspberry pi 4 using FT232H
Loading Linux Images over UART
Lecture 12: SPI and SD cards
Writing a “bare metal” operating system for Raspberry Pi 4 (Part 4) | rpi4-osdev
Writing a “bare metal” operating system for Raspberry Pi 4 (Part 5) | rpi4-osdev
The Mailbox Peripheral | Building an Operating System for the Raspberry Pi
Raspberry Pi 4 - Device Tree

Relevant background experience

I have extensive experience working on projects utilizing AVR chips from ATmel, using both C and Assembly languages. Additionally, I have recently started working with STM32F7 boards for ARM development. These projects have given me a deep understanding of the low-level interfaces present in microcontrollers.

I have done a very similar project with the ATmega328P chip where I interfaced an SD card to an OLED display over I2C to play a video. It uses a 1KB framebuffer and I did everything in AVR assembly. Hence, I am pretty confident about this project. I’ve open-sourced that project on GitHub.

Moreover, I have built an 8-bit computer on breadboards, which has provided me with clarity on computer architecture. As a Linux user for over 5 years, I have also gained familiarity with the kernel and developed my software development skills. I believe these experiences would surely help me with this project.

Details about my hobby projects can be found here.

Personal

I'm an electronics engineering student at IIITDM Kancheepuram in Chennai, India, set to graduate in May. Since 10th grade, I've built numerous hardware projects and developed a passion for exploring how technology works. Embedded systems allow me to fully understand complex systems hands-on, which strikes the perfect balance between my interests in electronics and software.

When I discovered RTEMS while searching for GSoC organizations in the embedded systems domain, I knew it was the perfect fit for me. The opportunity to write drivers for embedded systems aligned perfectly with my interests and skills.

As a self-driven and tenacious individual, I have honed my skills through various personal projects, and I am confident in my ability to make meaningful contributions to the RTEMS project. I am highly motivated to take on this GSoC opportunity and excited about the potential impact I can make.

Experience

Free software experience

My first encounter with FOSS was during Google Code-in back when I was in 11th grade. Since then, I am very grateful to FOSS and the community in general. The open nature of these projects has taught me a lot about coding and problem-solving approaches. Hence, I am highly motivated to give back to the FOSS community, and I have done so through various contributions, a few of which are:

- dense-analysis/ale: Add support for AVRA linting
- fairyglade/ly:
    - Make xinitrc path configurable
    - Use XDG_RUNTIME_DIR for storing Xauthority
- jarun/nnn: Allow specifying preview width
- gohugoio/hugoDocs: Add docs for shimming JS libraries

Language skill-set

I prefer coding in statically-typed languages as it helps a lot in systems programming. Here are the languages I currently know, listed in decreasing order of experience:

- C (Intermediate)
- AVR Assembly (intermediate)
- Shell scripting (intermediate)
- Go (intermediate)
- Python (intermediate)
- HTML, JS, CSS (intermediate)
- Rust (Beginner)
- Verilog (Beginner)

Apart from these languages, I like using Makefiles to build my embedded projects and I am also pretty comfortable with them.

Work experience

I have worked on three internships to date, during my Bachelor's degree, which are listed below in chronological order:

- Embedded Systems Intern @ Dextroware Devices, India:
    - Worked on a wireless USB input device called Mouseware
    - Wrote firmware for the transceivers and designed schematics
- Research Intern @ GSI Helmholtz Centre for Heavy Ion Research, Germany:
    - Worked on improving the accuracy of the FPGA-based data acquisition system in the lab
    - Wrote a command-line tool to efficiently and quickly perform the analysis.
- Embedded Systems Intern @ Aerospace Engineers, India:
    - Tasked with integrating sensors in their AUV's embedded stack
    - Wrote the firmware for the STM32 boards using STM32 HAL with a custom build system using Make.

Web URLs

Personal website: https://utkarshverma.me
Blog: https://bitbanged.com
Discord - Barusu#4230
GitHub: https://github.com/UtkarshVerma