A case for a generic Linux Driver for connecting MCUs to MPUs

Andrew Davis
afd@ti.com

Schuyler Patton
spatton@ti.com

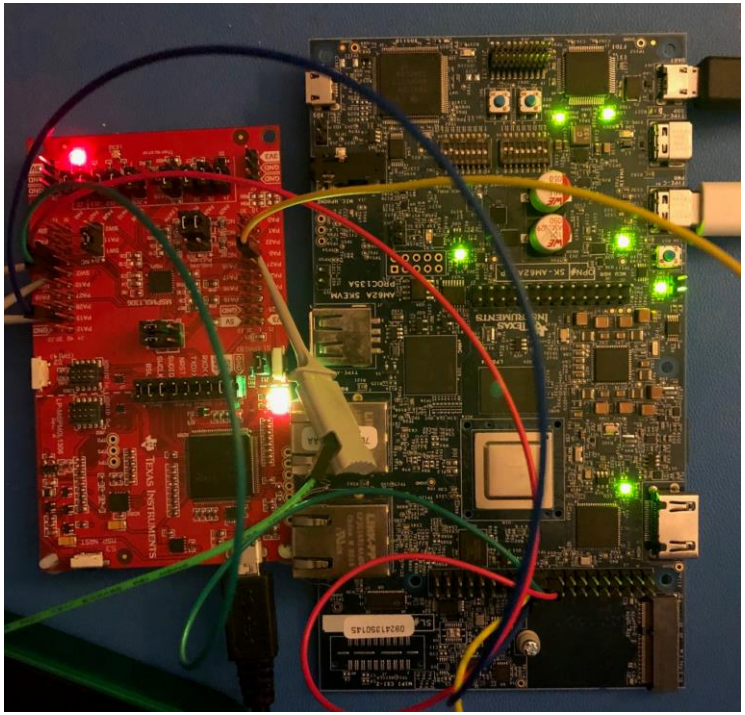
20 Sept 2024

About Us

- Andrew Davis
  - Open Source Product Technology team
  - Software and hardware architecture
  - New enabling technologies
  - Ease of use, open standards, open source ecosystems, and community engagement
- Schuyler Patton
  - Sitara MPU Systems and Applications team
  - Embedded Linux applications on Sitara devices
  - Plugging in Sitara MPU processors into other processors, peripherals
  - Networking support

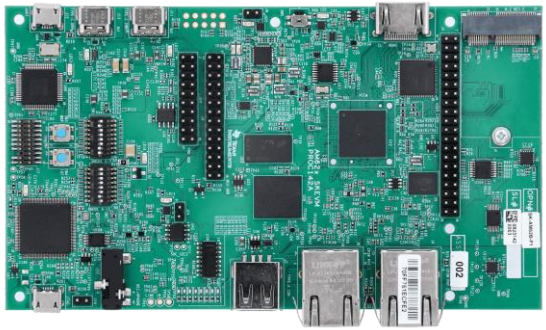
MPU + MCU attach for IOT applications

MPU = Micro-Processor Unit MCU = Micro-Controller Unit

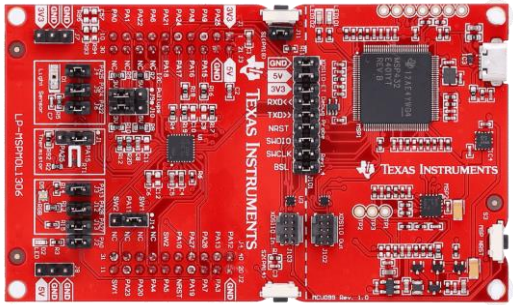


- Connecting the two processor types is not a new concept for IOT applications
- Various applications from existing applications to upgrading system concepts, offloading etc.
- Getting a MPU/MCU application integrated faster is always a desire.

MPU + MCU – Defining the two processor types



1400 GHz Arm® Cortex ®-A53 – multi-core 64-bit
Up to 8GBytes with DDR4

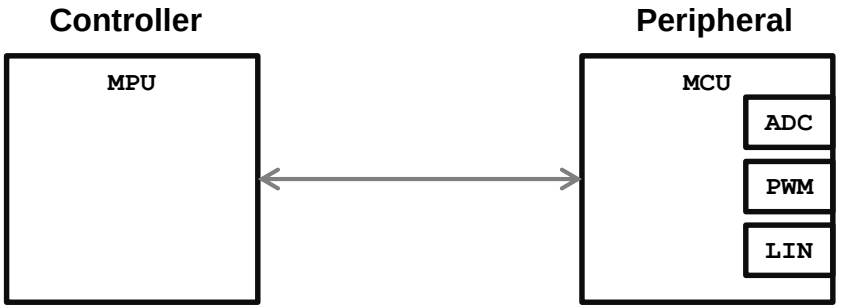


32-MHz Arm® Cortex®-M0+ MCU 32bit
64-KB flash,
4-KB SRAM

- MPU – Micro-Processor Unit
  - Running a OS like Linux
  - Has DDR
  - Applications not restricted on memory size
  - Lots of storage
  - Higher speed processors

- MCU – Micro-Controller Unit
  - Applications are very limited memory size for code and data
  - Slower speed processor

MPU + MCU attach – Application examples/reasons



MCU App that is front-end for an MPU app

- EV Charging offload
  - PWM
  - ADC
  - LIN
- Touchscreen controller
- Wireless protocols (Sub 1GHz)
- Low Latency apps
- Peripheral mis-match
- Extension of peripherals on not available MPU processor

MPU + MCU attach – Connecting the two processors
(Hardware)

- Selecting available interfaces
- MPU – Controller, MCU - Peripheral
  - CAN
  - I2C
  - SPI
  - UART

MPU + MCU attach – Connecting the two processors
(Software)

- MPU – Controller,
  - CAN (network – can0)
  - I2C (/dev/i2c)
  - SPI (/dev/spidev1.0)
  - UART (/dev/ttyS3)
- MCU - Peripheral
  - Application using MCU SDK Framework and Driver lib support
    - CAN
    - I2C
    - SPI
    - UART

MPU + MCU attach – Connecting the two processors
(Software)

Lots of options, could one interface abstraction be used?

- MPU – Controller
  - CAN (network – can0)
  - I2C (/dev/i2c)
  - SPI (/dev/spidev1.0)
  - UART (/dev/ttyS3)
- MCU - Peripheral
  - Application using MCU SDK Framework and Driver lib support
    - CAN
    - I2C
    - SPI
    - UART

MPU + MCU attach – A method for quick access

```
stty -F /dev/ttyS3 115200
cat /dev/ttyS3
```

- What we want:
  - Somewhat quick (and lazy) way to quickly to get information from the MCU to a Linux application
  - Could using the tty interface as a quick very common interface that abstracts the actual connecting interface
  - Can be accessed without having to an application for a quick test

MPU + MCU attach – A method for quick access

```
stty -F /dev/ttyS3 115200
cat /dev/ttyS3
```

- What we want:
  - Somewhat quick (and lazy) way to quickly to get information from the MCU to a Linux application
  - Could using the tty interface as a quick very common interface that abstracts the actual connecting interface
  - Can be accessed without having to an application for a quick test
  - Relieves Linux application developers of requiring a dedicated driver for the interconnection.

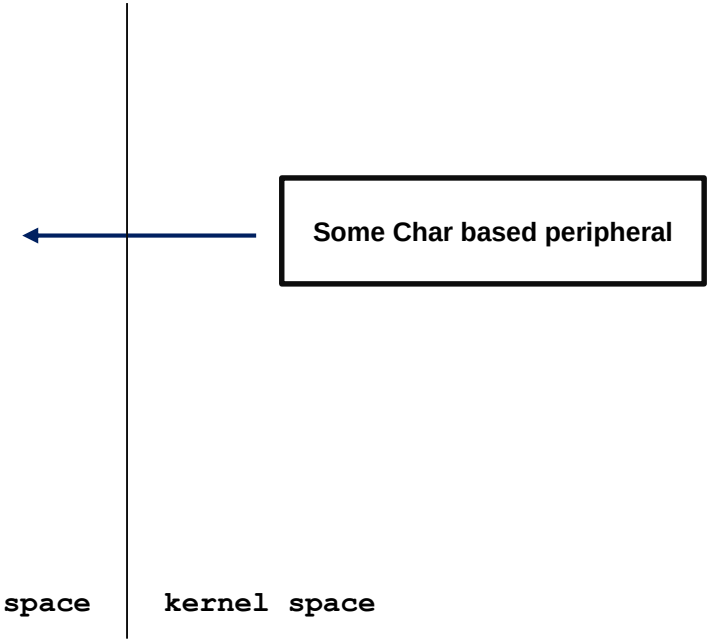
Why UART?

- UART interface for applications is the probably the simplest common communication interface.
- UART examples are plentiful on the web
- Data can be human readable if needed
- Lots of existing tooling for quick testing and control (cat, stty, etc..)

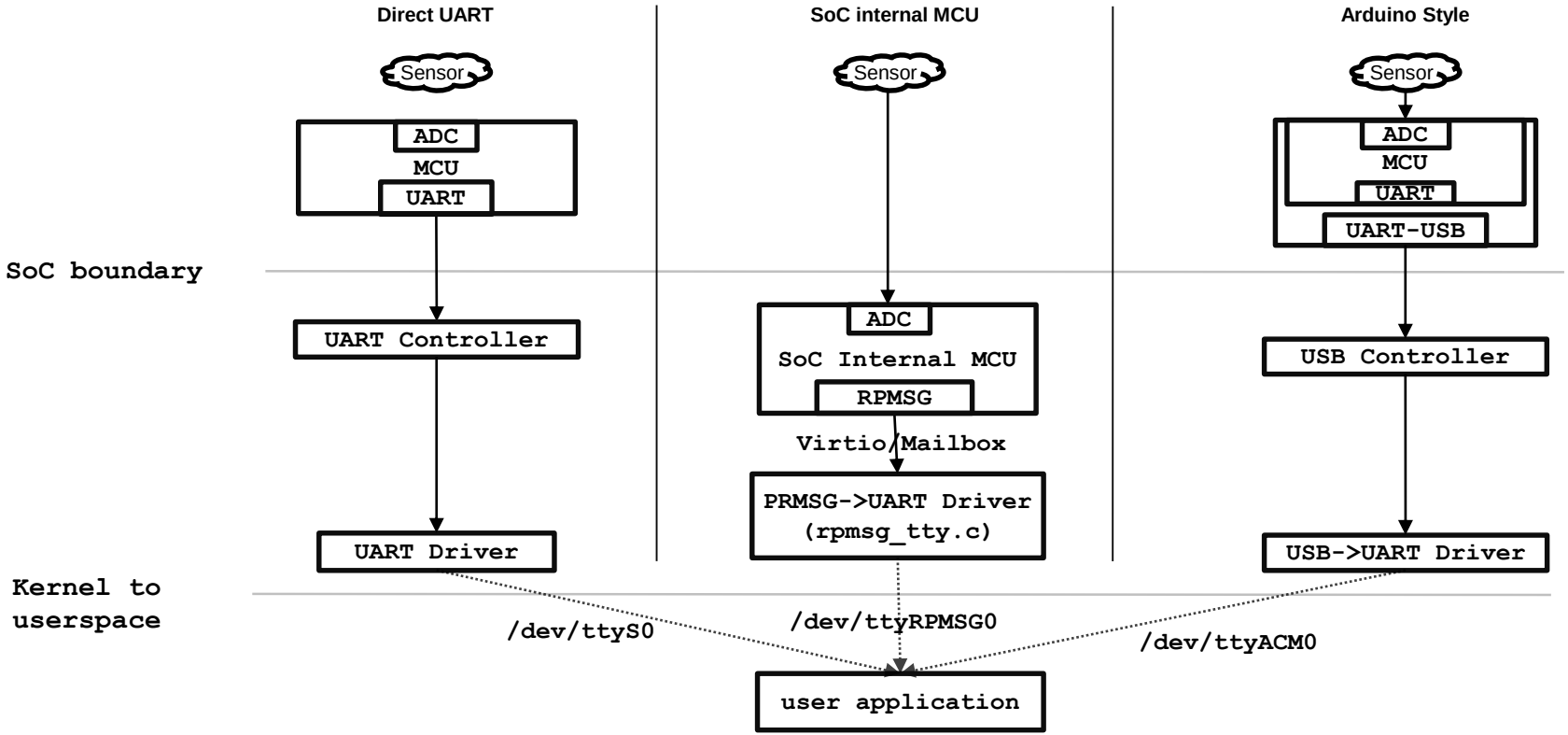
Need to read a peripheral…

user_application.c

```
main () {
char read_buf[RBUF_SIZE];
int  read_cnt = 0;
int  err_flag;


fd = open("/dev/ttyVU0",O_RDONLY | O_NOCTTY| O_SYNC);

while (1) {

  rd_cnt = read(fd,&read_buf,RBUF_SIZE);

  if (rd_cnt > 0)
     do_the_thing_that_needs_doing();

  sleep(an_amount_of_time);

}
```



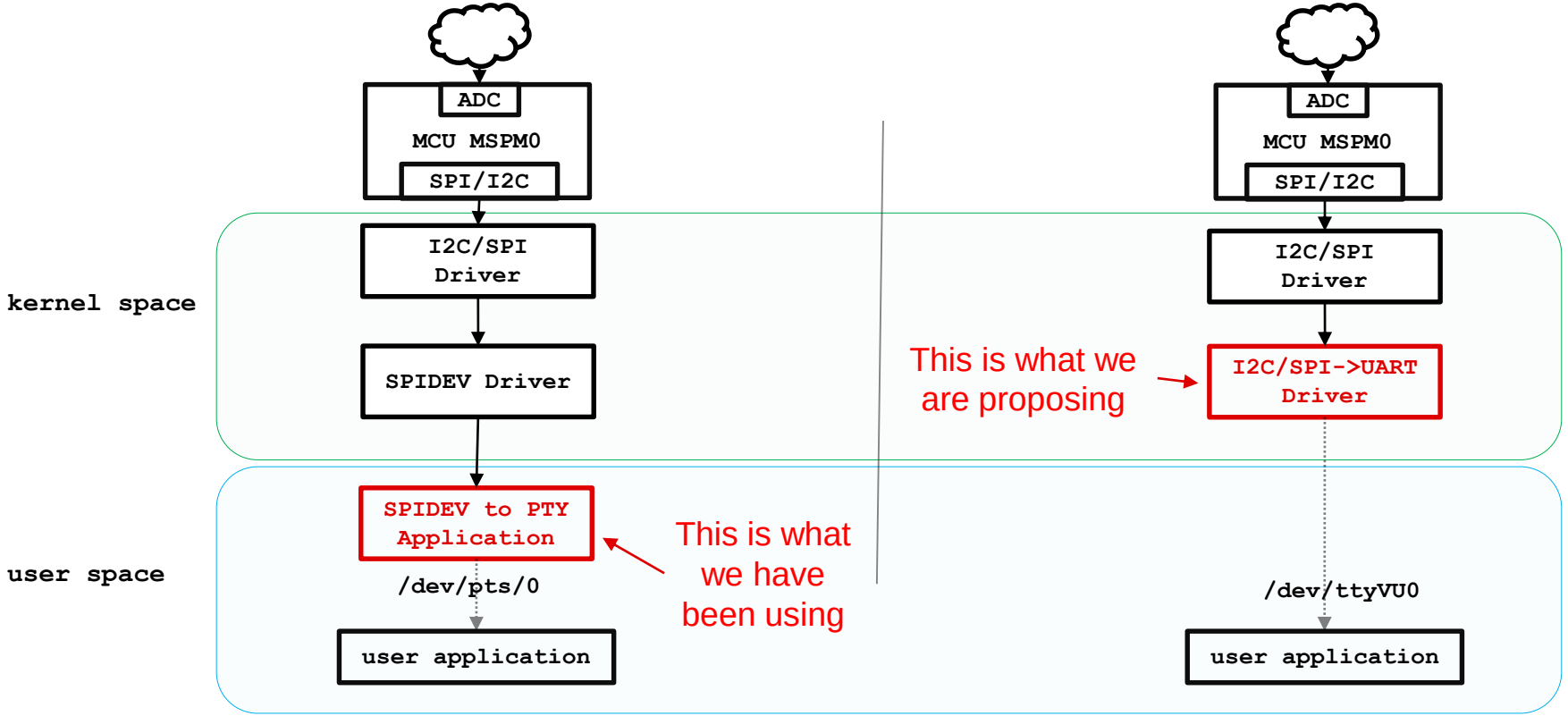
Some current connected MCU solutions



Why UART abstraction for SPI, I2C Use Cases

- Hardware protocol converters like USB<->UART add board cost
- Might want chip select like SPI
- Might want to daisy-chain like I2C
- Still want to keep the same application code (UART based)
  - UART works fine for simple data
  - Don’t want to re-write our all our tools/apps to use spidev or special I2C ioctls
- What do we do?:
  - Connect to the MCU with SPI/I2C physically but with UART interface to software

UART over SPI/I2C connected MCU



Userspace vs Kernel driver

- Userspace would need new interrupt support
  - Do not want to continuously poll device
  - Extend SPIDEV to support IRQs?
  - Could the current SPIDEV and I2C userspace interface be more network like (CAN)?
    - Might allow more standard addressing (CS / I2C address -> AF_SPI)
    - recv() to block for next message vs polling
- Kernel driver needs common encapsulation wire format over SPI/I2C
  - Control signals? RTS/CTS or XON/XOFF
  - Translate to I2C clock stretching?
  - How do we describe the MCU with DTS?

Summary

- Connecting the two processor types is not a new concept for IOT applications
- Want to provide a way to connect an MCU and MPU
- Want to use same Linux application code through a UART abstraction that allows different common peripheral interfaces as the interconnect
- Do we want a kernel level driver to make this UART abstraction standard?