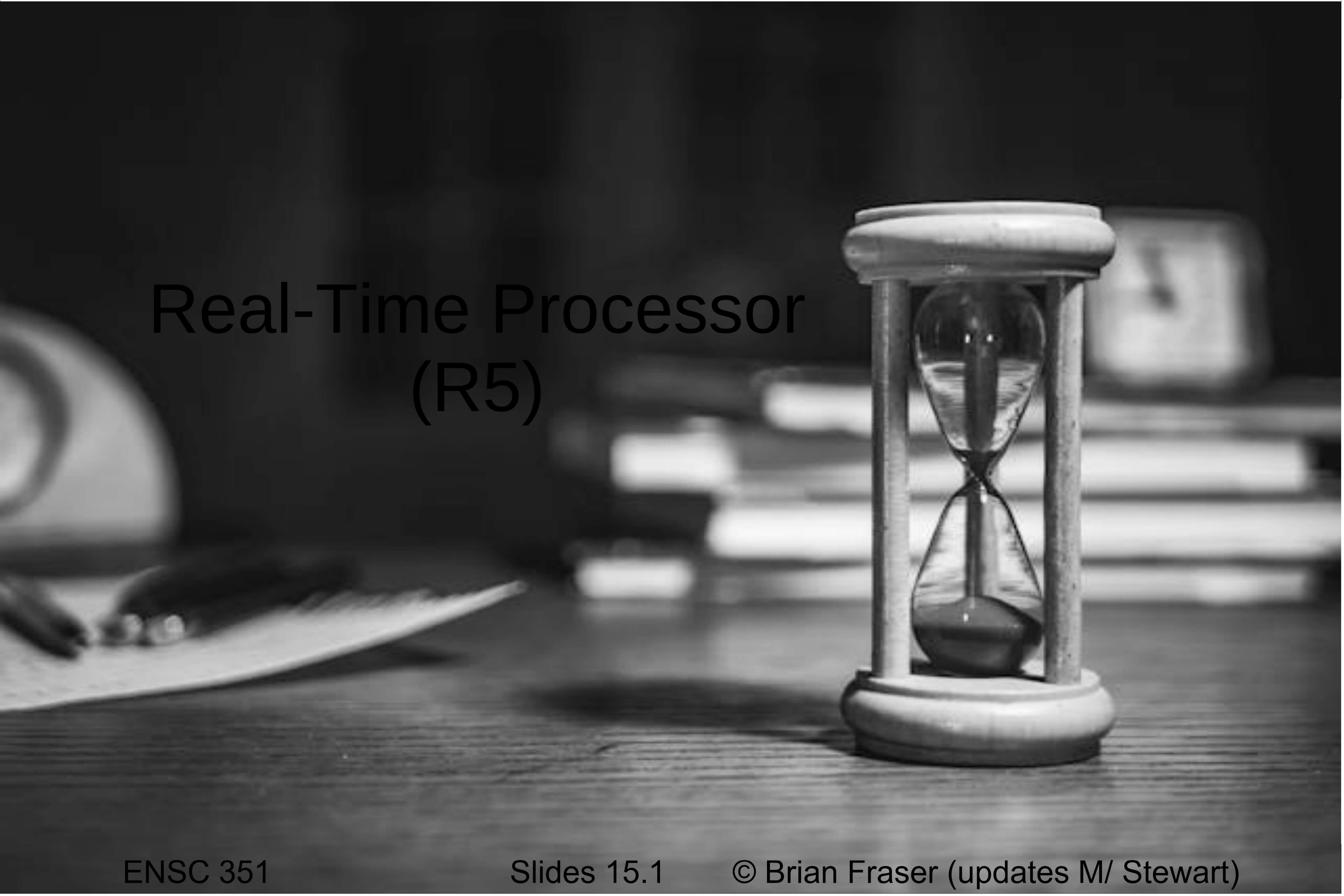


Real-Time Processor (R5)

Topics

1) How can we do hard real-time on the BYAI?
2) How can we get our code into the R5?
3) How can we use GPIO with the R5?

Hard Real-Time with BYAI:
About the R5

Our Definitions of Realtime

- Hard Real-time
  - User requires..
  - "Guaranteed Services"
    Mathematical/logical proof or exhaustive simulation required
  - Hard real-time is about..
- Soft Real-time
  - User only requires..

    (statistical analysis)
  - "Best effort Services"
    E.g.: Average # missed deadline < 2 per minute.
  - Soft real-time is about..

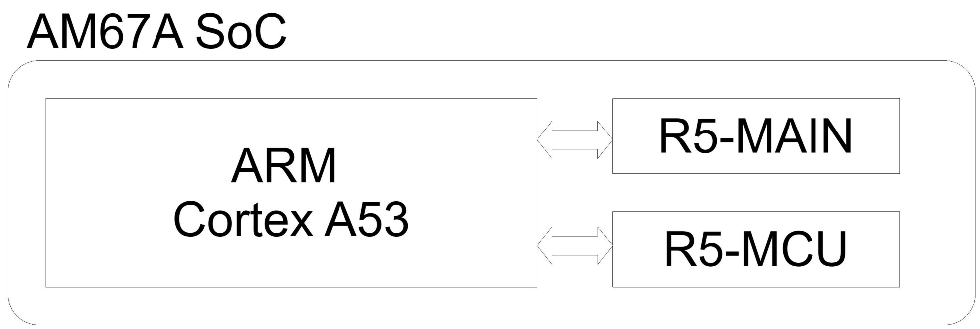
Motivation

- Linux can do fine with soft real-time tasks
  - ~10s of ms accuracy/latency
- Hard real-time
  - Cannot use Linux:
    ..
    but, Linux gives us great software power!
  - Analyze system and
    ..
    from soft real-time and non-real-time
- Hard real-time task solution
  - Run hard real-time tasks on a processor without Linux
    - Use external processor (Arduino)
    - Use internal microcontroller (R5)

R5F Sub System

- The BeagleY-AI has a
  Texas Instruments AM67A SoC..
  - Powerful main processor:
    Arm Cortex-A53 (quad-core, 1.4GHz)
  - 2 Dedicated real-time processors:
    Arm Cortex-R5 (single-core, 800MHz)
  - 2 general purpose digital signal processors (DSPs):
    C7x DSP (4-TOPS with Matrix Multiply Accelerator)
- We’ll use:
  - ..
  - ..
    GPIO for NeoPixel LED

R5



- R5 is useful for hard real-time interactions, such as:
  - deterministic latency to respond to GPIO event
  - .. GPIO and ..
    (Ex: timing ultrasonic distance sensor readings)
  - .. (manual) protocol implementation:
    UART, I2C, SPI, Neo-pixel 1-wire protocol, etc.
- Called the Cortex R5F Subsystem (R5FSS)
  - The *F* means it has a hardware floating point unit.

R5 Architecture

- 3 R5’s:
  - Dedicated R5 core to manage boot, power modes
  - Available R5 for running custom code.
  - Available R5; able to access different GPIO pins.
- Each R5:
  - 32-bit ARM processors, 800MHz
  - 2-banks of fast (uncached) memory:
    ..
- Resources
  - Can share RAM banks with Linux
  - Each R5F allows access to different GPIO pins
  - Peripheral access (UART, etc)
  - And more! (interrupts, ...)

Interface to R5

- Linux's Remote Processor Framework
  - ..
  - called remoteproc
- (byai)$ **ls /sys/class/remoteproc/**
  - remoteproc0 - DSP #1
  - remoteproc1 - DSP #2
  - remoteproc2 - R5 - MCU
  - remoteproc3 - R5 - WKUP
  - remoteproc4 - R5 - Main
- View status of processor by viewing its state file

  (byai)$ cat /sys/class/remoteproc/rempoteproc3/state

Coding for the R5

Sample Program

```c
#define DELAY_TIME_uS   (250 * 1000)

// Device tree nodes for pin aliases
#define LED0_NODE DT_ALIAS(led0)
#define BTN0_NODE DT_ALIAS(btn0)

static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
static const struct gpio_dt_spec btn = GPIO_DT_SPEC_GET(BTN0_NODE, gpios);

static void initialize_gpio(const struct gpio_dt_spec *pPin, int dir) {

  gpio_is_ready_dt(pPin);
  gpio_pin_configure_dt(pPin, dir);

}
```

```c
int main(void) {
    printf("Hello World! %s\n");

    initialize_gpio(&led, GPIO_OUTPUT_ACTIVE);
    initialize_gpio(&btn, GPIO_INPUT);

    while (true) {
        gpio_pin_toggle_dt(&led);

        k_busy_wait(DELAY_TIME_uS);

        // Delay longer when button pressed
        if (gpio_pin_get_dt(&btn) == 0) {
            k_busy_wait(DELAY_TIME_uS);
        }
    }
    return 0;
}
```

Build Process

- Our R5 programs will use..
- Setup
  - Install the Zephyr SDK & Tools
    SDK = ..
- On Host
  - Build project using CMake
    Builds build/zephyr_mcu.elf
  - Copy to NFS
- On Target
  - Copy zephyr_mcu.elf to /lib/firmware
  - Load into R5:
    echo zephy_mcu.elf | sudo tee /sys/class/remoteproc/remoteproc2/firmware
    echo start | sudo tee /sys/class/remoteproc/remoteproc2/state

GPIO with R5

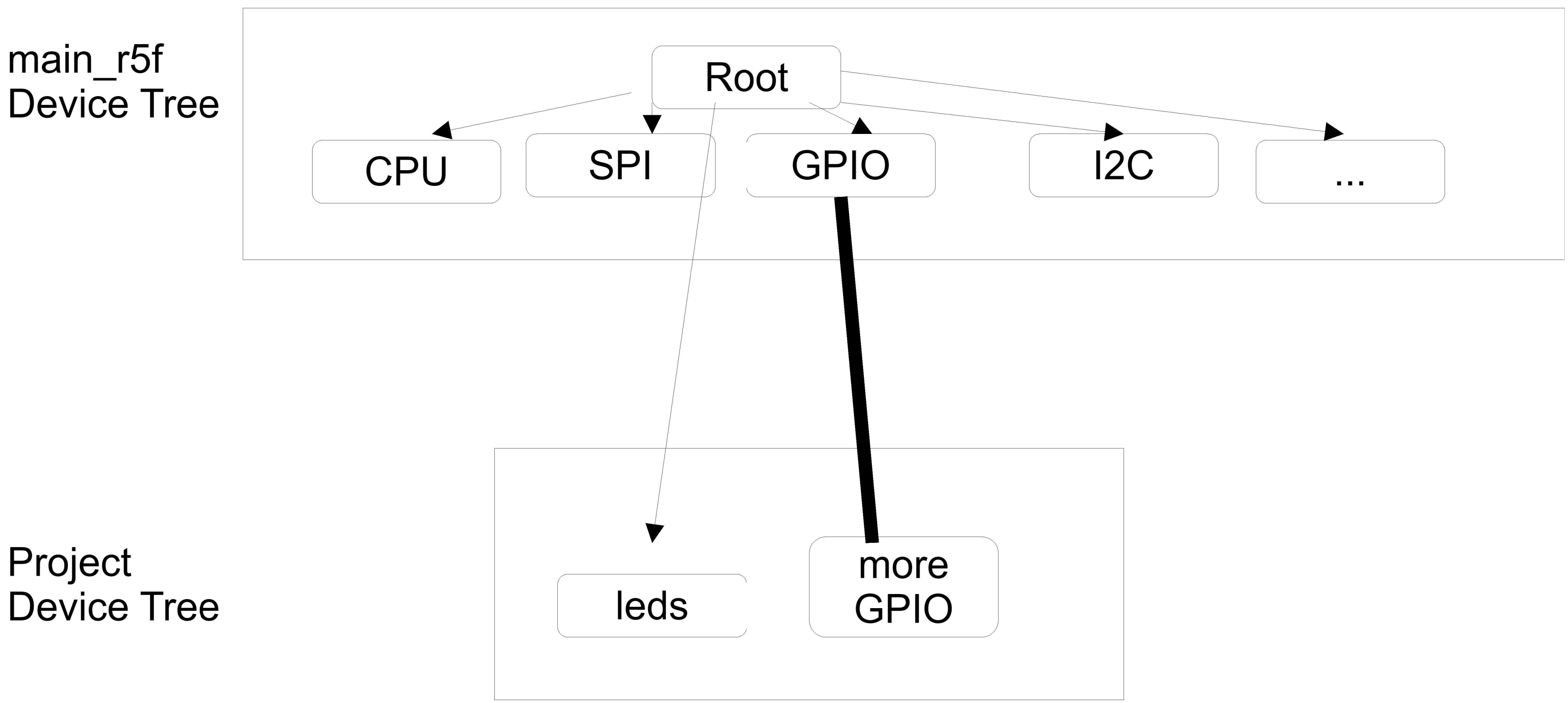
Overview

- C code refers to pins by name.

```
// Device tree nodes for pin aliases
#define LED0_NODE DT_ALIAS(led0)
static const struct gpio_dt_spec led = GPIO_DT_SPEC_GET(LED0_NODE, gpios);

int main(void)
{
    gpio_is_ready_dt(&led);
    gpio_pin_configure_dt(&led, GPIO_OUTPUT_ACTIVE);

    while (true) {
        gpio_pin_toggle_dt(&led);
        k_busy_wait(250 * 1000); //uS
    }
    return 0;
}
```

- ..

Device Tree

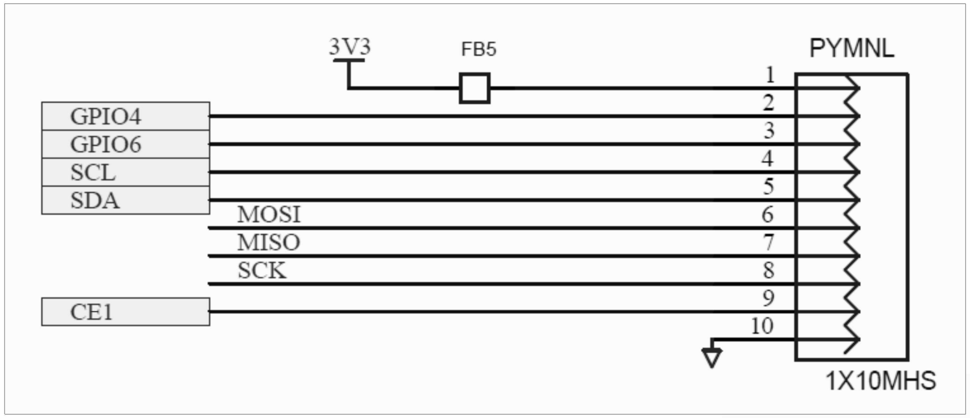

main_r5f
Device Tree
Root
CPU
SPI
GPIO
I2C
...
Project
Device Tree
leds
more
GPIO

My project’s device tree overlay.

```
/ {
    aliases {
        led0 = &mcu_led;
    };

    leds {
        compatible = "gpio-leds";

        mcu_led: led_gpio7 {
            gpios = <&mcu_gpio0 9 GPIO_ACTIVE_HIGH>;
        };
    };

    pinctrl_mcu: pinctrl_mcu@4084000 {
        compatible = "ti,k3-pinctrl";
        reg = <0x04084000 0x88>;
        status = "okay";
    };
};

&pinctrl_mcu {
    mcu_gpio0_led_btn_default: mcu_gpio0_led_btn_default {
        pinmux = <K3_PINMUX(0x0024, PIN_OUTPUT, MUX_MODE_7)>; /* (B3) GPIO7*/
    };
};

&mcu_gpio0 {
    pinctrl-0 = <&mcu_gpio0_led_btn_default>;
    pinctrl-names = "default";
    status = "okay";
};
```

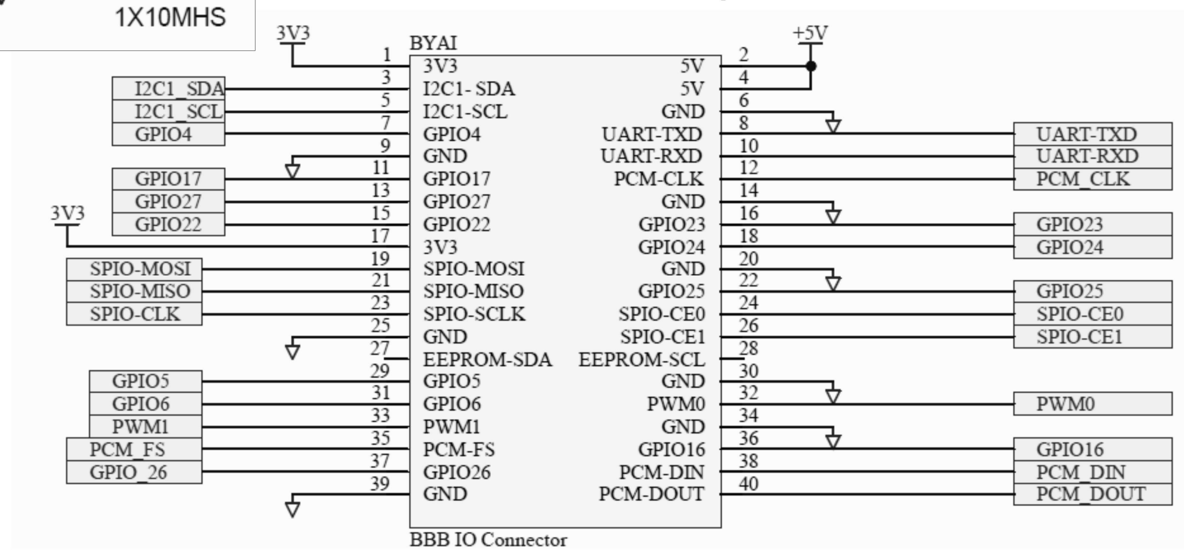
Demystifying GPIO

- How do we know what to put into a device tree?
  - On schematic, identify pin:



CE1 is PYMNL-Pin 9
Maps to SPI0-CE1

SPI0-CE1 maps to
hat-pin 26



Zen Hat Pinout

Zen Hat Pinout

Exposed Heade	Hat Use		Use	Name	Connector		Name	Use		Hat Use	Exposed Header
GPIO Header				3v3	1	2	5v				
GPIO Header	Accelerometer, ADC, Audio		I2C1 - SDA	GPIO2	3	4	5v				LED Strip
GPIO Header	Accelerometer, ADC, Audio		I2C1 - SCL	GPIO3	5	6	GND			UART Header	UART Header
GPIO Header		X		GPIO4	7	8	GPIO14	UART - TXD	X	UART - TXD	UART Header
GPIO Header				GND	9	10	GPIO15	UART - RXD	X	UART - RXD	UART Header
	Rotary Encoder - EQEP0_B			GPIO17	11	12	GPIO18	PCM - CLK		Audio - BCLK	
	LCD - DC			GPIO27	13	14	GND				LED Strip
	LCD - RST			GPIO22	15	16	GPIO23		X	NeoPixel Data Out (R5 GPIO)	LED Strip
				3v3	17	18	GPIO24			Encoder Push-button (R5 GPIO)	
GPIO Header	LCD - DIN		SPI0 - MOSI	GPIO10	19	20	GND				
GPIO Header		X?	SPI0 - MISO	GPIO9	21	22	GPIO25			LED	
GPIO Header	LCD - CLK		SPI0 - SCLK	GPIO11	23	24	GPIO8	SPI0 - CE0		LCD - CS	
				GND	25	26	GPIO7	SPI0 - CE1	X		GPIO Header
	<only I2C, but prefer I2C1>		EEPROM - SDA	GPIO0	27	28	GPIO1	EEPROM-SCL		*<only I2C, but prefer I2C1>*	
	Joystick Push-button			GPIO5	29	30	GND				
GPIO Header		X		GPIO6	31	32	GPIO12	PWM0		LED Emitter	
	LCD - Backlight (BL)		PWM1	GPIO13	33	34	GND				
	Audio - WCLK		PCM - FS	GPIO19	35	36	GPIO16			Rotary Encoder - EQEP0_A	
	Audio - Reset			GPIO26	37	38	GPIO20	PCM - DIN		Audio - DIN	
				GND	39	40	GPIO21	PCM - DOUT		Audio - DOUT	
		X	Available GPIO								

Checkout info on Hat Pin

- Use website to investigate a hat pin https://pinout.beagleboard.io/

PAD Config Register

- We know we are working with Ball Number B3
  - Look it up in processor data sheet: https://www.ti.com/lit/ds/symlink/am67a.pdf

BALL NUMBER [1]	BALL NAME [2] PADCONFIG Register [15] PADCONFIG Address [16]	SIGNAL NAME [3]
B3	WKUP_UART0_RXD PADCONFIG MCU_PADCONFIG9 0x04084024	WKUP_UART0_RXD
		MCU_SPI0_CS2
		MCU_GPIO0_9

- Tells us:
  - ..
(**MCU**_PADCONFIG9)
  - Configuration reg addr = 0x04084024

Config Register Offset

- Given the config register address,
  we actually need..
  - The pin’s config register offset is bottom 13 bits of the address:

```
#define K3_PINMUX(offset, value, mux_mode)  (((offset) & 0x1fff)) ((value) | (mux_mode))
```

  - So offset for address 0x04084024
    ..

```
&pinctrl_mcu {
    mcu_gpio0_led_btn_default: mcu_gpio0_led_btn_default {
        pinmux = <K3_PINMUX(0x0024, PIN_OUTPUT, MUX_MODE_7)>; /* (B3) GPIO7*/
    };
};
```

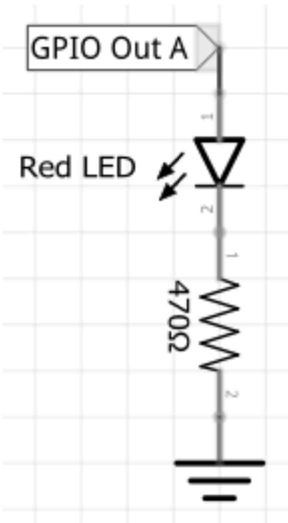
Configuration Confession

- This investigation is (likely?)correct;
  however, the configuration is not
  yet working for MCU GPIO!
  - A work-around is to
    execute the GPIO command in Linux to set configuration:
    gpioset gpiochip0 9=1
  - This causes Linux to setup the configuration
    in the way we need it for the R5.

Example
Blinky

Wire up

- Let’s have R5 flash a custom wired LED
  - Wire this circuit



Process: Build

- Build on host

  ```
  cmake -S . -B build -DBOARD=beagley_ai/j722s/mcu_r5f0_0

  cd build
  make
  cd ..
  ```

- Copy to NFS

  ```
  mkdir -p ~/cmpt433/public/r5/

  cp build/zephyr/zephyr.elf ~/cmpt433/public/r5/zephyr_mcu.elf
  ```

Process: Install

- Copy firmware from NFS to firmware folder

  sudo cp /mnt/remote/r5/zephyr_mcu.elf /lib/firmware/

- Start code

  echo zephyr_mcu.elf | sudo tee /sys/class/remoteproc/remoteproc2/firmware

  echo start | sudo tee /sys/class/remoteproc/remoteproc2/state

- Work-around for GPIO Config

  gpioset gpiochip0 9=1

Summary

- Use R5 to meet hard real-time deadlines for RT tasks
- Process
  - Develop on host
  - Compile on host, install on Target
- GPIO for R5
  - Use of device trees
  - Finding pin info:
    - Use circuit diagram to find pin.
    - Use BeagleY-AI pinout for ball number.
    - Use AM67a Datasheet for register info.
    - Configure in device tree.