EECS192 Lecture 2
Jan. 26, 2021

• Checkpoint 1 (Fri Jan 29): Hello World/LED Blink/Timing
• Checkpoint 2 (Fri Feb 5): driving motor and steering
• Project proposal (due 2/9 before class)
  – Strategy
  – Hardware
  – Block Diagram/Software Model
• LED/Port Information
• PWM for RC servo
• Memory Model- stack, and heap
High resolution timing using built-in 64 bit counter

```c
uint64_t task_counter_value1, task_counter_value2;
double runtime, starttime;
tick_start = xTaskGetTickCount(); // slow ~ 1 ms
timer_get_counter_value(TIMER_GROUP_0, TIMER_0,
    &task_counter_value1); // answer stored in variable

/* code to be timed here */

/* */
timer_get_counter_value(TIMER_GROUP_0, TIMER_0,
    &task_counter_value2); //
starttime = ((double)task_counter_value1/TIMER_SCALE);
runtime = ((double)task_counter_value2/TIMER_SCALE);
snprintf(log, sizeof(log), "Code took %lf seconds \n\r", runtime-starttime);

log_add(log); // Add to log queue
```
CP2- PWM for driving steering servo and ESC

Write code which performs the following sequence of functions:

- **C2.1:** Start wheels turning, and ramp up to full speed in 5 seconds and down to zero speed in another 5 seconds.
- **C2.2:** Set steering angle approximately half full-left and hold for 5 seconds. (For example, if full steering range is +20 degrees, set steering angle to +10 degrees.)
- **C2.3:** Set steering angle straight and hold for 5 seconds.
- **C2.4:** Set steering angle approximately half full-right, and hold for 5 seconds.
- **C2.5:** Set steering angle back to approximately straight.
- **C2.6:** Show steering changing and wheels turning at the same time
- **C2.7:** Report Data RAM and Instruction RAM usage. How much of each is left? [pio run –v in terminal window]
- **C2.8:** All members must fill out the checkpoint survey before the checkoff close. Completion is individually graded
• Checkpoint 1 (Fri Jan 29): Hello World/LED Blink/Timing
• Checkpoint 2 (Fri Feb 5): driving motor and steering
• Project proposal (due 2/9 before class)
  – Strategy
  – Hardware
  – Block Diagram/Software Model
• LED/Port Information
• PWM for RC servo
• Memory Model- stack, and heap
Project Proposal (due on bcourses 2/9 5 pm)

Overall Strategy
Hardware Design
  IO Connections
  Velocity Sensor Mounting
  Camera Height and Angle
Software Structure
Overall Strategy

Example track - Cory 3rd floor

[Diagram of a race track with annotations for start line, steps, crossing, and finish line]

[Graph showing data with labeled axes and points indicating start line, steps, crossing, and finish line]
Project Proposal: Input/Output

Line camera: 128 pixels, 200 Hz

PWM for ESC

PWM for steering servo

1.0 ms
1.5 ms
2.0 ms

20 ms

https://www.sparkfun.com/tutorials/283

Encoder velocity sensor

Other options? Gyro sensor?
ESP32-WROOM GPIO Connections

See ESP32-WROOM data sheet:
## ESP32-WROOM Module Connections

<table>
<thead>
<tr>
<th>Name</th>
<th>No.</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>1</td>
<td>P</td>
<td>Ground</td>
</tr>
<tr>
<td>3V3</td>
<td>2</td>
<td>P</td>
<td>Power supply</td>
</tr>
<tr>
<td>EN</td>
<td>3</td>
<td>I</td>
<td>Module-enable signal. Active high.</td>
</tr>
<tr>
<td>SENSOR_VP</td>
<td>4</td>
<td>I</td>
<td>GPIO36, ADC1_CH0, RTC_GPIO0</td>
</tr>
<tr>
<td>SENSOR_VN</td>
<td>5</td>
<td>I</td>
<td>GPIO39, ADC1_CH3, RTC_GPIO3</td>
</tr>
<tr>
<td>IO34</td>
<td>6</td>
<td>I</td>
<td>GPIO34, ADC1_CH6, RTC_GPIO4</td>
</tr>
<tr>
<td>IO35</td>
<td>7</td>
<td>I</td>
<td>GPIO35, ADC1_CH7, RTC_GPIO5</td>
</tr>
<tr>
<td>IO32</td>
<td>8</td>
<td>I/O</td>
<td>GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOUCH9, RTC_GPIO9</td>
</tr>
<tr>
<td>IO33</td>
<td>9</td>
<td>I/O</td>
<td>GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOUCH8, RTC_GPIO8</td>
</tr>
</tbody>
</table>

**Legend:**

- **P** = power
- **I** = input only
- **I/O** = either
## ESP32-WROOM Module Connections

### Huzzah32

<table>
<thead>
<tr>
<th>Name</th>
<th>No.</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP1-11</td>
<td>IO25</td>
<td>10</td>
<td>I/O, GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0</td>
</tr>
<tr>
<td>JP1-12</td>
<td>IO26</td>
<td>11</td>
<td>I/O, GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1</td>
</tr>
<tr>
<td>JP3-6</td>
<td>IO27</td>
<td>12</td>
<td>I/O, GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV</td>
</tr>
<tr>
<td>JP3-10</td>
<td>IO14</td>
<td>13</td>
<td>I/O, GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2</td>
</tr>
<tr>
<td>JP3-5</td>
<td>IO12</td>
<td>14</td>
<td>I/O, GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3</td>
</tr>
<tr>
<td>JP3-4</td>
<td>GND</td>
<td>15</td>
<td>P, Ground</td>
</tr>
<tr>
<td>JP3-4</td>
<td>IO13</td>
<td>16</td>
<td>I/O, GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER</td>
</tr>
<tr>
<td></td>
<td>SHD/SD2*</td>
<td>17</td>
<td>I/O, GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1_RXD</td>
</tr>
<tr>
<td></td>
<td>SWP/SD3*</td>
<td>18</td>
<td>I/O, GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1_TXD</td>
</tr>
<tr>
<td></td>
<td>SCS/CMD*</td>
<td>19</td>
<td>I/O, GPIO11, SD_CMD, SPICS0, HS1_CMD, U1_RTS</td>
</tr>
<tr>
<td></td>
<td>SCK/CLK*</td>
<td>20</td>
<td>I/O, GPIO6, SD_CLK, SPICLK, HS1_CLK, U1_CTS</td>
</tr>
<tr>
<td></td>
<td>SDO/SD0*</td>
<td>21</td>
<td>I/O, GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2_RTS</td>
</tr>
<tr>
<td></td>
<td>SDI/SD1*</td>
<td>22</td>
<td>I/O, GPIO8, SD_DATA1, SPID, HS1_DATA1, U2_CTS</td>
</tr>
</tbody>
</table>

### SPI Flash

- P = power
- I = input only
- I/O = either
## ESP32-WROOM Module Connections

### Huzzah32

<table>
<thead>
<tr>
<th>IO</th>
<th>Pin</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO15</td>
<td>23</td>
<td>I/O</td>
<td>GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3</td>
</tr>
<tr>
<td>IO2</td>
<td>24</td>
<td>I/O</td>
<td>GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0</td>
</tr>
<tr>
<td>IO0</td>
<td>25</td>
<td>I/O</td>
<td>GPIO0, ADC2_CI11, TOUCH11, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK</td>
</tr>
<tr>
<td>IO4</td>
<td>26</td>
<td>I/O</td>
<td>GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER</td>
</tr>
<tr>
<td>IO16</td>
<td>27</td>
<td>I/O</td>
<td>GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT</td>
</tr>
<tr>
<td>IO17</td>
<td>28</td>
<td>I/O</td>
<td>GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180</td>
</tr>
<tr>
<td>IO5</td>
<td>29</td>
<td>I/O</td>
<td>GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK</td>
</tr>
<tr>
<td>IO18</td>
<td>30</td>
<td>I/O</td>
<td>GPIO18, VSPICLK, HS1_DATA7</td>
</tr>
<tr>
<td>IO19</td>
<td>31</td>
<td>I/O</td>
<td>GPIO19, VSPIQ, U0CTS, EMAC_TXD0</td>
</tr>
<tr>
<td>NC</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IO21</td>
<td>33</td>
<td>I/O</td>
<td>GPIO21, VSPICH, EMAC_TX_EN</td>
</tr>
<tr>
<td>RXD0</td>
<td>34</td>
<td>I/O</td>
<td>GPIO3, U0RXD, CLK_OUT2</td>
</tr>
<tr>
<td>TXD0</td>
<td>35</td>
<td>I/O</td>
<td>GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2</td>
</tr>
<tr>
<td>IO22</td>
<td>36</td>
<td>I/O</td>
<td>GPIO22, VSPIWP, U0RTS, EMAC_TXD1</td>
</tr>
<tr>
<td>IO23</td>
<td>37</td>
<td>I/O</td>
<td>GPIO23, VSPID, HS1_STROBE</td>
</tr>
<tr>
<td>GND</td>
<td>38</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>

- **P** = power
- **I** = input only
- **I/O** = either
Velocity sensor mounting (preview - week 4)

3.3 V DC
0 Vdc

100 us response time

SkeletonHuzzah32 SW Block Diagram

Keyboard input

user_task

heartbeat

uart_log_task

LED

timerEvt_task

UART

Control_task

log_queue

wifi_log_task

sendto UDP socket

Possible starting point for proposal: missing inputs, outputs, Steering, velocity, etc

main() start tasks and suspend

Note conventions- data flow left to right
Mastering the FreeRTOS™ Real Time Kernel

26. Execution pattern highlighting task prioritization and pre-emption in a hypothetical application in which each task has been assigned a unique priority.
Project Proposal: multithread example (rough outline)

Planning/

Process sensor

Calculate control

Calculate state

Main() idle

Other lower priority processes, user input, monitoring, logging

Real Time Application Design Tutorial
https://freertos.org/tutorial/index.html
How to communicate between tasks?

Shared global variables:
```c
double x, xold, v;
```

Task 1 (sensor processing)
```c
xold = x;
x = readSensor();
v = (x - xold) / T;
```

Task 2 (control)
```c
y = kp * x + kd * v;
SetOutput(y);
```

What problems are there with this approach to sharing variables?
Queue in FreeRTOS

A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.
Queue in FreeRTOS

Task A

```c
int x;
x = 20;
```

Task B

```c
int y;
```

The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).

Task B has removed one item, leaving only the second value written by Task A remaining in the queue. This is the value Task B would receive next if it read from the queue again. The queue now has four empty spaces remaining.

Figure 31. An example sequence of writes to, and reads from a queue
EECS192 Lecture 2
Jan. 26, 2021

• Checkpoint 1 (Fri Jan 29): Hello World/LED Blink/Timing
• Checkpoint 2 (Fri Feb 5): driving motor and steering
• Project proposal (due 2/9 before class)
  – Strategy
  – Hardware
  – Block Diagram/Software Model
• LED/Port Information
• PWM for RC servo
• Memory Model- stack, and heap
# 1.3 ESD handling ratings

Table 3. ESD handling ratings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Min.</th>
<th>Max.</th>
<th>Unit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{HBM}$</td>
<td>Electrostatic discharge voltage, human body model</td>
<td>$-2000$</td>
<td>$+2000$</td>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>$V_{CDM}$</td>
<td>Electrostatic discharge voltage, charged-device model</td>
<td>$-500$</td>
<td>$+500$</td>
<td>V</td>
<td>2</td>
</tr>
<tr>
<td>$I_{LAT}$</td>
<td>Latch-up current at ambient temperature of 105 °C</td>
<td>$-100$</td>
<td>$+100$</td>
<td>mA</td>
<td>3</td>
</tr>
</tbody>
</table>

2. Determined according to JEDEC Standard JESD22-C101, *Field-Induced Charged-Device Model Test Method for Electrostatic-Discharge-Withinstand Thresholds of Microelectronic Components*.
3. Determined according to JEDEC Standard JESD78, *IC Latch-Up Test*.
Connecting LED & CPU Port Information

https://www.electronicshub.org/light-emitting-diode-basics/
### LED & CPU Port Information - typical

#### Table 13: DC Characteristics (3.3 V, 25 °C)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{IN} )</td>
<td>Pin capacitance</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>pF</td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>High-level input voltage</td>
<td>0.75×VDD (^1)</td>
<td>-</td>
<td>VDD (^1)+0.3</td>
<td>V</td>
</tr>
<tr>
<td>( V_{IL} )</td>
<td>Low-level input voltage</td>
<td>-0.3</td>
<td>-</td>
<td>0.25×VDD (^1)</td>
<td>V</td>
</tr>
<tr>
<td>( I_{IH} )</td>
<td>High-level input current</td>
<td>-</td>
<td>-</td>
<td>50</td>
<td>nA</td>
</tr>
<tr>
<td>( I_{IL} )</td>
<td>Low-level input current</td>
<td>-</td>
<td>-</td>
<td>50</td>
<td>nA</td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>High-level output voltage</td>
<td>0.8×VDD (^1)</td>
<td>-</td>
<td>-</td>
<td>V</td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Low-level output voltage</td>
<td>-</td>
<td>-</td>
<td>0.1×VDD (^1)</td>
<td>V</td>
</tr>
<tr>
<td>( I_{OH} )</td>
<td>High-level source current (VDD (^1) = 3.3 V, ( V_{OH} ) = 2.64 V, output drive strength set to the maximum)</td>
<td>VDD3P3_CPU power domain (^1,2)</td>
<td>-</td>
<td>40</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td>VDD3P3_RTC power domain (^1,2)</td>
<td>-</td>
<td>40</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VDD_SDIO power domain (^1,3)</td>
<td>-</td>
<td>20</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>( I_{OL} )</td>
<td>Low-level sink current (VDD (^1) = 3.3 V, ( V_{OL} ) = 0.495 V, output drive strength set to the maximum)</td>
<td>-</td>
<td>-</td>
<td>28</td>
<td>mA</td>
</tr>
<tr>
<td>( R_{PU} )</td>
<td>Resistance of internal pull-up resistor</td>
<td>-</td>
<td>45</td>
<td>-</td>
<td>kΩ</td>
</tr>
<tr>
<td>( R_{PD} )</td>
<td>Resistance of internal pull-down resistor</td>
<td>-</td>
<td>45</td>
<td>-</td>
<td>kΩ</td>
</tr>
<tr>
<td>( V_{IL,n_{RST}} )</td>
<td>Low-level input voltage of CHIP_PU to power off the chip</td>
<td>-</td>
<td>-</td>
<td>0.6</td>
<td>V</td>
</tr>
</tbody>
</table>

---

**LATCHUP!**
• Checkpoint 1 (Fri Jan 29): Hello World/LED Blink/Timing
• Checkpoint 2 (Fri Feb 5): driving motor and steering
• Project proposal (due 2/9 before class)
  – Strategy
  – Hardware
  – Block Diagram/Software Model
• LED/Port Information
• PWM for RC servo
• Memory Model- stack, and heap
Pulse Width Modulation

https://github.com/espressif/esp-idf/tree/release/v4.2/examples/peripherals/mcpwm/mcpwm_servo_control

Also see
~/home/.platformio/packages/framework-espidf/examples/peripherals/mcpwm

https://www.instructables.com/id/PANTILT-Camera-With-ESP32/
Motor Control Pulse Width Modulator (MCPWM) (Ch 17)
- UTEA: the PWM timer is counting up and its value is equal to register A.
- UTEB: the PWM timer is counting up and its value is equal to register B.
Setting up mcpwm
(see https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/mcpwm.html#structmcpwm__config__t)

4. Setting of the timer frequency and initial duty within mcpwm_config_t structure.
https://github.com/espressif/esp-idf/blob/release/v4.2/components/driver/include/driver/mcpwm.h

typedef struct {
    uint32_t frequency;  /* Set frequency of MCPWM in Hz*/
    float cmpr_a; /* Set % duty cycle for operator a (MCPWMXA) */
    float cmpr_b; /* Set % duty cycle for operator b (MCPWMXB) */
    mcpwm_duty_type_t duty_mode; /* Set type of duty cycle*/
    /* Set type of MCPWM counter*/
    mcpwm_counter_type_t counter_mode;
} mcpwm_config_t;

5. Call mcpwm_init() with the above parameters to make the configuration effective.
```c
#include "driver/mcpwm.h"
#include "soc/mcpwm_periph.h"

#define SERVO_MIN_PULSEWIDTH 1000  // Minimum pulse width in microsecond
#define SERVO_MAX_PULSEWIDTH 2000  // Maximum pulse width in microsecond
#define SERVO_MAX_DEGREE 90       // Maximum angle which servo can rotate

static void mcpwm_example_gpio_initialize(void)
{
    printf("initializing mcpwm servo control gpio......\n");
    mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, 18);
    // Set GPIO 18 as PWM0A, to which servo is connected
}

/* @brief Use this function to calculate pulse width per degree rotation
 * @param degree_of_rotation the angle to which servo has to rotate
 * @return - calculated pulse width */

static uint32_t servo_per_degree_init(uint32_t degree_of_rotation)
{
    uint32_t cal_pulsewidth = 0;
    cal_pulsewidth = SERVO_MIN_PULSEWIDTH +
                     (SERVO_MAX_PULSEWIDTH - SERVO_MIN_PULSEWIDTH) *
                     (degree_of_rotation) / SERVO_MAX_DEGREE;
    return cal_pulsewidth;
}
```
void mcpwm_example_servo_control(void *arg)
{
    uint32_t angle, count;
    mcpwm_example_gpio_initialize();
    printf("Configuring Initial Parameters of mcpwm......\n");
    mcpwm_config_t pwm_config;
    pwm_config.frequency = 50; //frequency = 50Hz, i.e. time period= 20ms
    pwm_config.cmpr_a = 0;    //duty cycle of PWMxA = 0
    pwm_config.cmpr_b = 0;    //duty cycle of PWMxb = 0
    pwm_config.counter_mode = MCPWM_UP_COUNTER;
    pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
    //Configure PWM0A & PWM0B with above settings
    mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config );

    while (1)
    {
        for (count = 0; count < SERVO_MAX_DEGREE; count++)
        {
            printf("Angle of rotation: %d\n", count);
            angle = servo_per_degree_init(count);
            printf("pulse width: %dus\n", angle);
            mcpwm_set_duty_in_us(MCPWM_UNIT_0,
                                  MCPWM_TIMER_0, MCPWM_OPR_A, angle);
            vTaskDelay(10);
            //Add delay, since it takes time for servo to rotate,
            //generally 100ms/60degree rotation at 5V
            //also avoid starving idle process
        }
    }
}
Setting PWM duty cycle

mcpwm_set_duty_in_us(mcpwm_unit_t mcpwm_num,
                    mcpwm_timer_t timer_num,
                    mcpwm_generator_t gen,
                    uint32_t duty_in_us);

gen: set the generator(MCPWMXA/MCPWMXB), ‘x’ is operator number selected

/** * @brief Select MCPWM operator */
typedef enum {
    MCPWM_GEN_A = 0, /*Select MCPWMXA, where 'X' is operator number*/
    MCPWM_GEN_B,    /*Select MCPWMXB, where 'X' is operator number*/
    MCPWM_GEN_MAX,  /*Num of generators to each operator of MCPWM*/
} mcpwm_generator_t;
• Checkpoint 1 (Fri Jan 29): Hello World/LED Blink/Timing
• Checkpoint 2 (Fri Feb 5): driving motor and steering
• Project proposal (due 2/9 before class)
  – Strategy
  – Hardware
  – Block Diagram/Software Model
• LED/Port Information
• PWM for RC servo
• Memory Model- stack, and heap
Intro to stack, heap, malloc, free, etc

Memory model

- **stack**
  - Inactive frame
  - Active frame
  - Stack origin

- **heap**
  - Uninitialized data
  - BSS
  - Initialized data
  - Data
  - Text

E.g., local variables in functions


 STACK ORIGIN

 STACK POINTER = 9
FreeRTOS Example Heap Operation

Figure 7. RAM being allocated and freed from the heap_4 array

A. xTaskCreate(); x3
B. vTaskDelete();
C. xQueueCreate();
D. pvPortMalloc();
E. vQueueDelete();
F. vPortFree();

Note: Stacks are specified in words, not bytes. Requesting the stack size to 1K when calling xTaskCreate will get 4K bytes of stack as the word size is 4 bytes.
Some timing critical code may be placed into IRAM to reduce the penalty associated with loading the code from flash. ESP32 reads code and data from flash via a 32 kB cache. In some cases, placing a function into IRAM may reduce delays caused by a cache miss.
Creating RTOS objects using statically allocated RAM has the benefit of providing the application writer with more control:

RTOS objects can be placed at specific memory locations.

The maximum RAM footprint can be determined at link time, rather than run time.

The application writer does not need to concern themselves with graceful handling of memory allocation failures.

It allows the RTOS to be used in applications that simply don’t allow any dynamic memory allocation (although FreeRTOS includes allocation schemes that can overcome most objections).

**Note:** `printf-stdarg.c` from FreeRTOS+TCP drastically decreases stack usage for most tasks.
Extra Slides
### Table 4: Embedded Memory Address Mapping

<table>
<thead>
<tr>
<th>Bus Type</th>
<th>Boundary Address</th>
<th>Size</th>
<th>Target</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>0x3FF8_0000</td>
<td>8 KB</td>
<td>RTC FAST Memory</td>
<td>PRO_CPU Only</td>
</tr>
<tr>
<td></td>
<td>0x3FF8_2000</td>
<td>56 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>0x3FF9_0000</td>
<td>64 KB</td>
<td>Internal ROM 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x3FFA_0000</td>
<td>56 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>0x3FFA_E000</td>
<td>200 KB</td>
<td>Internal SRAM 2</td>
<td>DMA</td>
</tr>
<tr>
<td>Data</td>
<td>0x3FFE_0000</td>
<td>128 KB</td>
<td>Internal SRAM 1</td>
<td>DMA</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4000_0000</td>
<td>32 KB</td>
<td>Internal ROM 0</td>
<td>Remap</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4000_8000</td>
<td>352 KB</td>
<td>Internal ROM 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x4006_0000</td>
<td>64 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4007_0000</td>
<td>64 KB</td>
<td>Internal SRAM 0</td>
<td>Cache</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4008_0000</td>
<td>128 KB</td>
<td>Internal SRAM 0</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400A_0000</td>
<td>64 KB</td>
<td>Internal SRAM 1</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400B_0000</td>
<td>32 KB</td>
<td>Internal SRAM 1</td>
<td>Remap</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400B_8000</td>
<td>32 KB</td>
<td>Internal SRAM 1</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400C_0000</td>
<td>8 KB</td>
<td>RTC FAST Memory</td>
<td>PRO_CPU Only</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x5000_0000</td>
<td>8 KB</td>
<td>RTC SLOW Memory</td>
<td></td>
</tr>
</tbody>
</table>

- Internal SRAM1 128KB
- Internal SRAM0 192KB (64KB used for cache)
- Internal SRAM2 200KB
Table 4: Embedded Memory Address Mapping

<table>
<thead>
<tr>
<th>Bus Type</th>
<th>Boundary Address</th>
<th>Size</th>
<th>Target</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>0x3FF8_0000</td>
<td>8 KB</td>
<td>RTC FAST Memory</td>
<td>PRO_CPU Only</td>
</tr>
<tr>
<td></td>
<td>0x3FF8_2000</td>
<td>56 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x3FF9_0000</td>
<td>64 KB</td>
<td>Internal ROM 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0x3FFA_0000</td>
<td>56 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>0x3FFD_0000</td>
<td>200 KB</td>
<td>Internal SRAM 2</td>
<td>DMA</td>
</tr>
<tr>
<td>Data</td>
<td>0x3FFE_0000</td>
<td>128 KB</td>
<td>Internal SRAM 1</td>
<td>DMA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bus Type</th>
<th>Boundary Address</th>
<th>Size</th>
<th>Target</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>0x4000_0000</td>
<td>32 KB</td>
<td>Internal ROM 0</td>
<td>Remap</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4000_8000</td>
<td>352 KB</td>
<td>Internal ROM 0</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4006_0000</td>
<td>64 KB</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x4007_0000</td>
<td>64 KB</td>
<td>Internal SRAM 0</td>
<td>Cache</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400B_0000</td>
<td>128 KB</td>
<td>Internal SRAM 0</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400A_0000</td>
<td>64 KB</td>
<td>Internal SRAM 1</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400B_8000</td>
<td>32 KB</td>
<td>Internal SRAM 1</td>
<td>Remap</td>
</tr>
<tr>
<td>Instruction</td>
<td>0x400C_0000</td>
<td>8 KB</td>
<td>RTC FAST Memory</td>
<td>PRO_CPU Only</td>
</tr>
<tr>
<td>Data Instruction</td>
<td>0x5000_0000</td>
<td>8 KB</td>
<td>RTC SLOW Memory</td>
<td></td>
</tr>
</tbody>
</table>

C:\Users\ronf\teach\EE192\skeleton-2021>
python c:\Users\ronf\platformio\packages\framework-esp3df\components\esptool_py\esptool\esptool.py --chip esp32 image_info .pio\build\featheresp32\firmware.bin

Entry point: 400814d0
6 segments
Segment 1: len 0x190fc load 0x3f400020 file_offs 0x00000018 [DROM] memory mapped external FLASH
Segment 2: len 0x04dc8 load 0x3fbb0000 file_offs 0x0001911c [BYTE_ACCESSIBLE, DRAM, DMA]
Segment 3: len 0x00404 load 0x40080000 file_offs 0x0001deec [IRAM]
Segment 4: len 0x01d18 load 0x40080404 file_offs 0x0001e2f8 [IRAM]
Segment 5: len 0x75054 load 0x400d0020 file_offs 0x00020018 [IROM]
Segment 6: len 0x13c50 load 0x4008211c file_offs 0x00095074 [IRAM]
Heap memory (in Data RAM)

Used for stack, local variables, global variables malloc() and free()

heap_caps_print_heap_info(MALLOC_CAP_8BIT);

Heap info before starting tasks
Heap summary for capabilities 0x00000006:
  At 0x3ffae6e0 len 6432 free 0 allocated 6300 min_free 0
      largest_free_block 0 alloc_blocks 25 free_blocks 0 total_blocks 25
At 0x3ffbada8 len 152152 free 136724 allocated 15284 min_free 135908
      largest_free_block 135908 alloc_blocks 26 free_blocks 2 total_blocks 28
At 0x3ffe0440 len 15072 free 15036 allocated 0 min_free 15036
      largest_free_block 15036 alloc_blocks 0 free_blocks 1 total_blocks 1
At 0x3ffe4350 len 113840 free 113804 allocated 0 min_free 113804
      largest_free_block 113804 alloc_blocks 0 free_blocks 1 total_blocks 1
Totals:
  free 265564 allocated 21584 min_free 264748 largest_free_block 135908

Double is how many bytes? (8)
double track_data[20000] would overflow
Heap memory after starting tasks

Heap info after starting tasks

User Task started

Heap summary for capabilities 0x00000006:

At 0x3ffae6e0 len 6432 free 0 allocated 6300 min_free 0
  largest_free_block 0 alloc_blocks 25 free_blocks 0 total_blocks 25
At 0x3ffbada8 len 152152 free 75752 allocated 75616 min_free 75368
  largest_free_block 75412 alloc_blocks 184 free_blocks 4 total_blocks 188
At 0x3ffe0440 len 15072 free 15036 allocated 0 min_free 15036
  largest_free_block 15036 alloc_blocks 0 free_blocks 1 total_blocks 1
At 0x3ffe4350 len 113840 free 113804 allocated 0 min_free 113804
  largest_free_block 113804 alloc_blocks 0 free_blocks 1 total_blocks 1

Totals:
  free 204592 allocated 81916 min_free 204208 largest_free_block 113804

70K allocated for tasks