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

CP1- Measuring Timing from ESP32

High resolution timing using built-in 64 bit counter

```
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

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

Project Proposal (due on bcourses 2/9 5 pm)

Overall Strategy

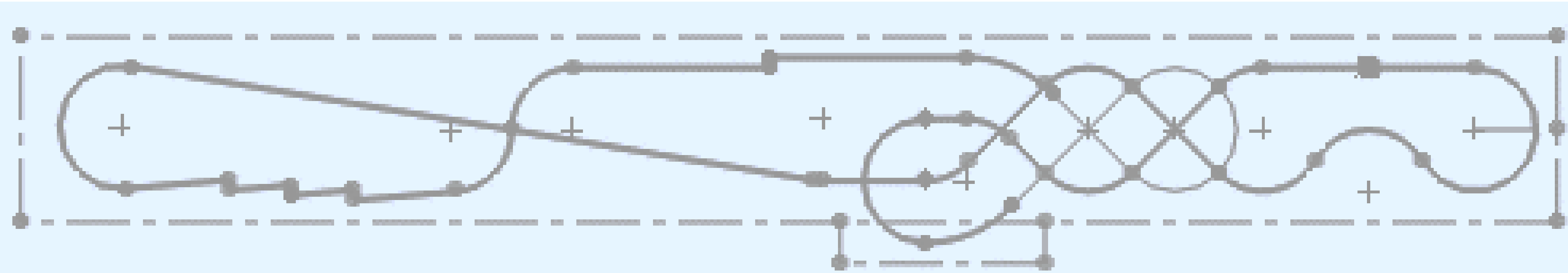
Hardware Design

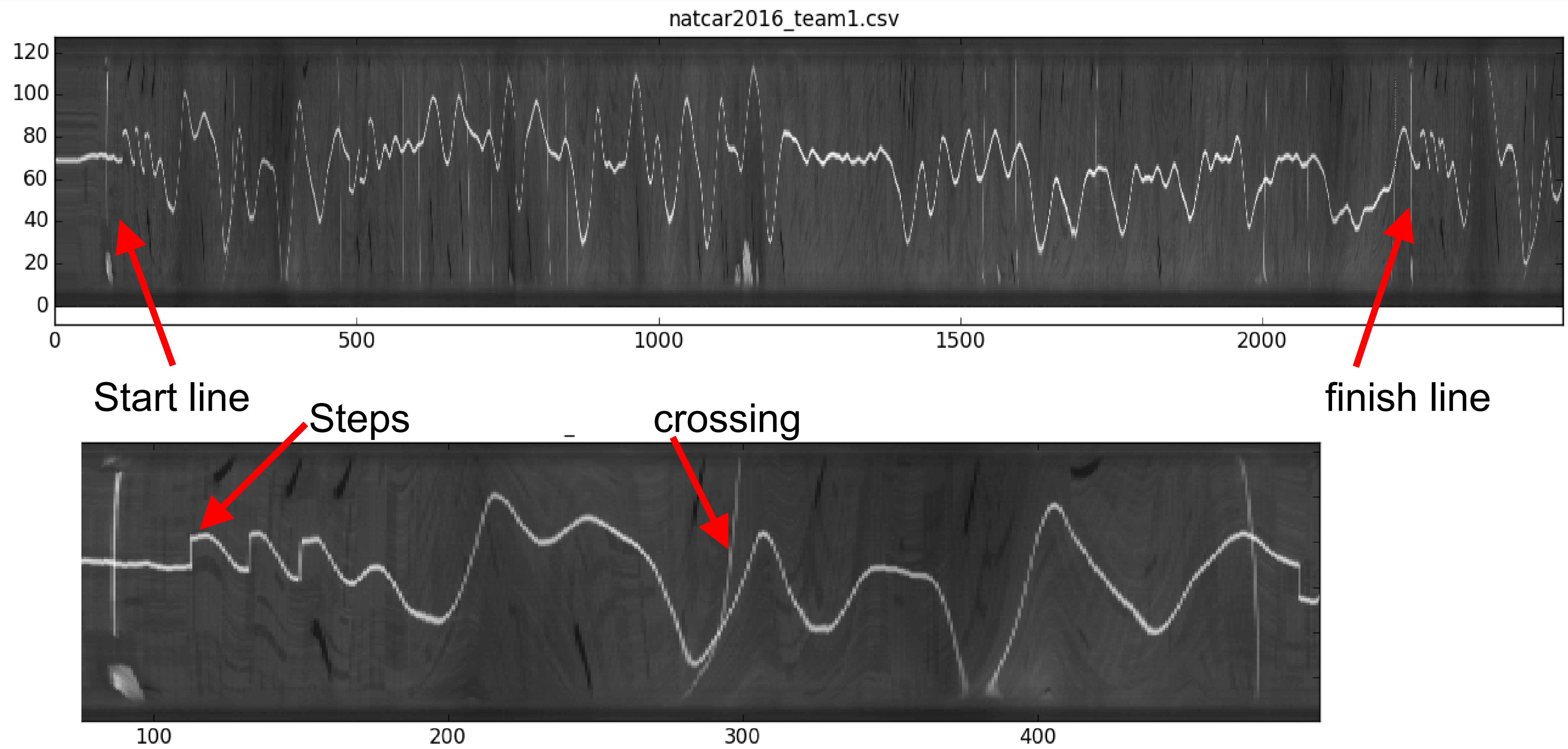
- IO Connections
- Velocity Sensor Mounting
- Camera Height and Angle

Software Structure

Overall Strategy

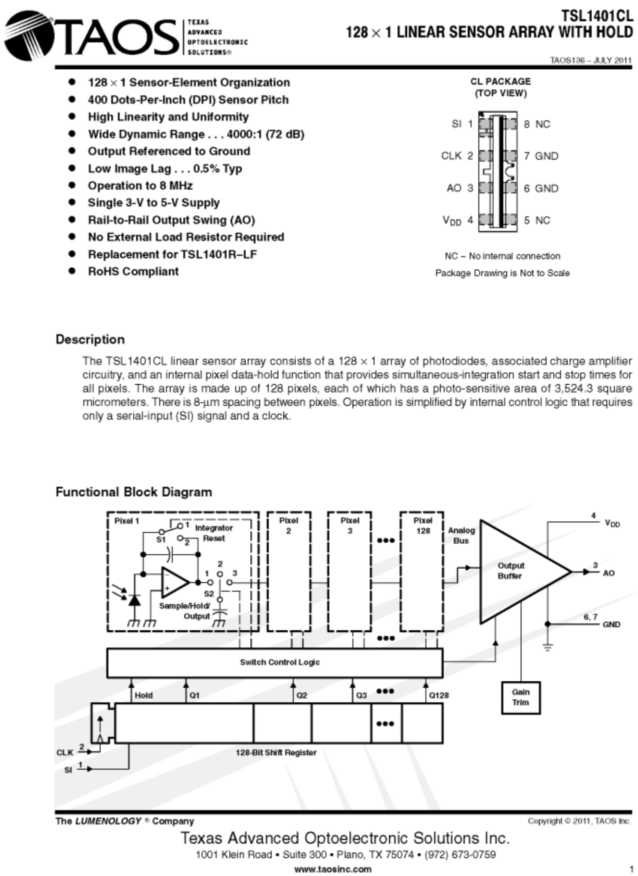
Example track- Cory 3rd floor





Project Proposal: Input/Output

Line camera: 128 pixels, 200 Hz


TSL1401CL
128 × 1 LINEAR SENSOR ARRAY WITH HOLD
TAOS136 – JULY 2011

- 128 × 1 Sensor-Element Organization
- 400 Dots-Per-Inch (DPI) Sensor Pitch
- High Linearity and Uniformity
- Wide Dynamic Range . . . 4000:1 (72 dB)
- Output Referenced to Ground
- Low Image Lag . . . 0.5% Typ
- Operation to 8 MHz
- Single 3-V to 5-V Supply
- Rail-to-Rail Output Swing (AO)
- No External Load Resistor Required
- Replacement for TSL1401R–LF
- RoHS Compliant

CL PACKAGE
(TOP VIEW)

SI 1 — 8 NC
CLK 2 — 7 GND
AO 3 — 6 GND
V_{DD} 4 — 5 NC

NC – No internal connection
Package Drawing is Not to Scale

Description

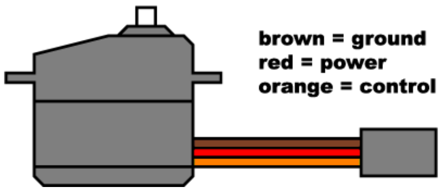
The TSL1401CL linear sensor array consists of a 128 × 1 array of photodiodes, associated charge amplifier circuitry, and an internal pixel data-hold function that provides simultaneous-integration start and stop times for all pixels. The array is made up of 128 pixels, each of which has a photo-sensitive area of 3,524.3 square micrometers. There is 8-μm spacing between pixels. Operation is simplified by internal control logic that requires only a serial-input (SI) signal and a clock.

Functional Block Diagram

The *LUMENOLOGY* ® Company

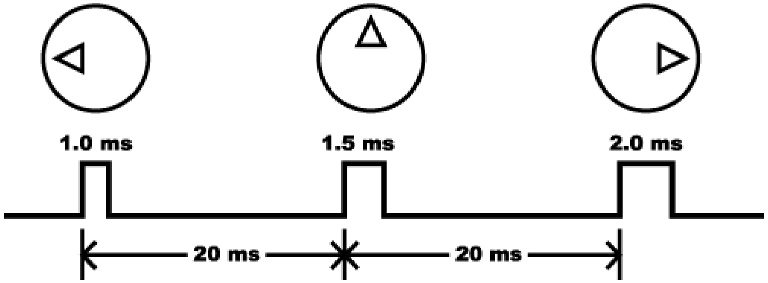
Texas Advanced Optoelectronic Solutions Inc.
1001 Klein Road • Suite 300 • Plano, TX 75074 • (972) 673-0759
www.taosinc.com

1



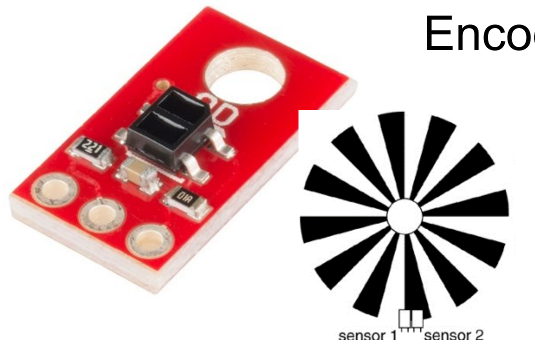
PWM for ESC

PWM for steering servo



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

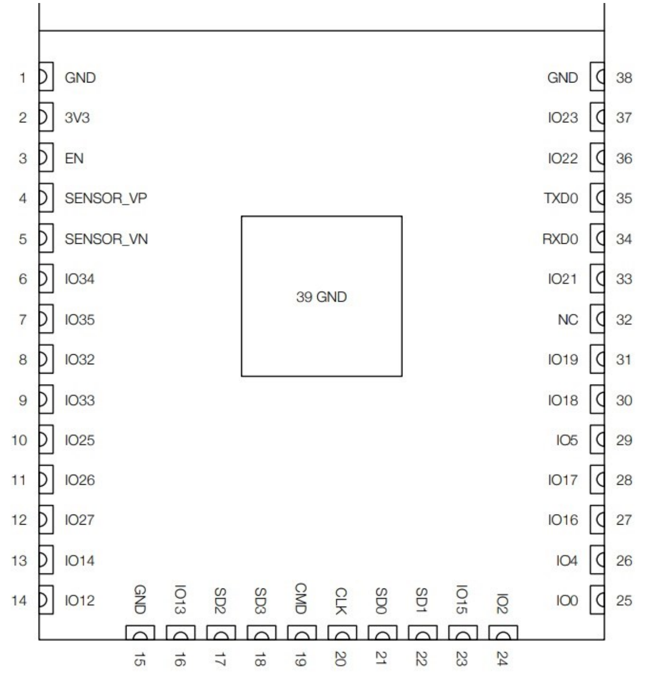
Encoder velocity sensor

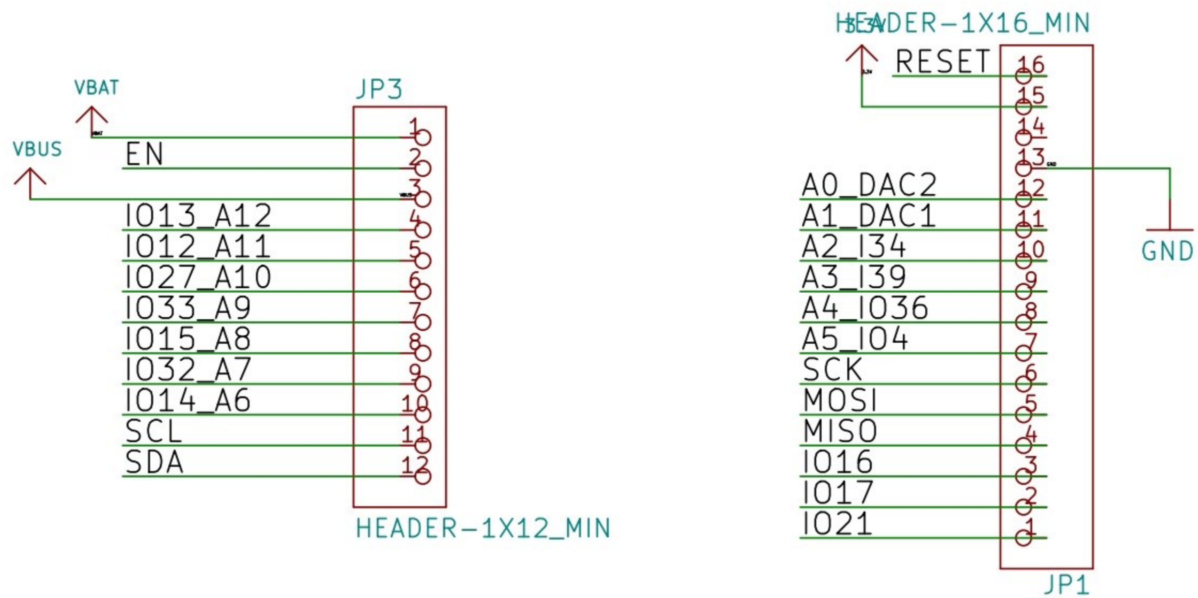


Other options? Gyro sensor?

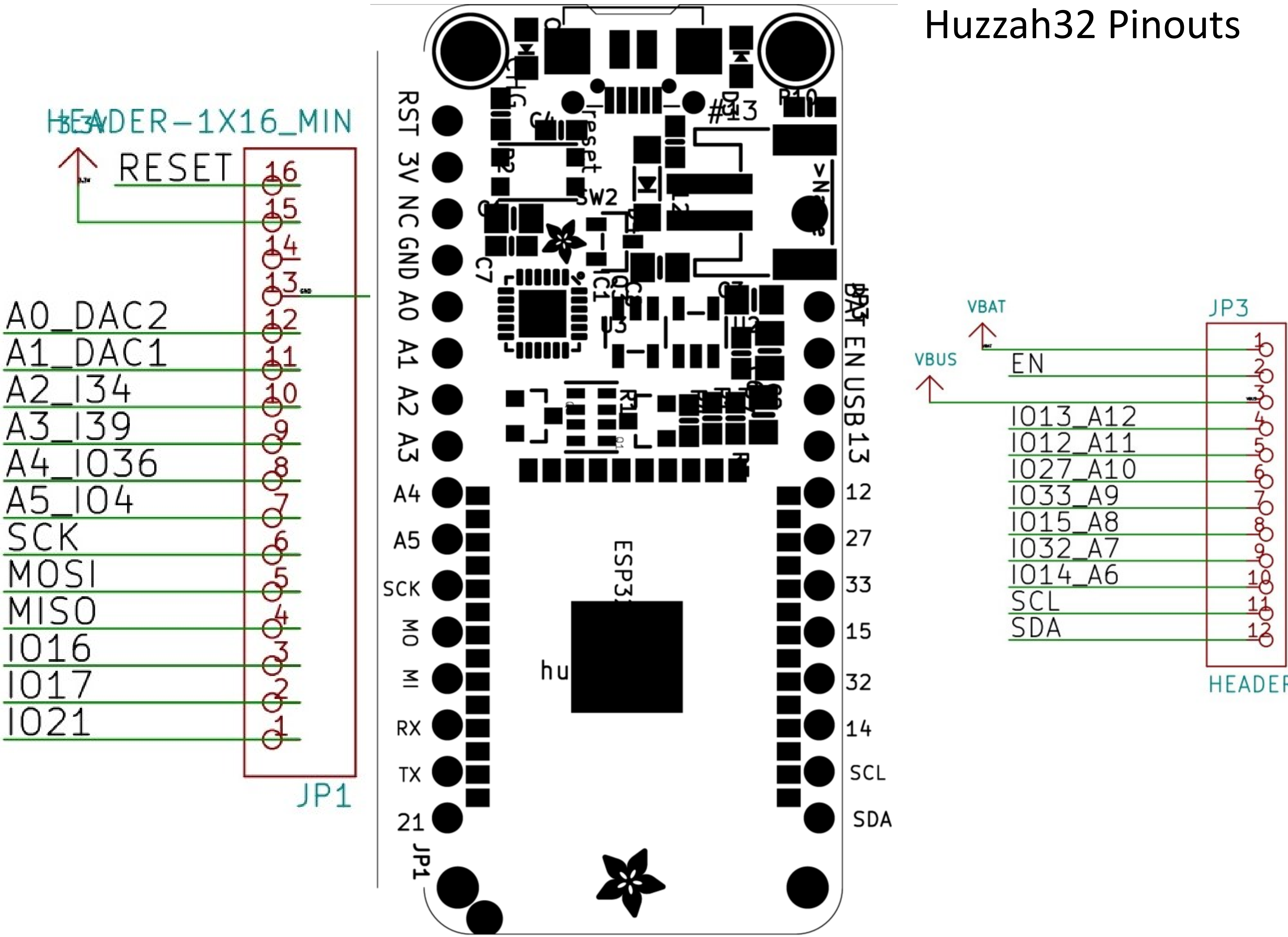
ESP32-WROOM GPIO Connections

See ESP32-WROOM data sheet:
https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf





Huzzah32 Pinouts



ESP32-WROOM Module Connections

Huzzah32

JP1-8
JP1-9
JP1-10
Vref?

JP3-9

JP3-7

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

P=power
I= input only
I/O = either

ESP32-WROOM Module Connections

Huzzah32	Name	No.	Type	Function
JP1-11	IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
JP1-12	IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
JP3-6	IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
JP3-10	IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
JP3-5	IO12	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
	GND	15	P	Ground
JP3-4	IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SPI Flash	SHD/SD2*	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SPI Flash	SWP/SD3*	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SPI Flash	SCS/CMD*	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SPI Flash	SCK/CLK*	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SPI Flash	SDO/SD0*	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SPI Flash	SDI/SD1*	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS

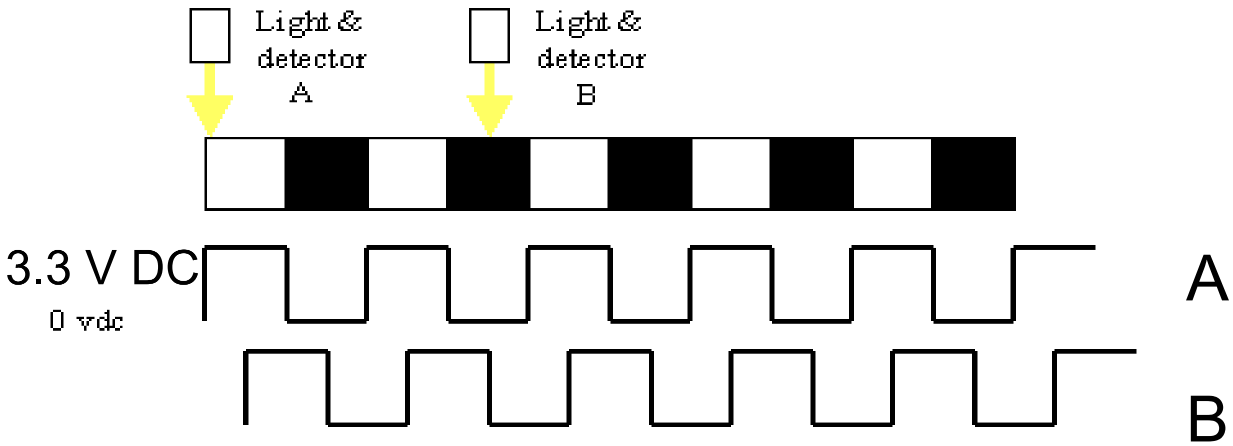
P=power
I= input only
I/O = either

ESP32-WROOM Module Connections

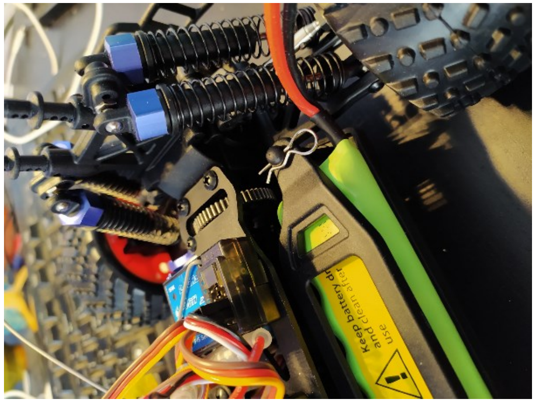
Huzzah32				
JP3-8	IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
RTS?	IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
DTR?	IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
JP1-7	IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
JP1-3	IO16	27	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
JP1-2	IO17	28	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
JP1-6	IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
JP1-5	IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
JP1-4	IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
	NC	32	-	-
JP1-1	IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
UART	RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
	TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
JP3-11	IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
JP3-12	IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
	GND	38	P	Ground

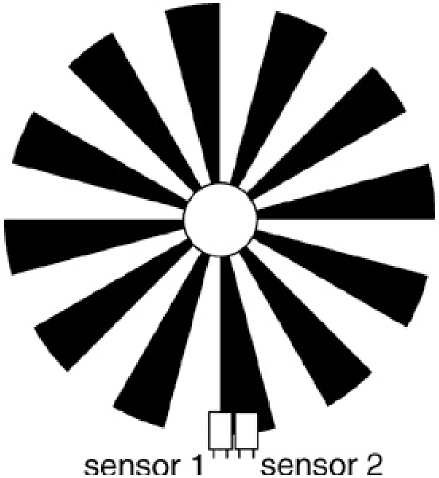
P=power
I= input only
I/O = either

Velocity sensor mounting (preview- week 4)



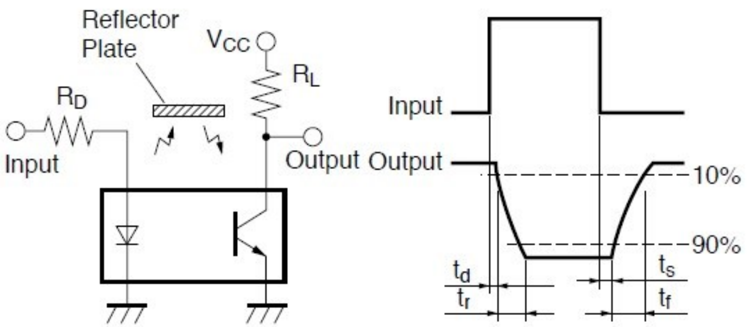
https://www.sinotech.com/wp-content/uploads/quadrature-encoder.gif



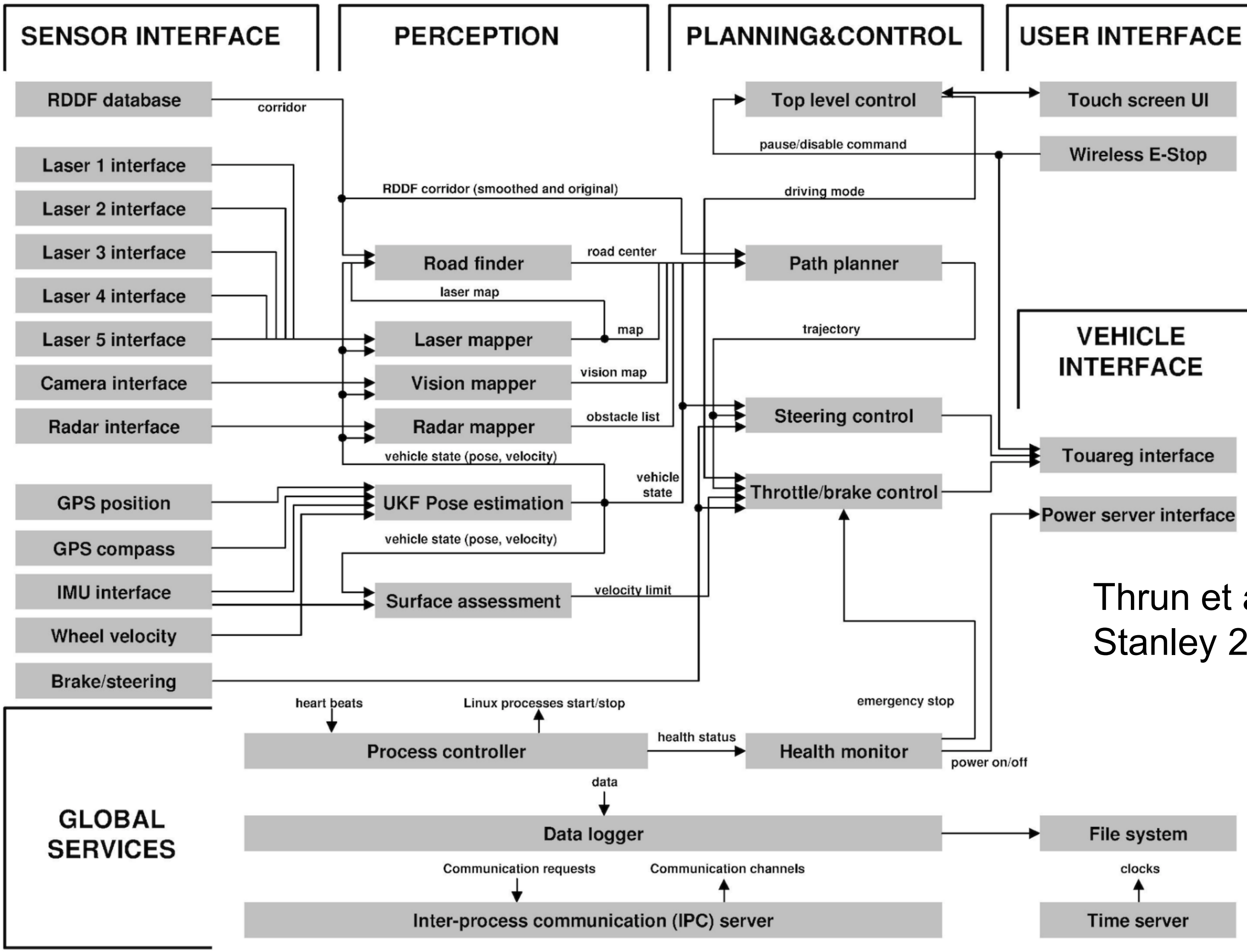


100 us
response time

Fig.9 Test Circuit for Response Time

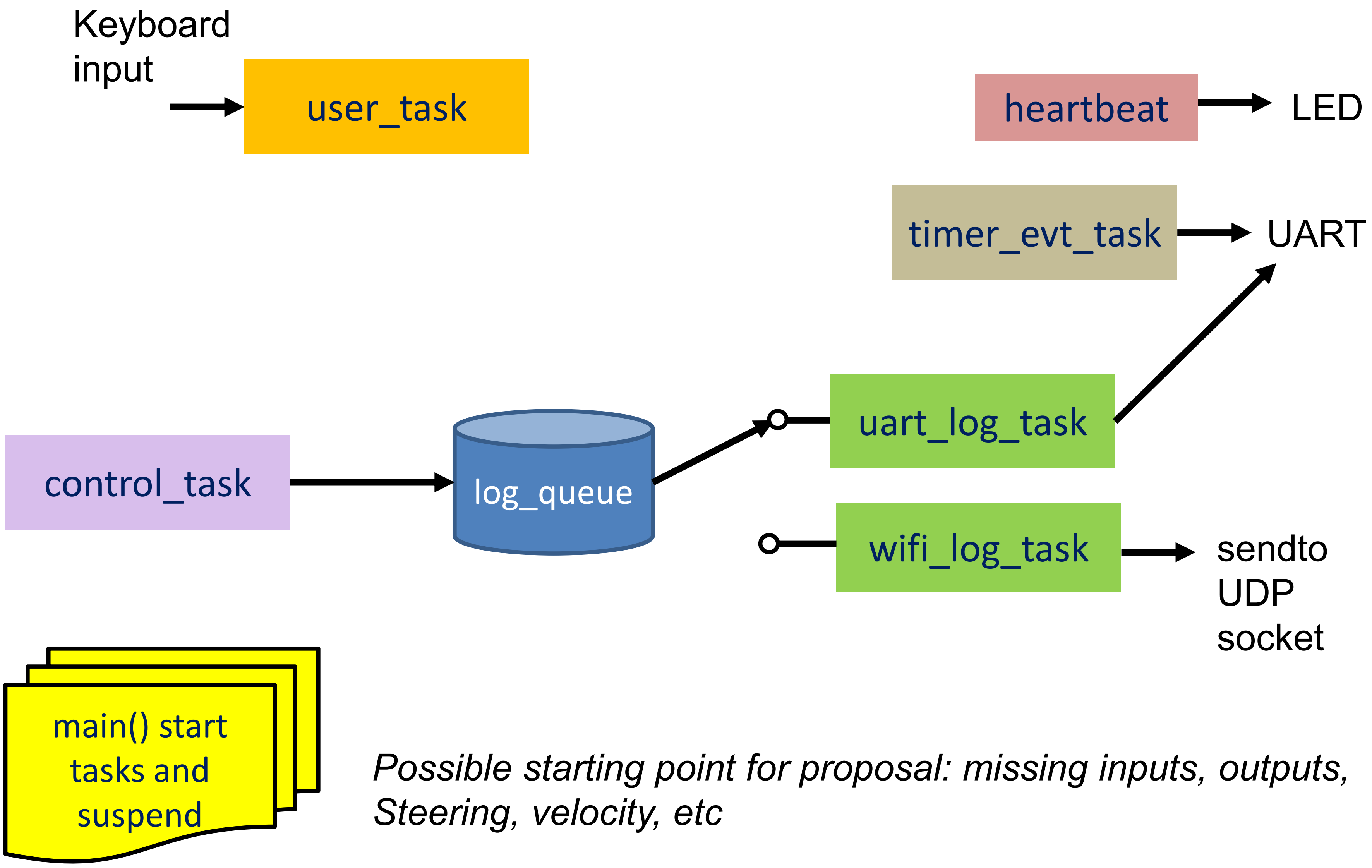


Project Proposal: Block Diagram/Structure



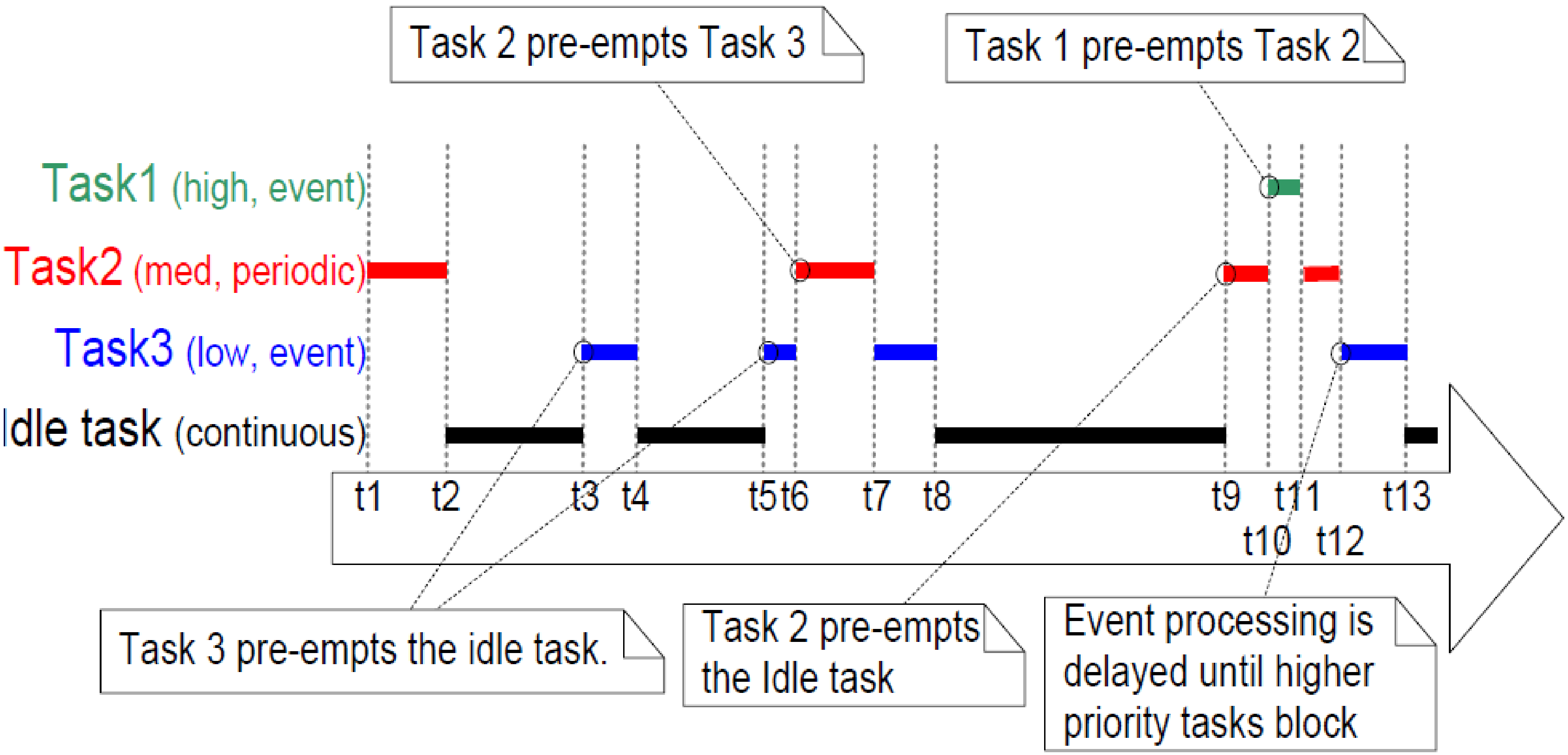
Thrun et al
Stanley 2005

SkeletonHuzzah32 SW Block Diagram



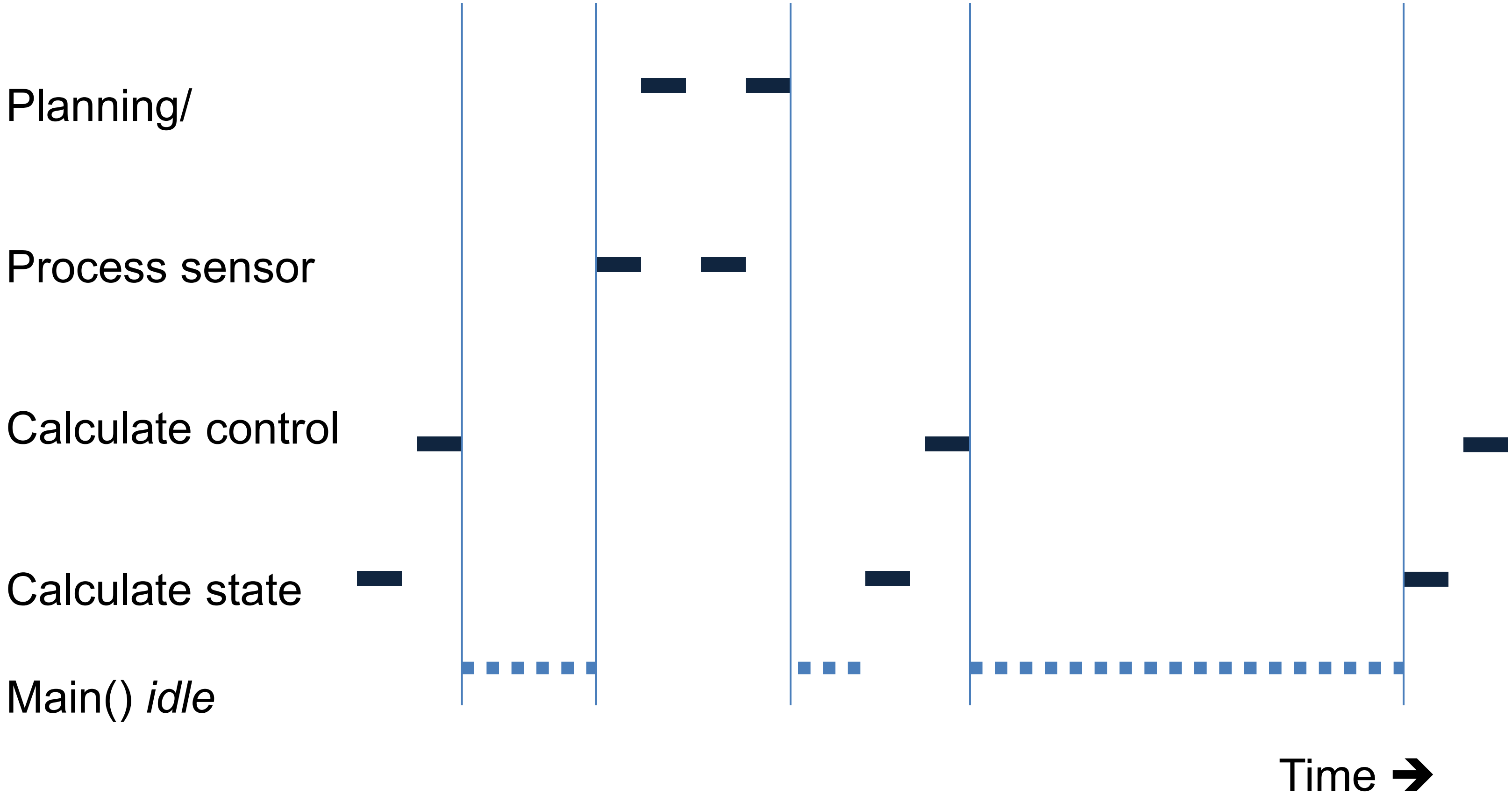
Possible starting point for proposal: missing inputs, outputs, Steering, velocity, etc

Note conventions- data flow left to right



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)



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：

```
double x, xold, v;
```

Task 1 (sensor processing)

```
xold =x;
x=readSensor();
v=(x-xold)/T;
```

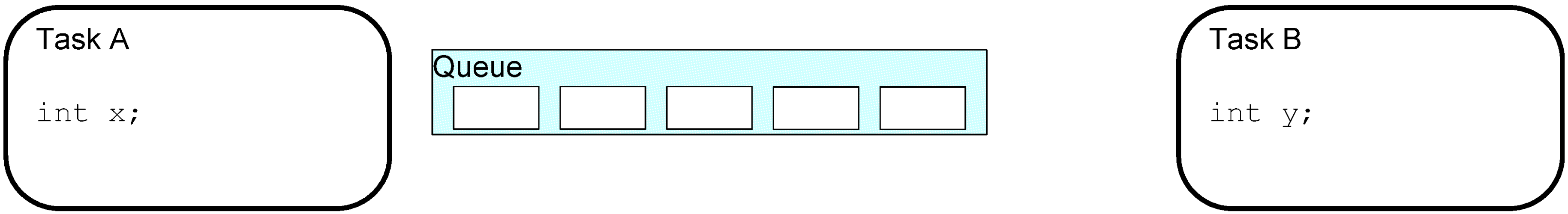
Task 2 (control)

```
y=kp*x+kd*v;
SetOutput(y);
```

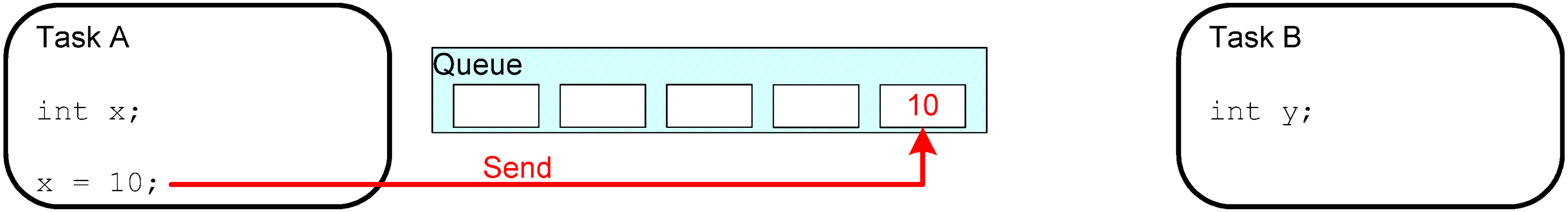
What problems are there with this approach to sharing variables?

Queue in FreeRTOS

161204 Pre-release for FreeRTOS V8.x.x. See http://www.FreeRTOS.org/FreeRTOS-V9.html for information about FreeRTOS V9.x.x. Use http://www.FreeRTOS.org/contact to provide feedback, corrections, and check for updates.



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

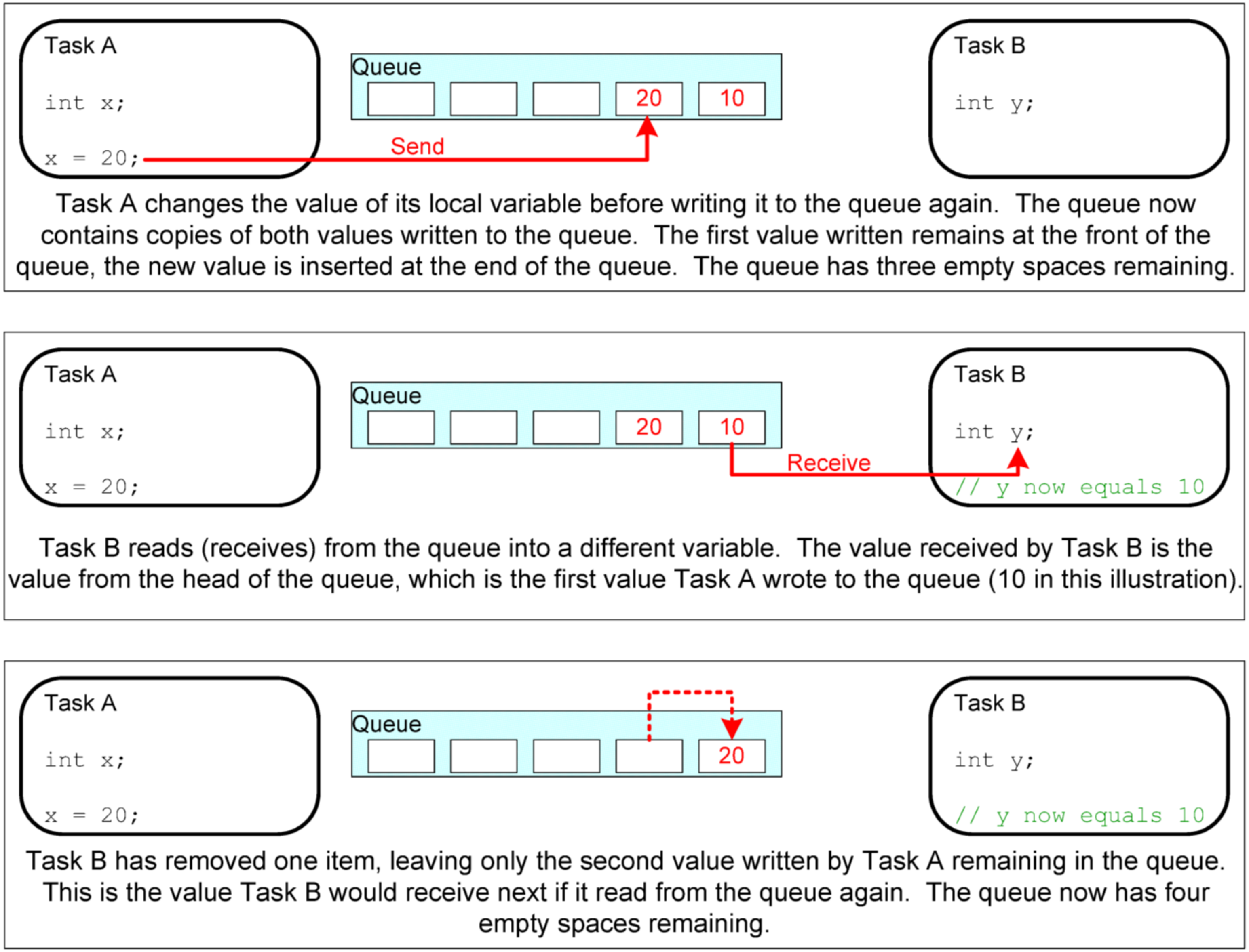


Figure 31. An example sequence of writes to, and reads from a queue

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

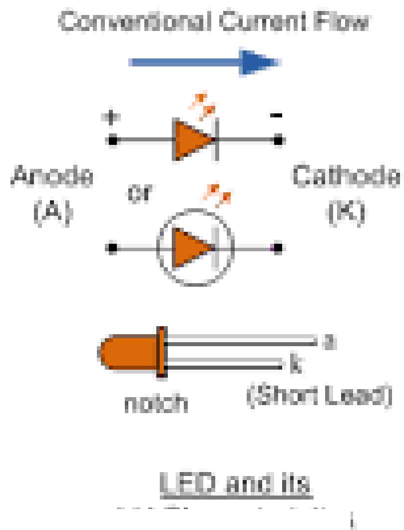
1.3 ESD handling ratings

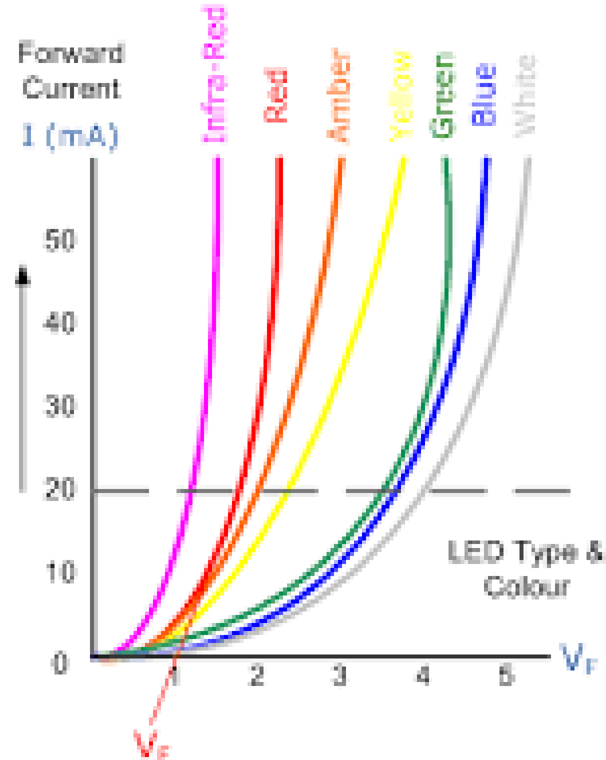
Table 3. ESD handling ratings

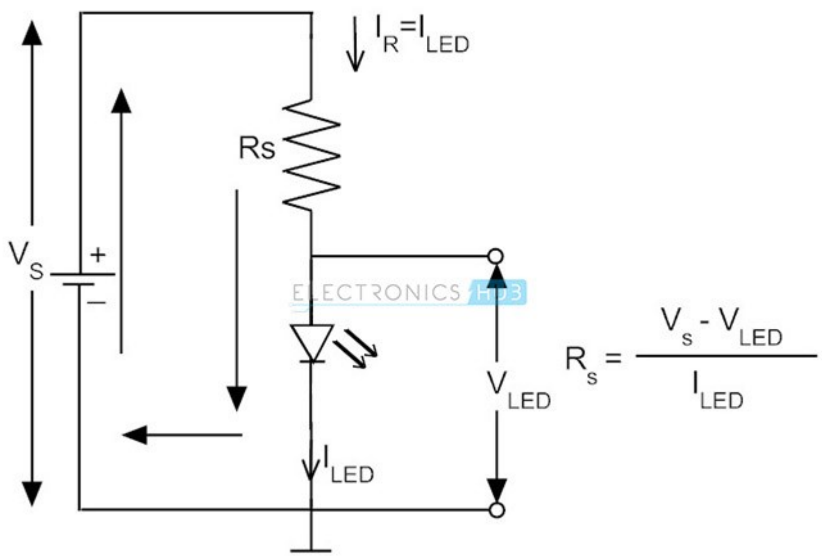
Symbol	Description	Min.	Max.	Unit	Notes
V_{HBM}	Electrostatic discharge voltage, human body model	–2000	+2000	V	1
V_{CDM}	Electrostatic discharge voltage, charged-device model	–500	+500	V	2
I_{LAT}	Latch-up current at ambient temperature of 105 °C	–100	+100	mA	3

1. Determined according to JEDEC Standard JESD22-A114, *Electrostatic Discharge (ESD) Sensitivity Testing Human Body Model (HBM)*.
2. Determined according to JEDEC Standard JESD22-C101, *Field-Induced Charged-Device Model Test Method for Electrostatic-Discharge-Withstand Thresholds of Microelectronic Components*.
3. Determined according to JEDEC Standard JESD78, *IC Latch-Up Test*.

Connecting LED & CPU Port Information







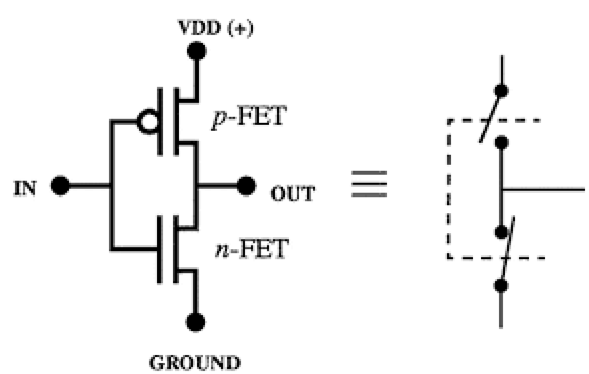
https://www.electronicshub.org/light-emitting-diode-basics/

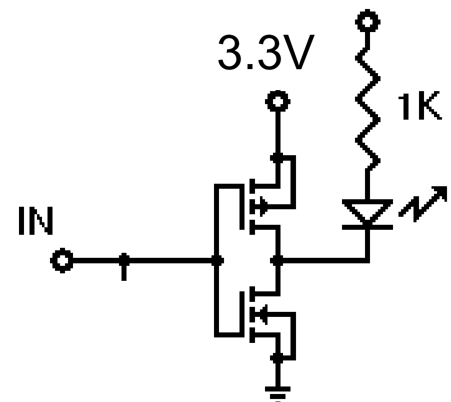
LED & CPU Port Information- typical

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

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

LATCHUP!





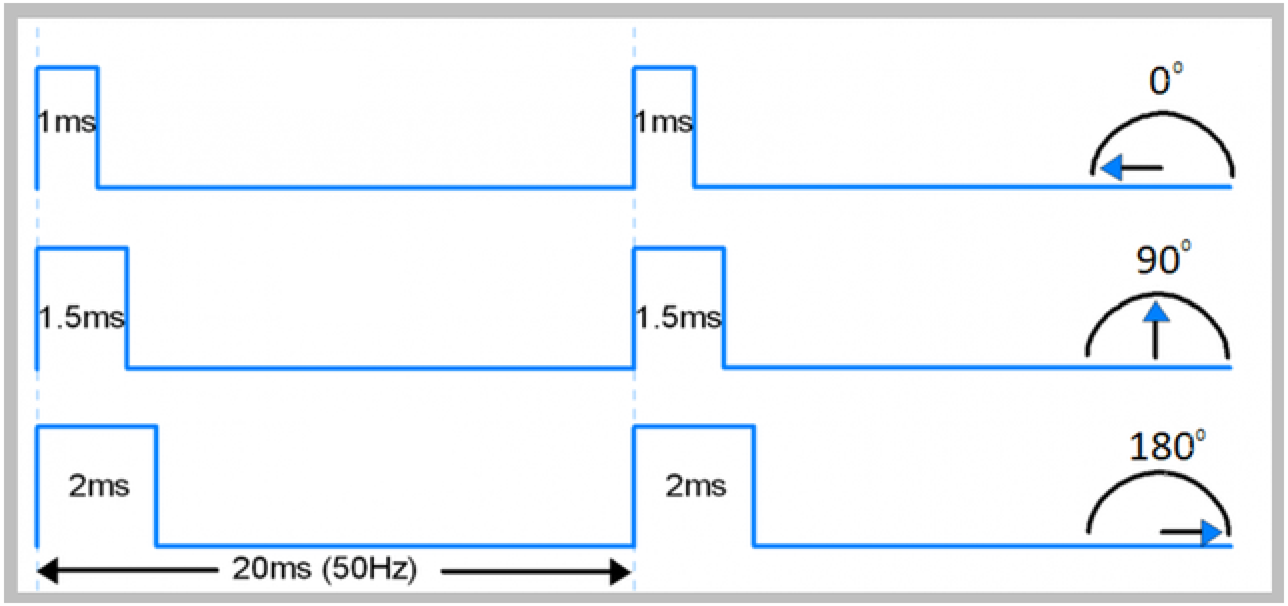
EECS192 Lecture 2
Jan. 26, 2021

- 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

Pulse Width Modulation

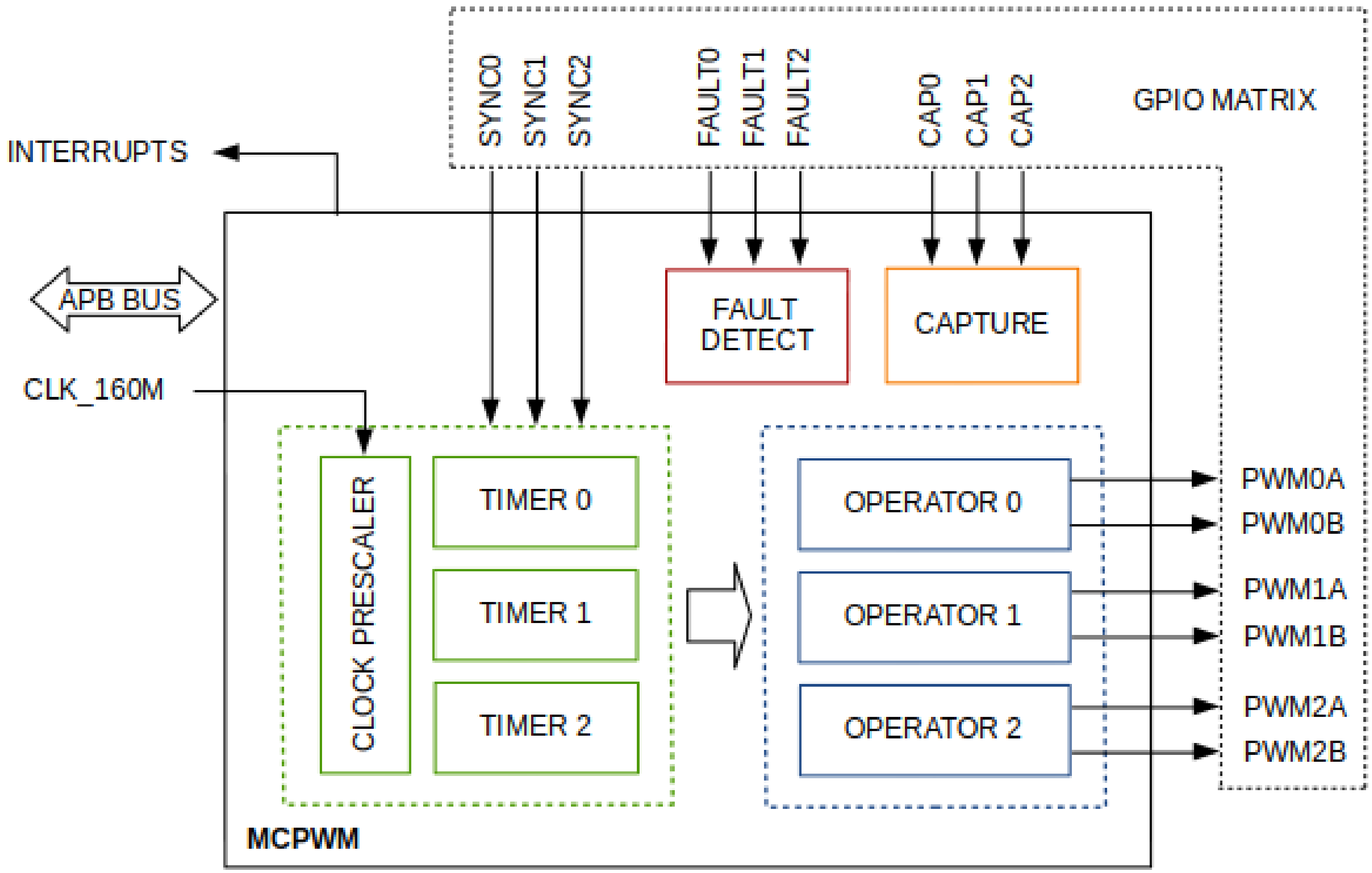
https://github.com/espressif/esp-idf/tree/release/v4.2
/examples/peripherals/mcpwm/mcpwm_servo_control

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)



Motor Control Pulse Width Modulator (MCPWM) (Ch 17)

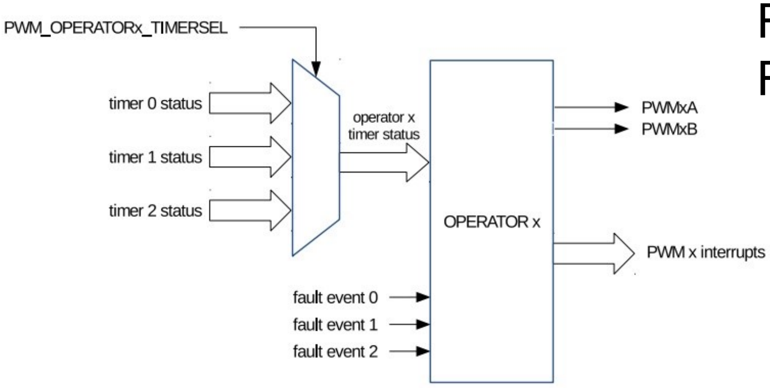


Figure 95: Operator Submodule

Period =7
Reg A =5
Reg B=3

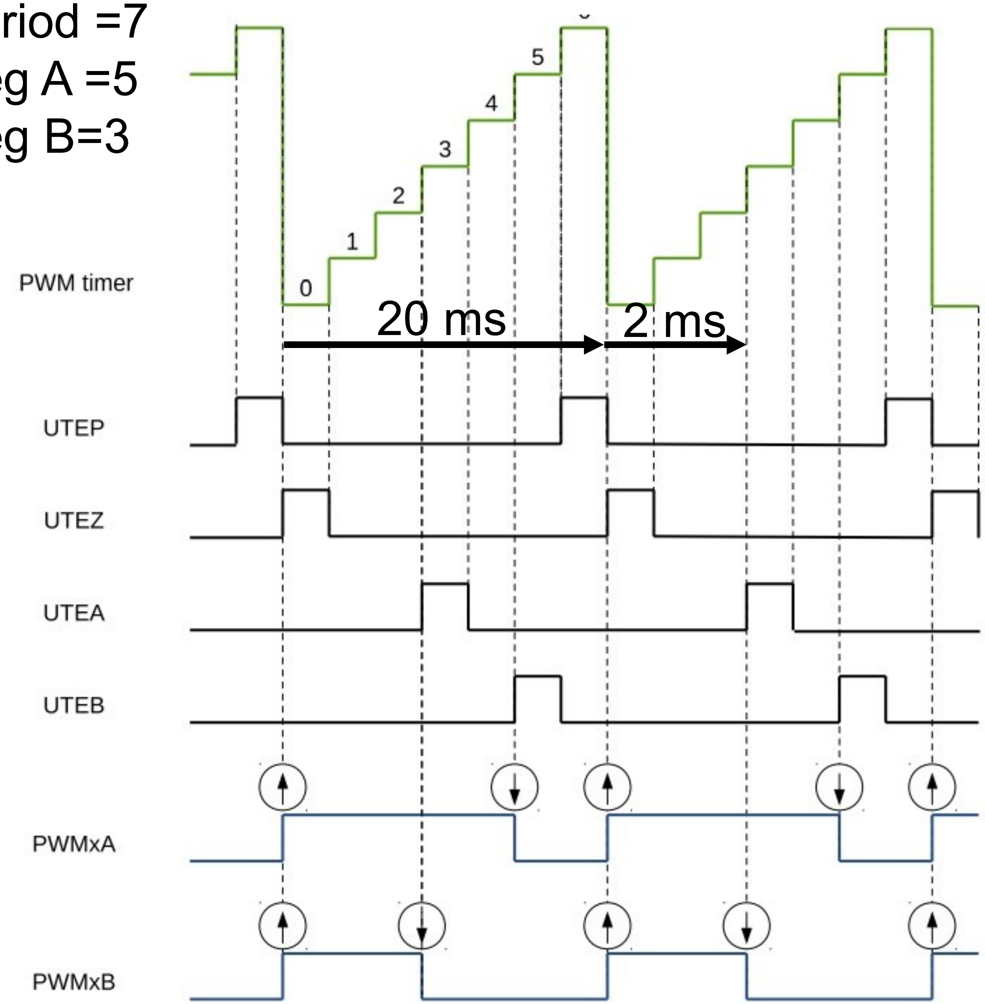


Figure 107: Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWMxA and PWMxB — Active High

Duty cycle 5/7

Duty cycle 3/7

– 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.

mcpwm_servo_control_example.c (1/2)

```
#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;
}
```

mcpwm_servo_control_example.c, (2/2)

```
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

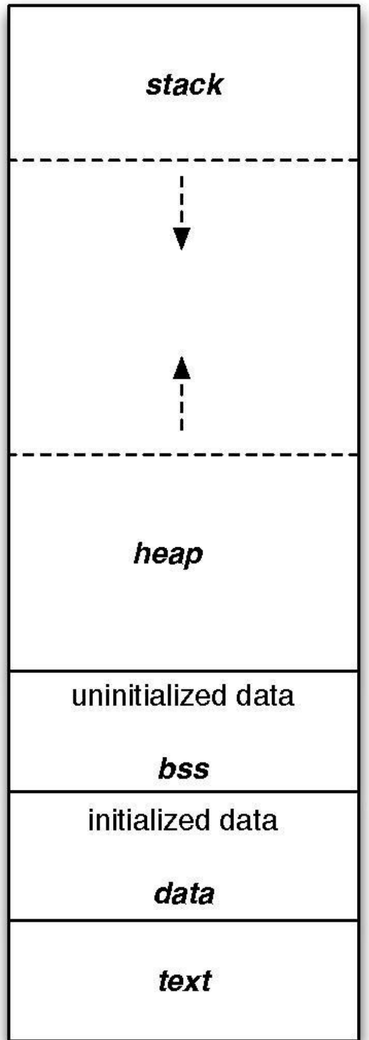
```
/** * @brief Select MCPWM operator */
typedef enum {
   MCPWM_GEN_A = 0,  /*Select MCPWMXA, where 'X' is operator number*/
   MCPWM_GEN_B,      /*Select MCPWMXB, where 'X' is operator number*/
   MCPWM_GEN_MAX,    /*Num of generators to each operator of MCPWM*/
} mcpwm_generator_t;
```

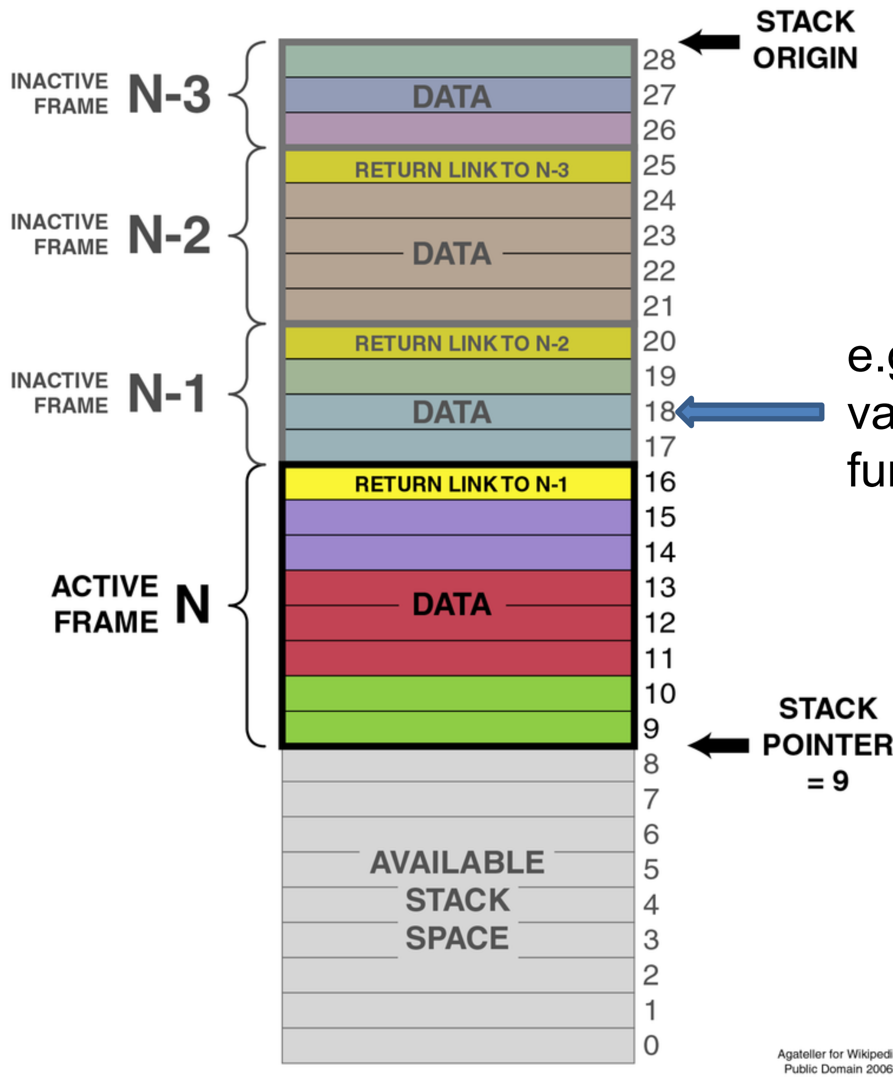
EECS192 Lecture 2
Jan. 26, 2021

- 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

Intro to stack, heap, malloc, free, etc

Memory model

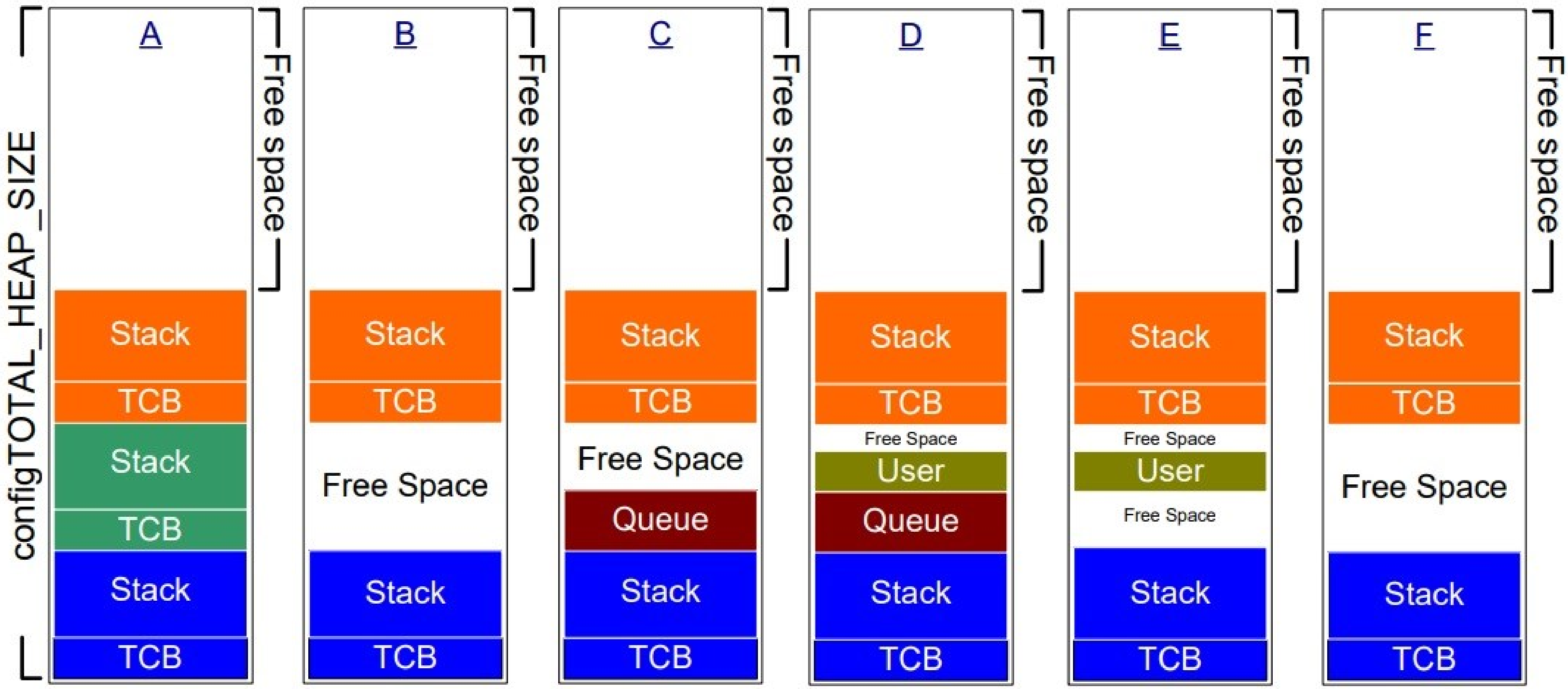




e.g. local variables in functions

From Wikipedia:https://en.wikipedia.org/wiki/Data_segment

FreeRTOS Example Heap Operation



TCB = task control block

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.

From [Mastering the FreeRTOS Real Time Kernel – a Hands On Tutorial Guide]

Huzzah32/ESP32 WROOM memory

Internal SRAM1 128KB
Internal SRAM0 192KB
Internal SRAM2 200KB

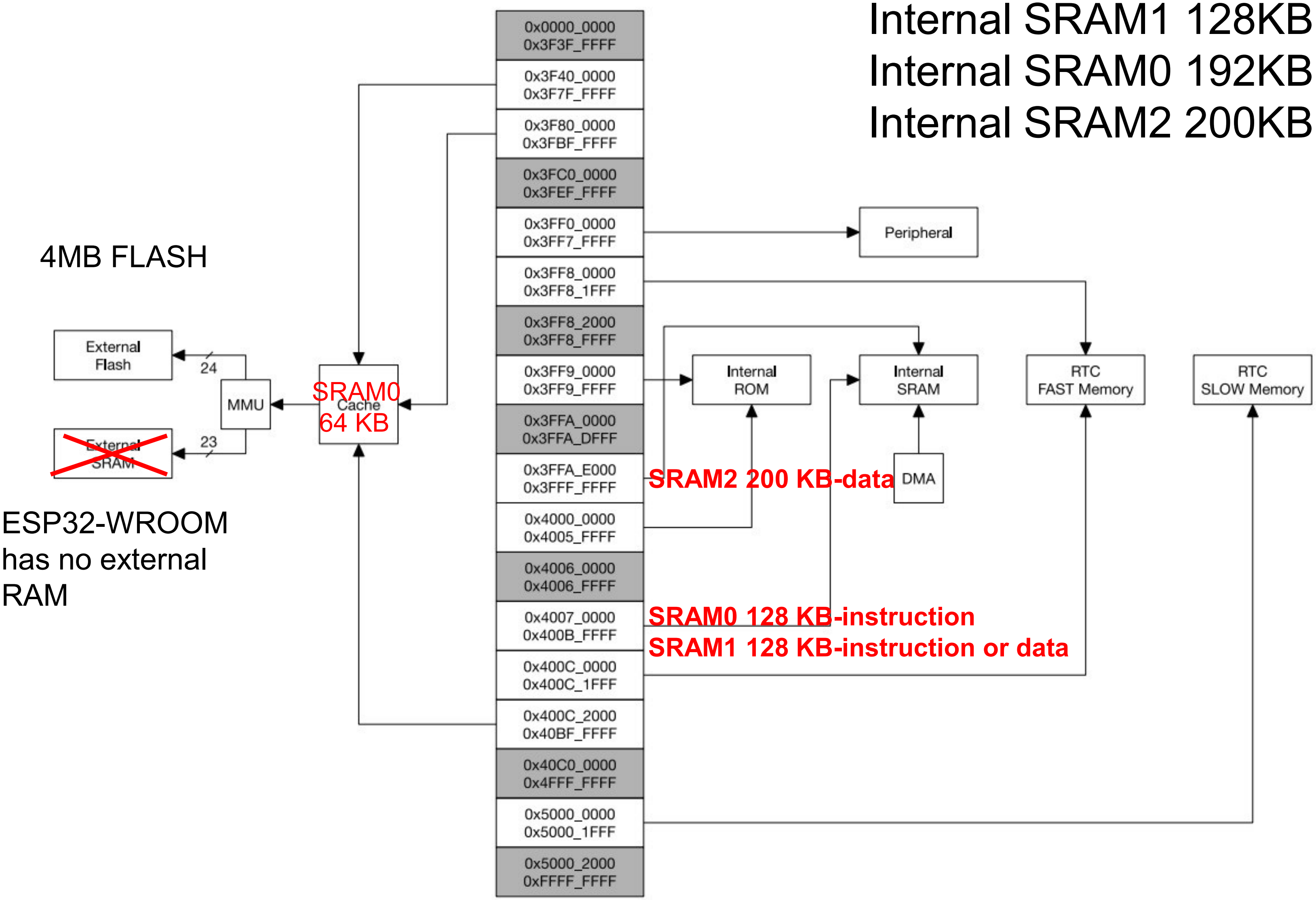


Figure 2: System Address Mapping

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.

Static vs Dynamic Memory Allocation

https://freertos.org/Static_Vs_Dynamic_Memory_Allocation.html

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

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

Internal SRAM1 128KB
Internal SRAM0 192KB (64KB used for cache)
Internal SRAM2 200KB

Table 4: Embedded Memory Address Mapping

Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	Reserved	-
Data	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	-
	0x3FFA_0000	0x3FFA_DFFF	56 KB	Reserved	-
Data	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
Data	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Instruction	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
Instruction	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	-
	0x4006_0000	0x4006_FFFF	64 KB	Reserved	-
Instruction	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
Instruction	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	-
Instruction	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	-
Instruction	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
Instruction	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	-
Instruction	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
Bus Type	Boundary Address		Size	Target	Comment
	Low Address	High Address			
Data Instruction	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	-

load 0x3f400020 len 0x190fc file_offs 0x00000018
[DROM] Data Read Only Memory

load 0x3ffb0000 len 0x04dc8 file_offs 0x0001911c
[BYTE_ACCESSIBLE, DRAM, DMA] Data RAM
~20KB

load 0x40080000 len 0x00404 [IRAM]
load 0x40080404 len 0x01d18 [IRAM]
load 0x4008211c len 0x13c50 f [IRAM]
0x40095d6c (~89 KB)

C:\Users\ronf\teach\EE192\skeleton-2021>
python c:\Users\ronf\.platformio\packages\framework-espidf\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 0x3ffb0000 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