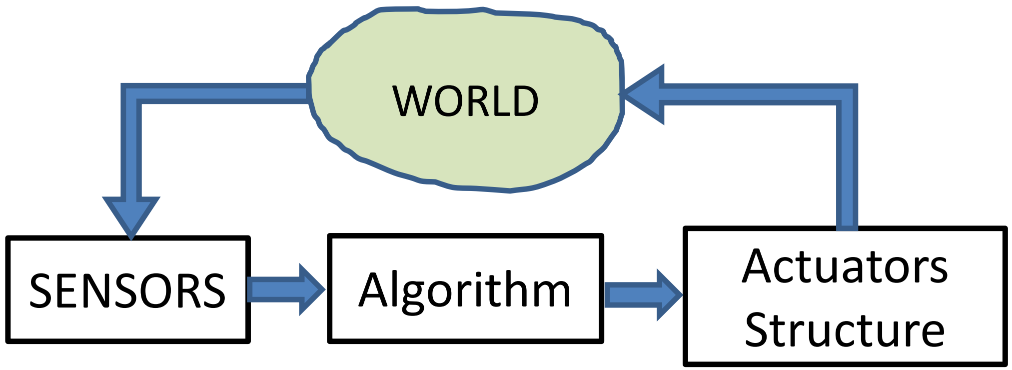
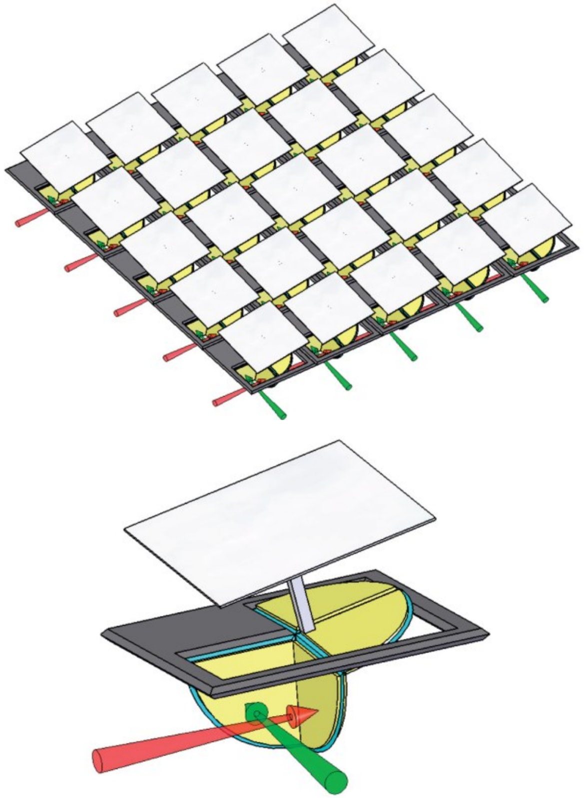
EECS192 Lecture 1
Jan. 19, 2021

- What is Mechatronics?
- Project Description
- Autonomous system example
- Course Organization
- Huzzah32/ESP32 overview
- FreeRTOS tasks and timing

What is Mechatronics?

- Moore's Law for electronics
- Moore's Law for mechanics(?)





Folded mirror array

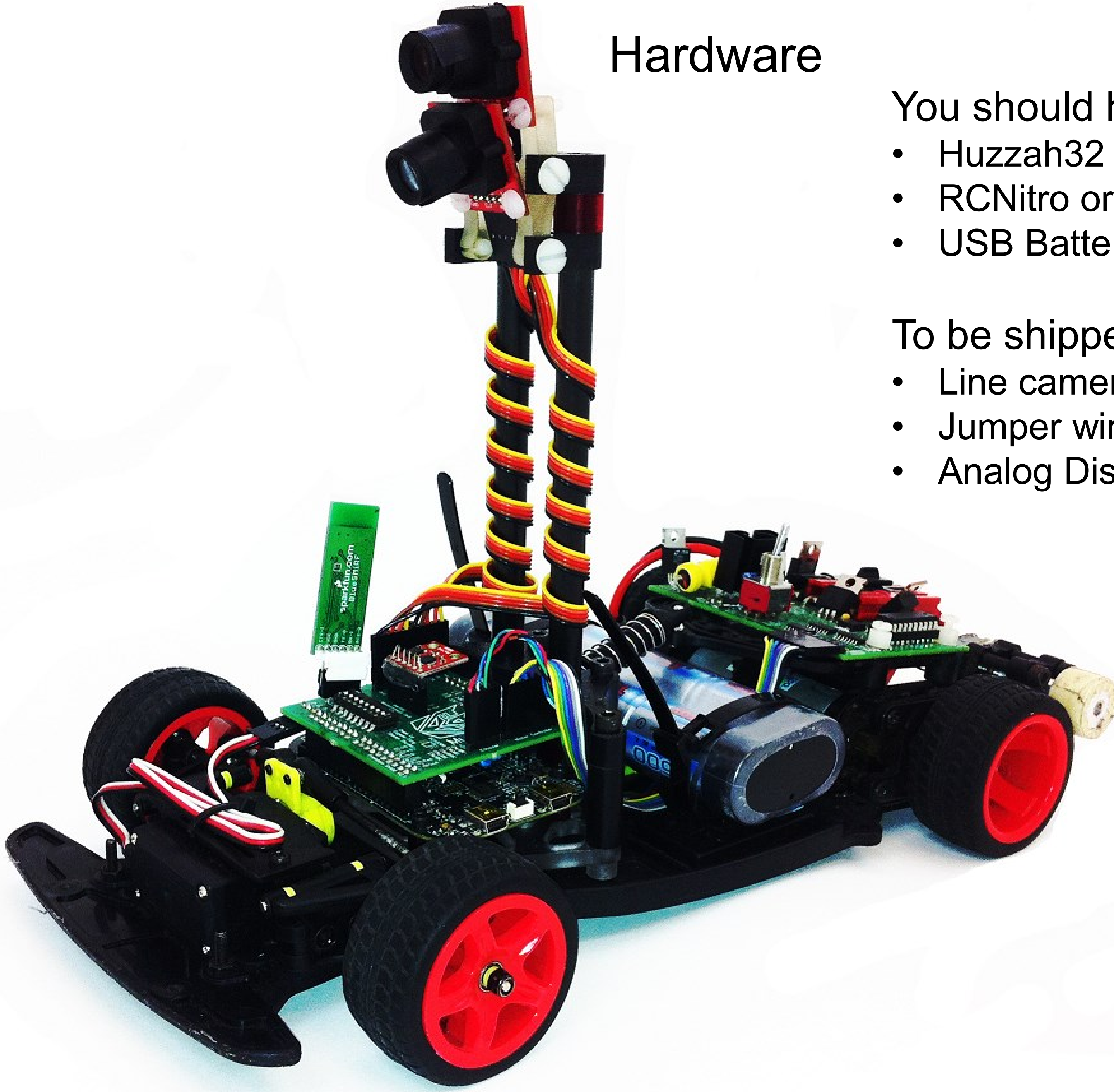
Key Technologies for Mechatronics:

- Signal processing
- Control
- …

Project Description

- Design Autonomous Race Car
- Unknown track (in principle, but home track will be known)
- Follow track without hitting cones.
- Stop at end of track.
- Winning speed: 3.3 m/sec (Spring 2017 Natcar winner)
- (2018 9.7 ft/sec = 3.0 m/sec)
- Learning allowed (though only have 5 minutes total for best run)

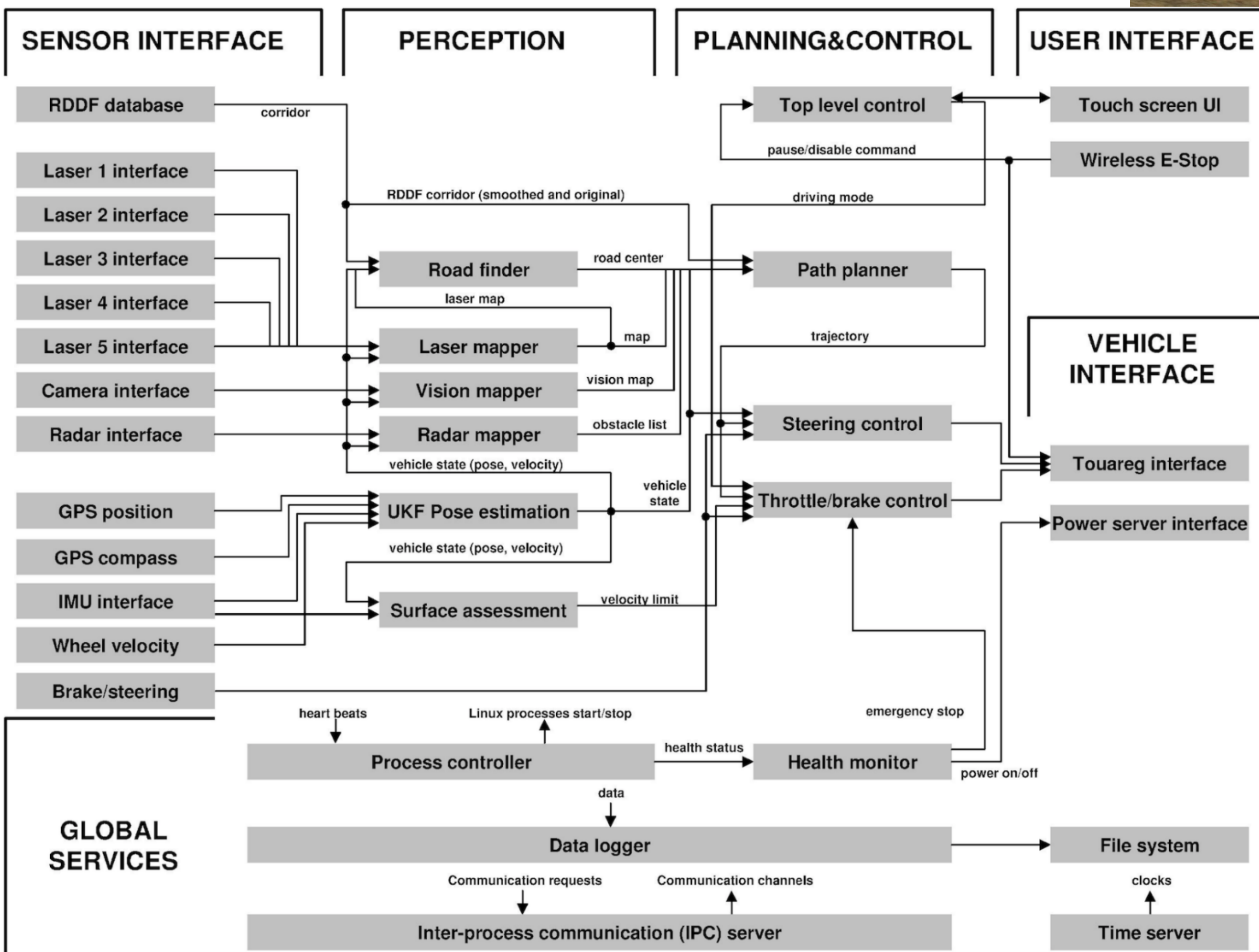
Hardware

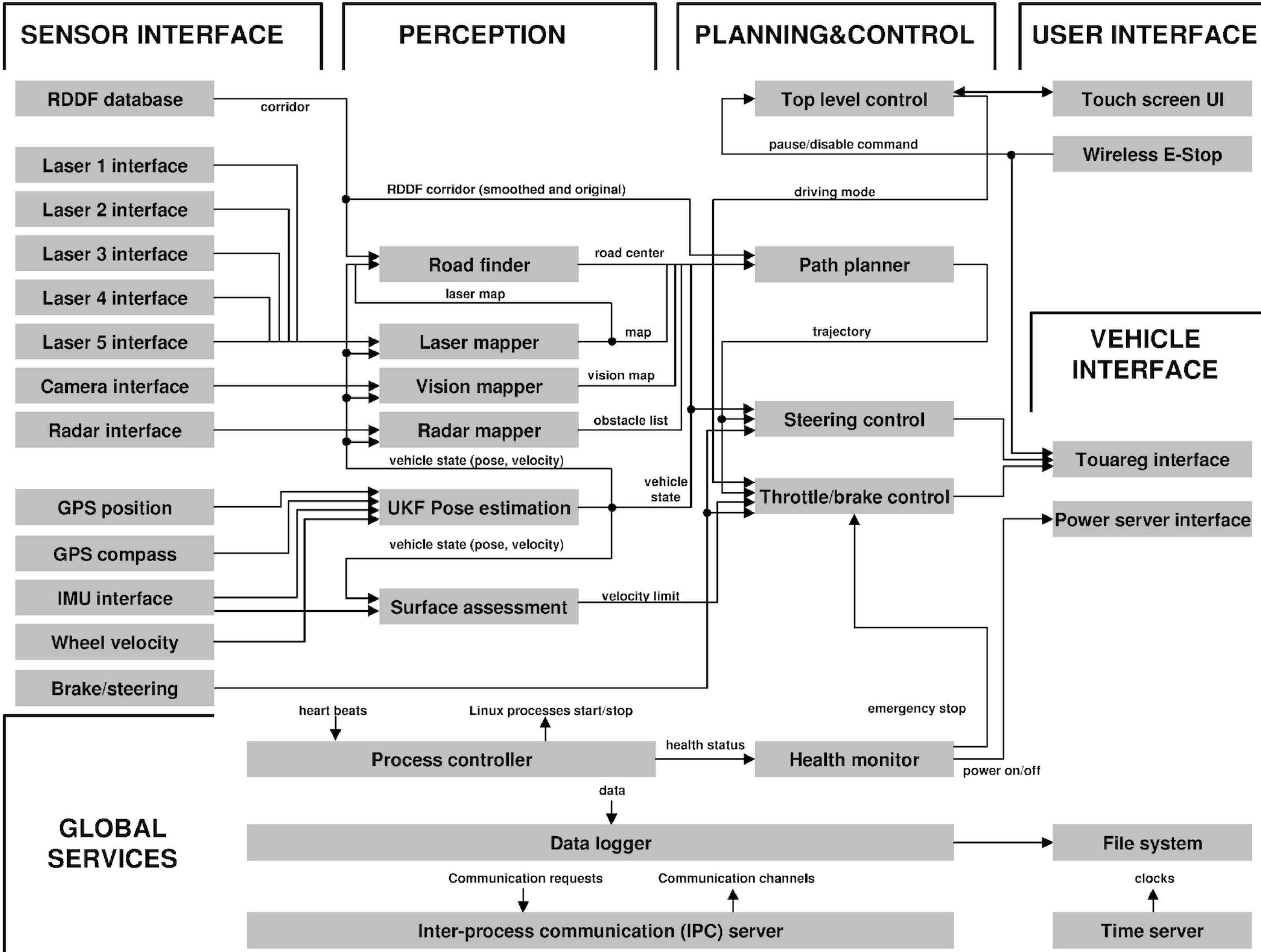


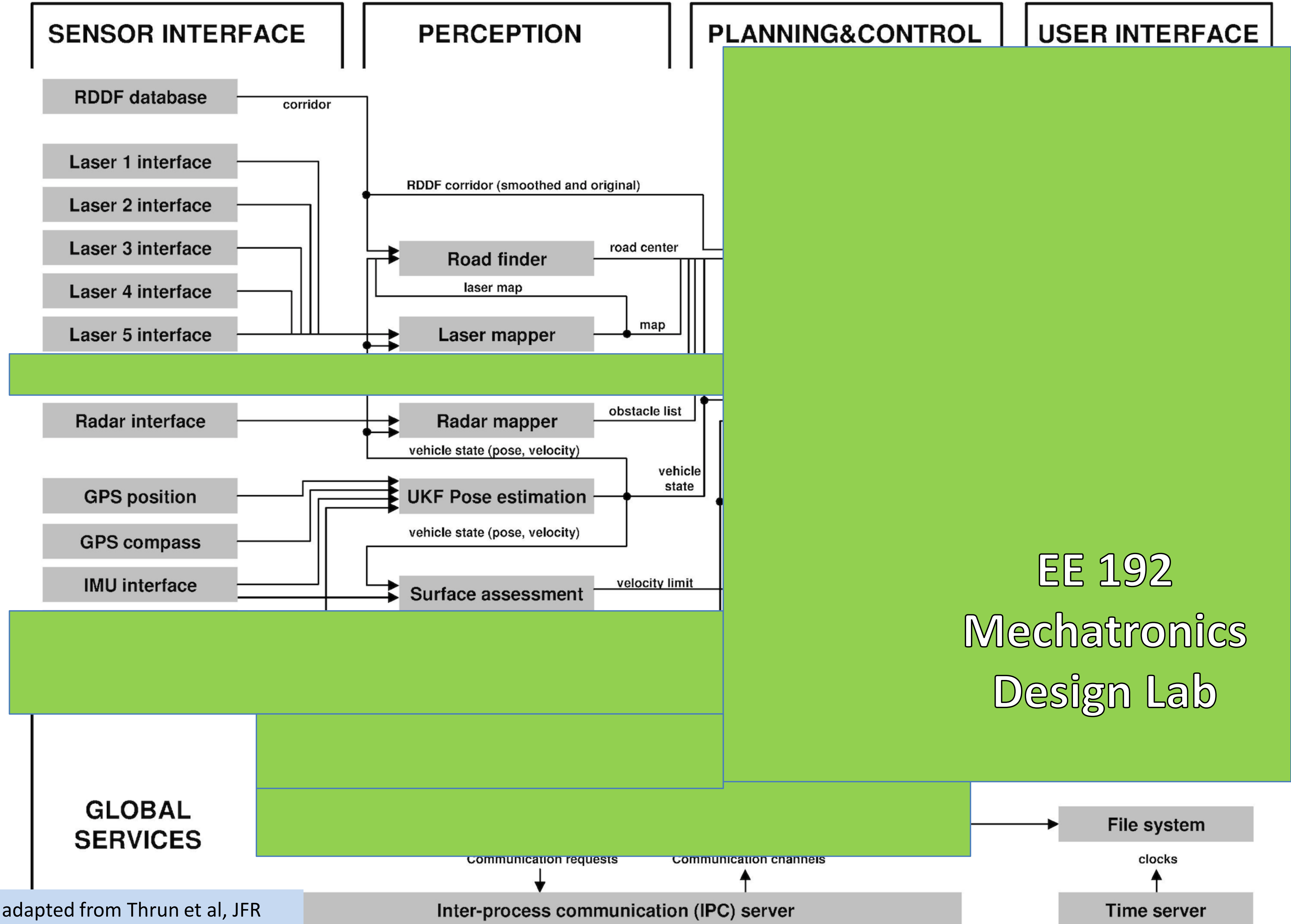
You should have:
- Huzzah32
- RCNitro or equiv car
- USB Battery

To be shipped from Berkeley
- Line camera
- Jumper wires
- Analog Discovery scope

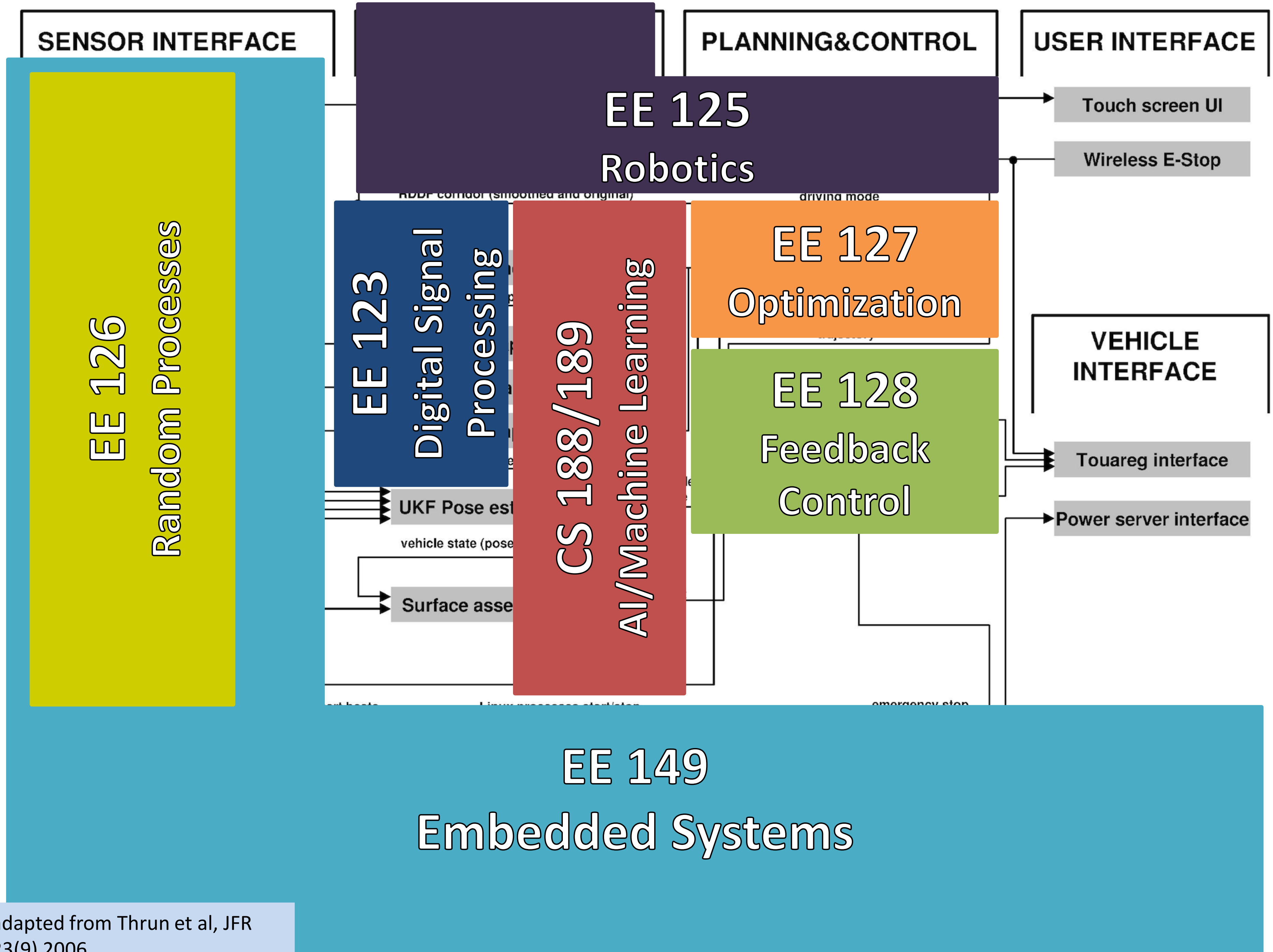
Thrun et al Stanley 2005




SENSOR INTERFACE
PERCEPTION
PLANNING&CONTROL
USER INTERFACE
RDDF database
corridor
Top level control
Touch screen UI
Wireless E-Stop
pause/disable command
Laser 1 interface
Laser 2 interface
Laser 3 interface
Laser 4 interface
Laser 5 interface
Camera interface
Radar interface
RDDF corridor (smoothed and original)
driving mode
Road finder
road center
Path planner
laser map
map
trajectory
VEHICLE INTERFACE
Laser mapper
Vision mapper
vision map
Radar mapper
obstacle list
Steering control
Touareg interface
vehicle state (pose, velocity)
vehicle state
GPS position
UKF Pose estimation
Throttle/brake control
Power server interface
GPS compass
vehicle state (pose, velocity)
IMU interface
Surface assessment
velocity limit
Wheel velocity
Brake/steering
heart beats
Linux processes start/stop
emergency stop
GLOBAL SERVICES
Process controller
health status
Health monitor
power on/off
data
Data logger
File system
Communication requests
Communication channels
clocks
Inter-process communication (IPC) server
Time server



adapted from Thrun et al, JFR 23(9) 2006



adapted from Thrun et al, JFR 23(9) 2006

Course Organization

- Tues Lecture -> Wed Lab Demo -> 9 days -> Fri checkoff hour (live Zoom) (tba)
- Partners: 2 is good, 3 is possible.
- Return equipment at end of semester for grade, and car+CPU for reimbursement
- Checkoffs ``better is enemy of good''- robustness: needs to work in a window

Course Organization (cont)

Checkpoint sequence:

- NEW: CPU -> drive motor+servo -> line sense -> line follow/fig 8 -> velocity control -> high speed steering

Round 1/Round2

Syllabus:

https://inst.eecs.berkeley.edu/~ee192/sp21/docs/syllabus.pdf

Course Organization (cont)

- Emphasis: robustness, simplicity. Design/Simulate/Build/Test
- Goal: 10 hours per week per team member.
  - Part of checkpoint: report weekly hours (important for course tuning)

- What about Complexity?
- Reliability of the overall system $(90\%)^N$

(connectors, power supply, ~~heat sinks, solder joints~~, CPU stack, car mechanics, camera mount, control stability, lighting robustness,…)

Checkpoint 0

Form a group of 2 (perhaps 3) students. (Remember, choose wisely!)

Have the GitHub usernames of all team members to get a private course GitHub repository.

Note: these repositories will be deleted at the end of the semester for re-use next semester. You are advised to keep a local clone!

CheckPoint 1 Fri Jan. 29

- Install tools, e.g. PlatformIO
- Compile and run SkeletonHuzzah32
- LED fade using PWM
- Timing:
  - How long does a double cos() take?
  - How long does log_add("string"); take compared to snprintf(log,sizeof(log),"string"); log_add(log);
  - How long does it take to print a floating point?

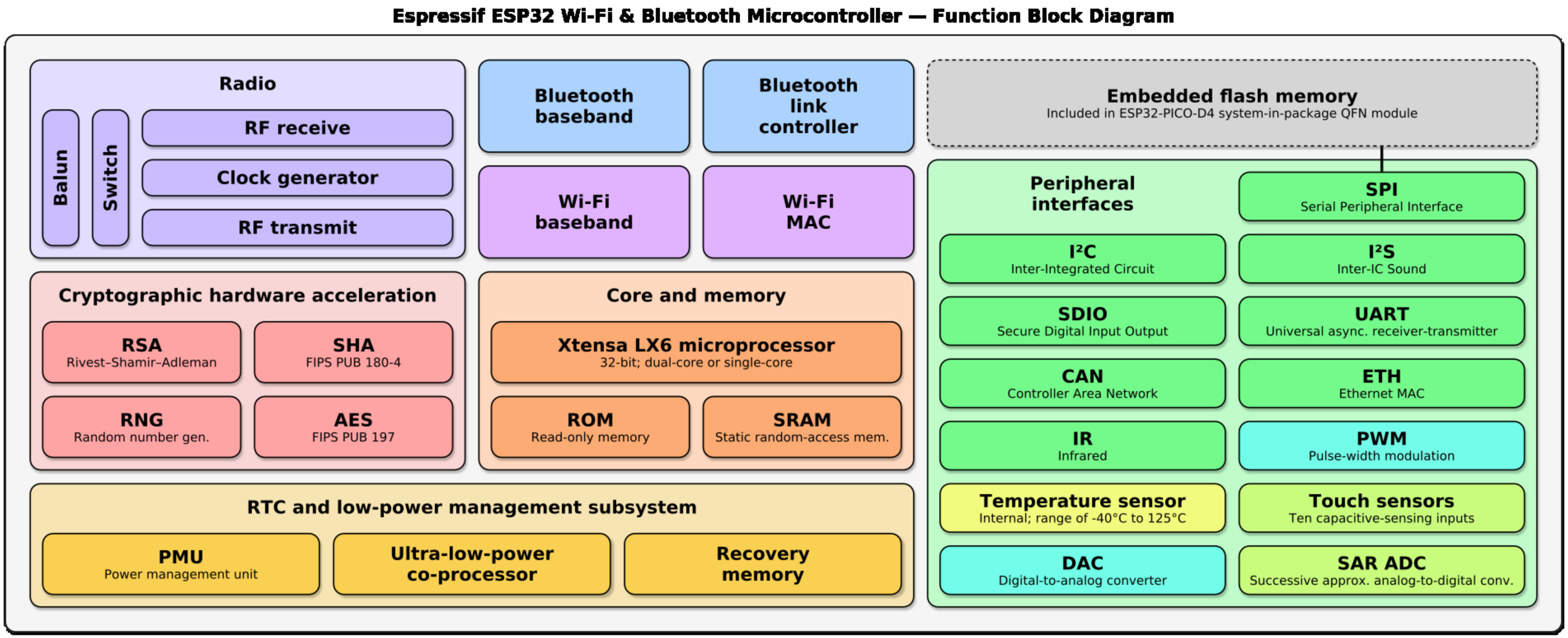
Huzzah32-ESP32

- 240 MHz dual core Tensilica LX6 microcontroller with 600 DMIPS
- Integrated 192 KB instruction SRAM, 200KB Data SRAM, 128KB either
- ESP32-WROOM-32 integrates a 4 MB SPI flash, which is connected to GPIO6, GPIO7, GPIO8, GPIO9, GPIO10 and GPIO11. These six pins cannot be used as regular GPIOs.
- Integrated 802.11b/g/n HT40 Wi-Fi transceiver, baseband, stack and LWIP

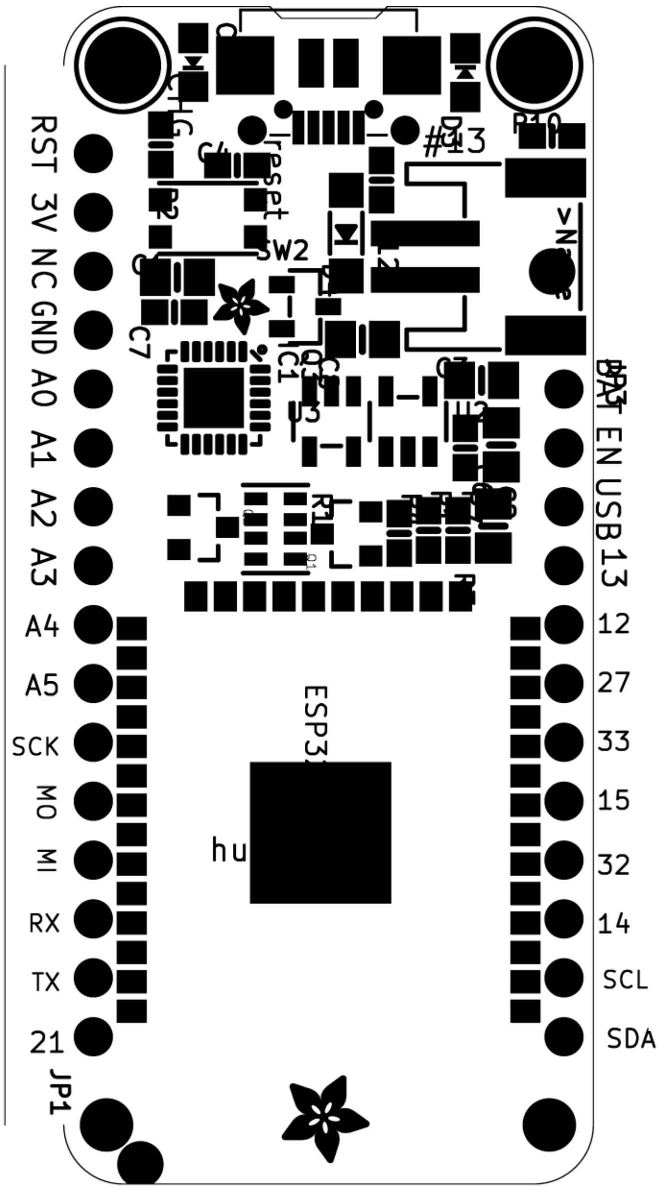
- 3 x UARTs
- 3 x SPI, 2 x I2C
- 12 x ADC input channels
- PWM/timer input/output available on every GPIO pin
- OpenOCD debug interface with 32 kB TRAX buffer

Integrated dual mode Bluetooth (classic and BLE)
Ultra-low noise analog amplifier
Hall sensor
10x capacitive touch interface
32 kHz crystal oscillator
2 x I2S Audio
2 x DAC
On-board PCB antenna
SDIO master/slave 50 MHz
SD card interface support

ESP32 Hardware block diagram



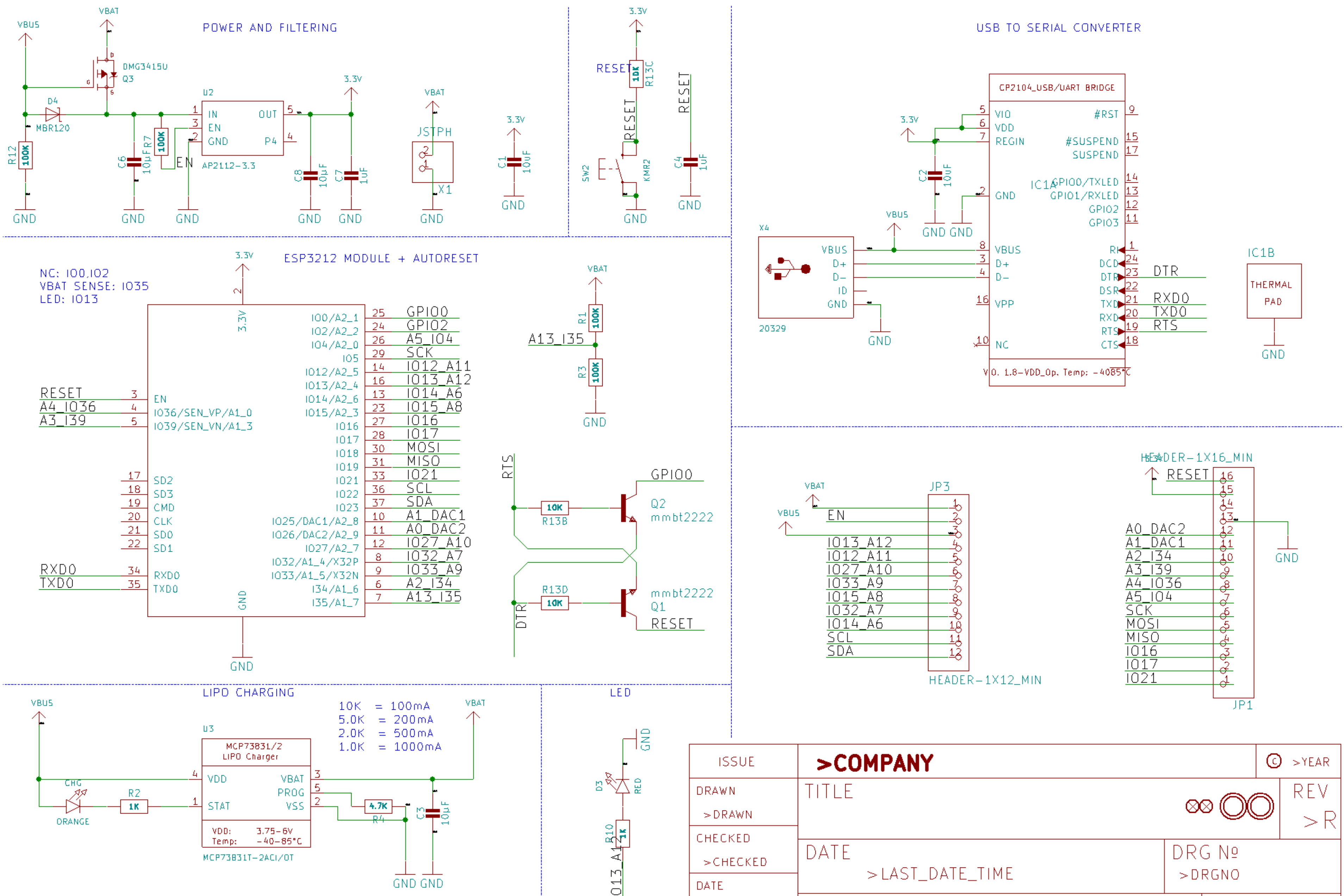
Huzzah32 pin out



GPIO	ALT	ID	
		RESET	1
		3.3V	2
			3
		GND	4
26	DAC2	A0	5
25	DAC1	A1	6
34	ADC6	A2	7
39	ADC3	A3	8
36	ADC0	A4	9
4		A5	10
5	SCK	A16	11
18	MOSI	A17	12
19	MISO	A18	13
16		A19	14
17		A20	15
21		A21	16

	ID	ALT	GPIO
28	VBAT		
27	EN 3.3V		
26	VUSB		
25	A12	LED	13
24	A11	BOOT	12
23	A10		27
22	A9	ADC5	33
21	A8		15
20	A7	ADC4	32
19	A6		14
18	A15	SCL	22
17	A14	SDA	23

Huzzah32 Schematic

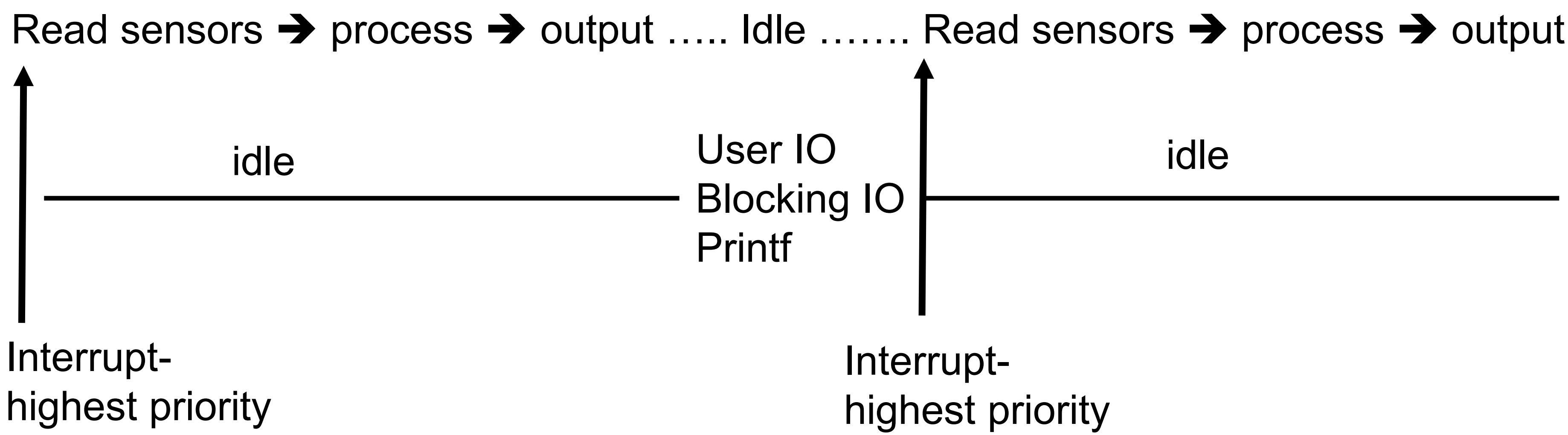


https://cdn-learn.adafruit.com/assets/assets/000/041/630/large1024/feather_schem.png?1494449413

Software Introduction

- Why Real-time? Bare metal vs Linux
- Measuring Timing
- Tasks in FreeRTOS
- Printing to UART and log task

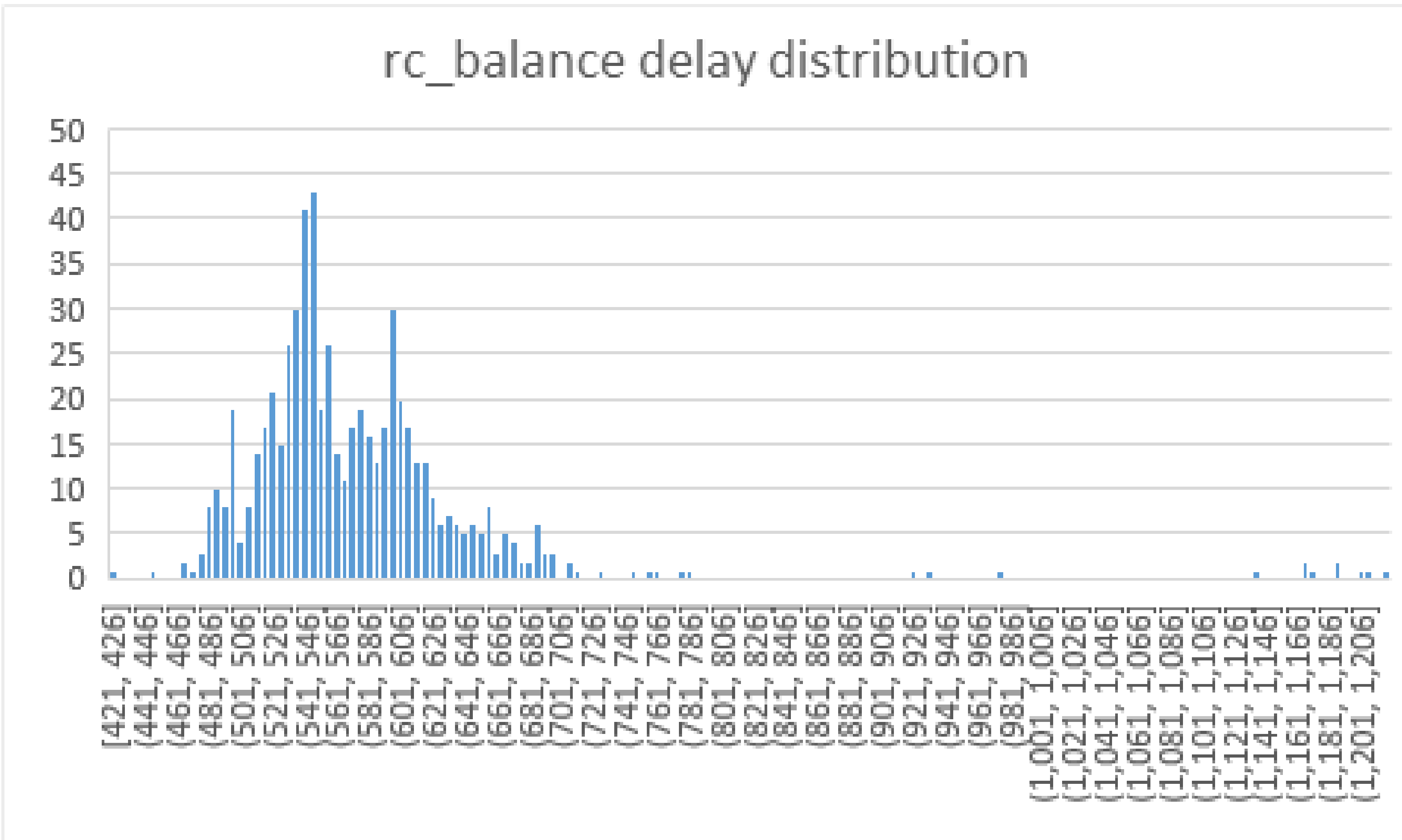
Software Notes-Basic real time model



Delay leads to instability (for EE128/ME134- negative phase), will show later when discuss control

Example Timing Uncertainty in Linux: main control loop

Calculation loop: input/process/output. Note outliers.

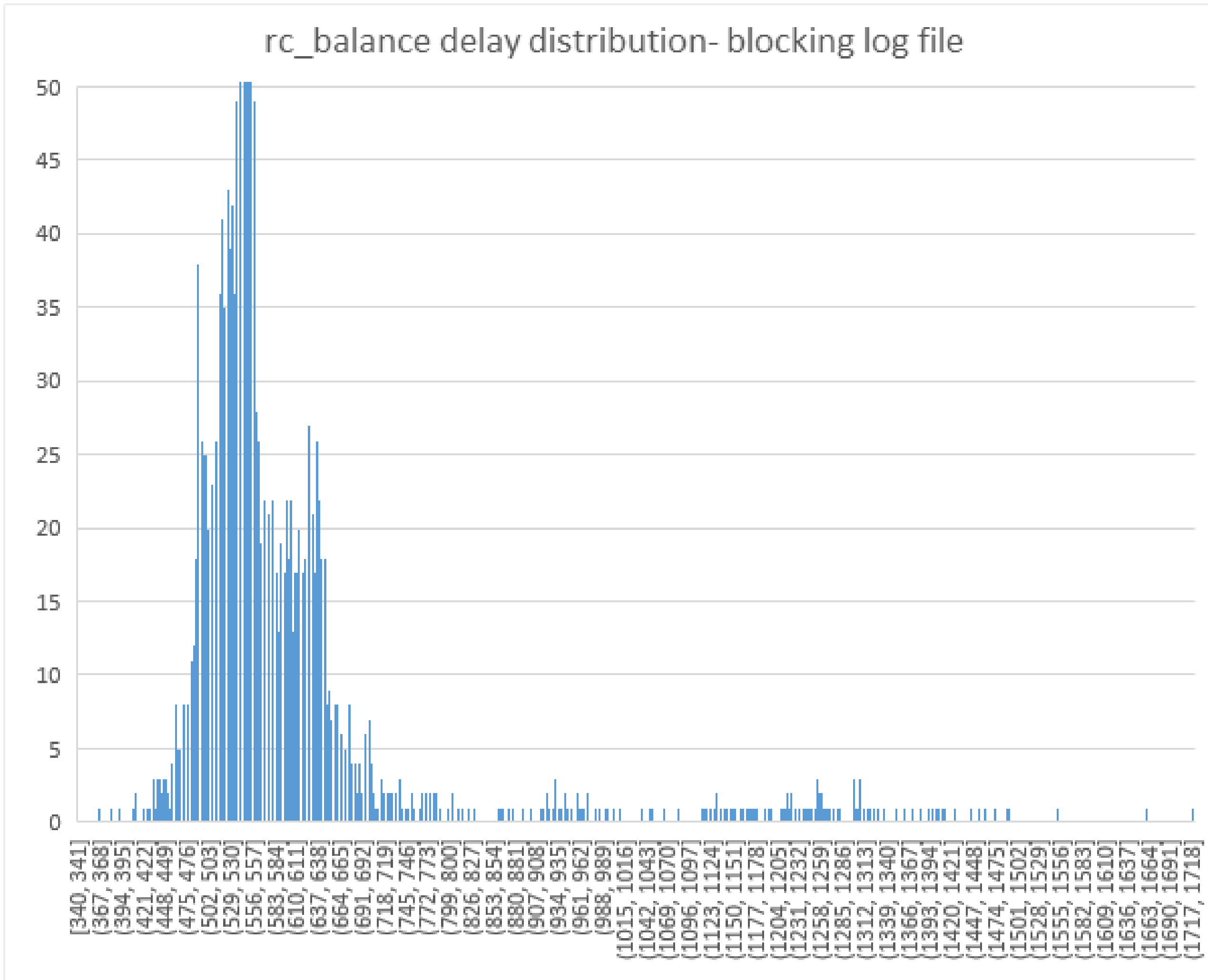


420 us

1200 us

Example Timing Uncertainty in Linux

Using printf in control loop (NOT RECOMMENDED)



340 us

1700 us

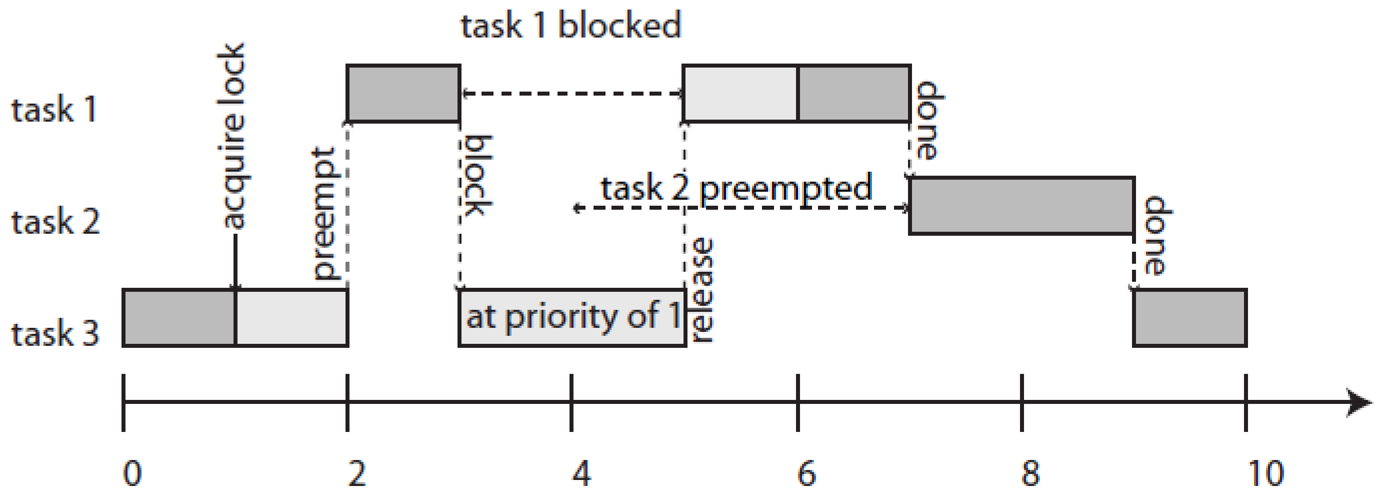


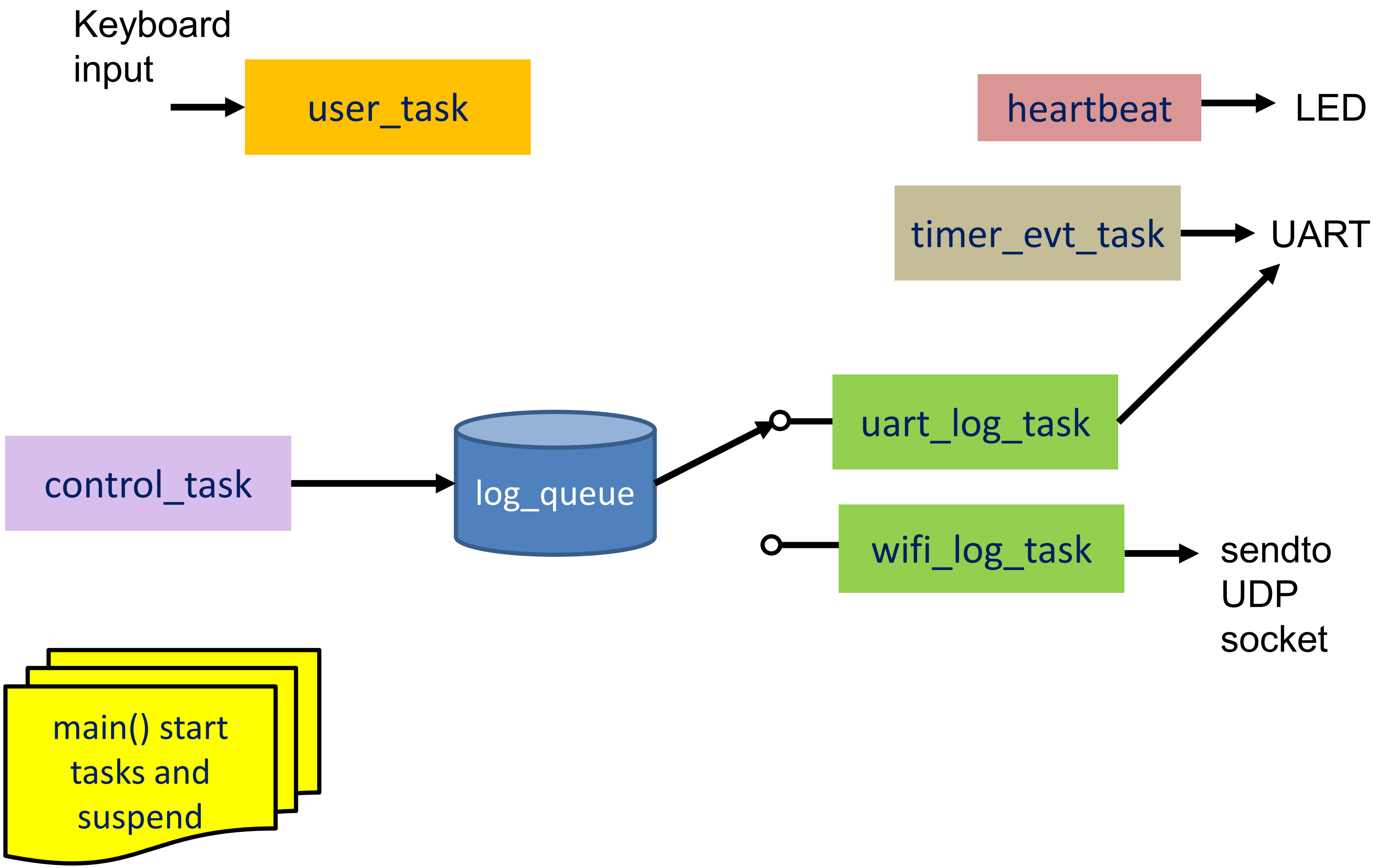
Figure 12.10: Illustration of the priority inheritance protocol. Task 1 has highest priority, task 3 lowest. Task 3 acquires a lock on a shared object, entering a critical section. It gets preempted by task 1, which then tries to acquire the lock and blocks. Task 3 inherits the priority of task 1, preventing preemption by task 2.

FreeRTOS+VS+PlatformIO

- Queue – used across timing domains for coherence of data
- Multiple tasks created at different priority for handling control, sensors, logging, etc
  - Log task: use for non-blocking printf for debugging

- Task List to monitor tasks (time used, stack used)

SkeletonHuzzah32 SW Block Diagram



Note conventions- data flow left to right

Measuring Timing from ESP32

```
tick_start = xTaskGetTickCount();
```

This gives resolution of 1 ms, depending on `CONFIG_FREERTOS_HZ`

High resolution timing using built-in 64 bit counter

```
uint64_t task_counter_value;
double starttime;

timer_get_counter_value(TIMER_GROUP_0, TIMER_0,
&task_counter_value);

starttime=((double)task_counter_value/TIMER_SCALE);
```

Non-blocking print

https://github.com/ucb-ee192/SkeletonHuzzah32

```
snprintf(log, sizeof(log),
         "Idle. sum of cos = %d \n",(long)ZSum);
log_add(log);
```

← snprintf is too slow, can use itoa(), etc

```
void log_add(char *log)
{ xQueueSend(log_queue, log, 0);
     // send data to back of queue,
    // non-blocking, wait=0 ==> return
immediately if the queue is already full.

}
static void uart_log_task(void *pvParameters)
{ …
xQueueReceive(log_queue, log, portMAX_DELAY);
```

Timing of printf, etc

took about 40 us with log_add(),
but 970 us with snprintf()

Checkpoint 1: measure timing for real-time debugging

Task creation example

```
/* heartbeattask.c*/
static void heartbeatTask(void *pvParameters); //
static=local


if(xTaskCreate(heartbeat,
"WRITE_TASK_1",
configMINIMAL_STACK_SIZE+1024,
NULL,
tskIDLE_PRIORITY + 2,
NULL)
      != pdPASS)
     {   printf("Task creation failed!.\r\n");
         while(1); // hang indefinitely
     }
```

C function

Name for debugging purposes

Stack size for task

Pointer to parameters to pass into task

Task priority (0=lowest)

Optional handle to created task

https://github.com/ucb-ee192/SkeletonHuzzah32

Monitoring FreeRTOS Tasks- Stack Usage

```
void print_tasks();
```

# of tasks 12

Task name	**number of cycles**		
IDLE0	283440623	49%	(CPU 0)
usertask	142791054	24%	
IDLE1	145004887	25%	(CPU 1)
heartbeat	6661	<1%	
timer_evt_task	194739	<1%	
Tmr Svc	55	<1%	
control_task	60360	<1%	
esp_timer	209	<1%	
ipc0	10215	<1%	
main	95764	<1%	
log_task	13490	<1%	
ipc1	15121	<1%	(inter process comm?)

➔ Starving the ``idle” process (will cause a crash).
➔ Make sure every process has vTaskDelay() for a lower priority process to run

Monitoring FreeRTOS Tasks

```
void print_tasks();
```

Name	State	Priority	Stack	Task#
control_task	R	2	348	14
usertask	R	0	504	16
IDLE1	R	0	1116	7
IDLE0	R	0	1012	6
heartbeat	B	1	1584	15
timer_evt_task	B	2	756	13
Tmr Svc	B	1	1592	8
main	S	1	2476	5
log_task	B	1	856	12
esp_timer	B	22	3640	1
ipc1	B	24	596	3
ipc0	B	24	564	2

Priority: 0 is lowest priority. Usertask is also low priority as it busy waits for input
Task #: order of task startup
State: R running, B blocked, S suspend

Class Introductions

Name
Year/major
Location (time zone)
Have a partner/team already?

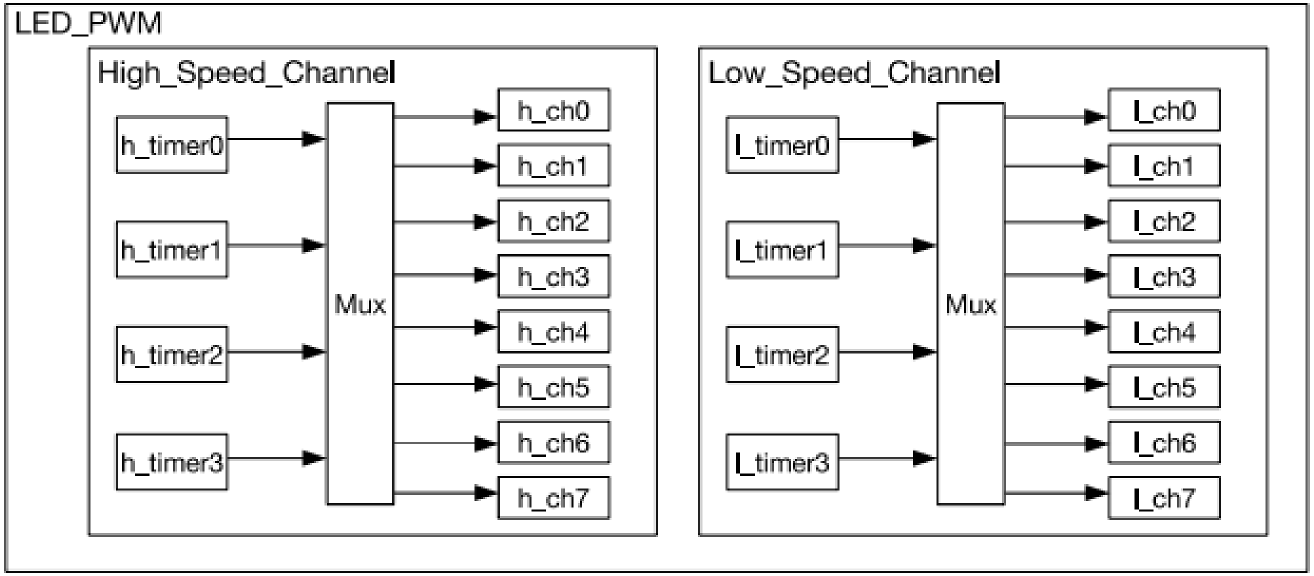
Extra Slides

Pulse Width Modulation

https://github.com/espressif/esp-idf/tree/b0150615dff529662772a60dcb57d5b559f480e2/examples/peripherals/mcpwm

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

LED PWM controller (Ch 15)



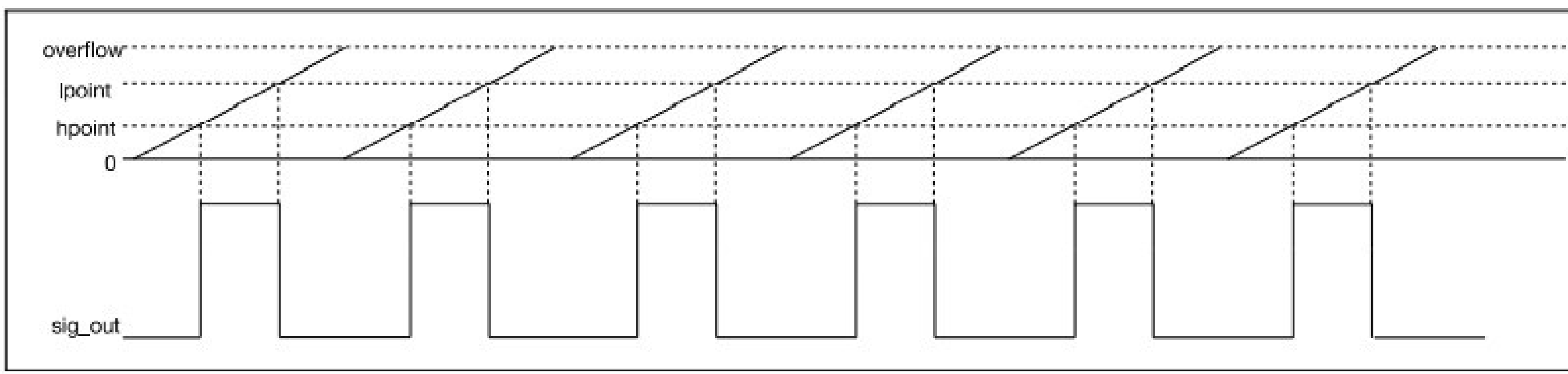


Figure 88: LED PWM Output Signal Diagram

LED PWM controller (Ch 15)

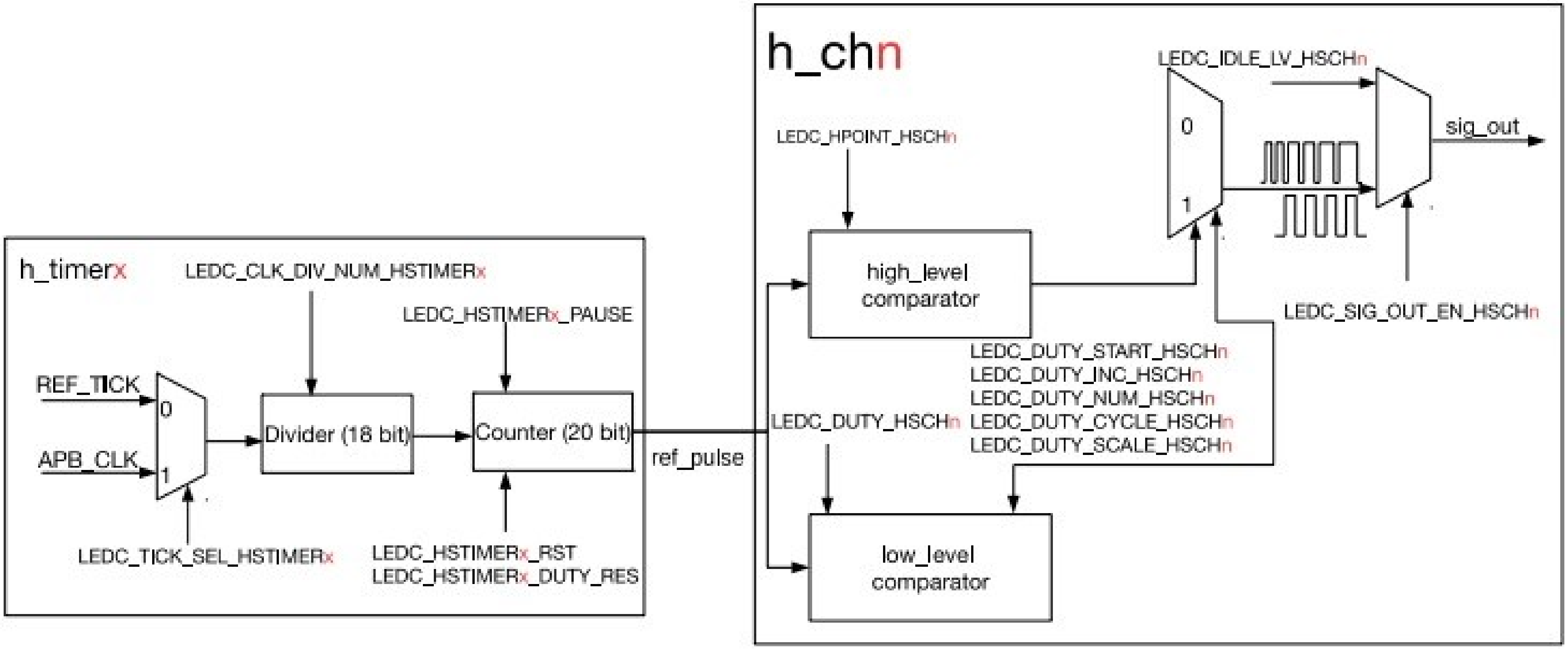


Figure 86: LED_PWM High-speed Channel Diagram

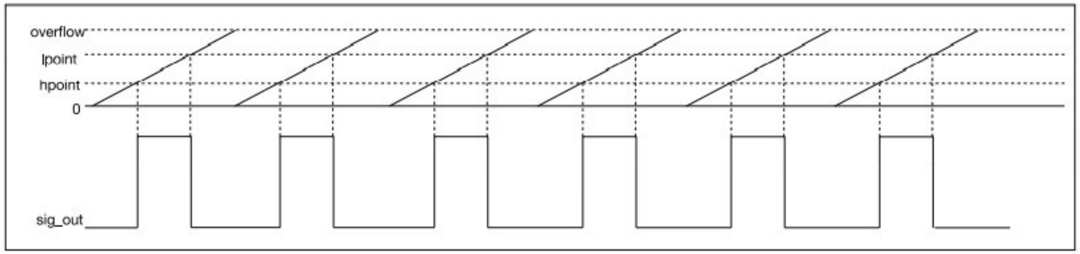
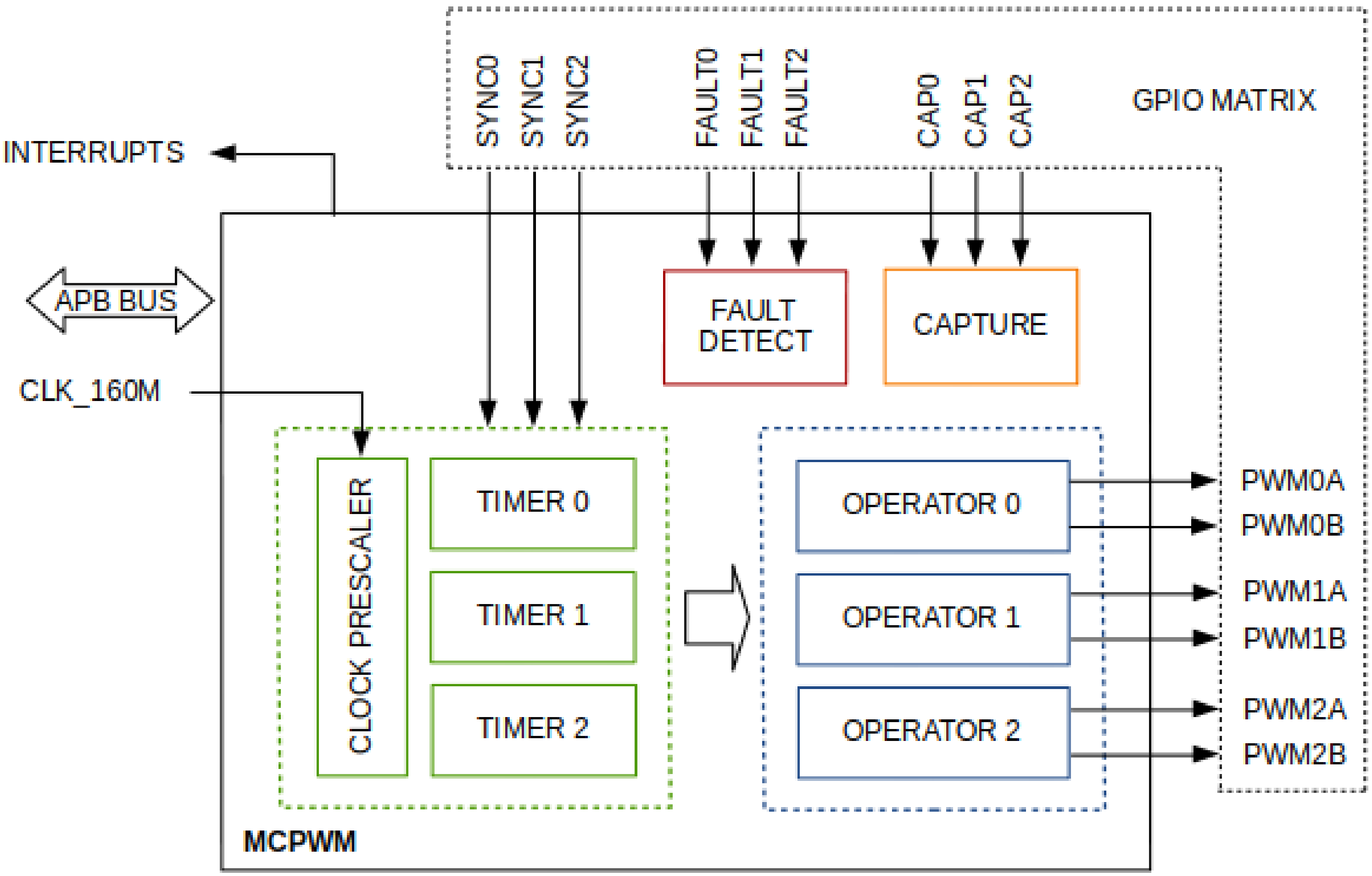


Figure 88: LED PWM Output Signal Diagram

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



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

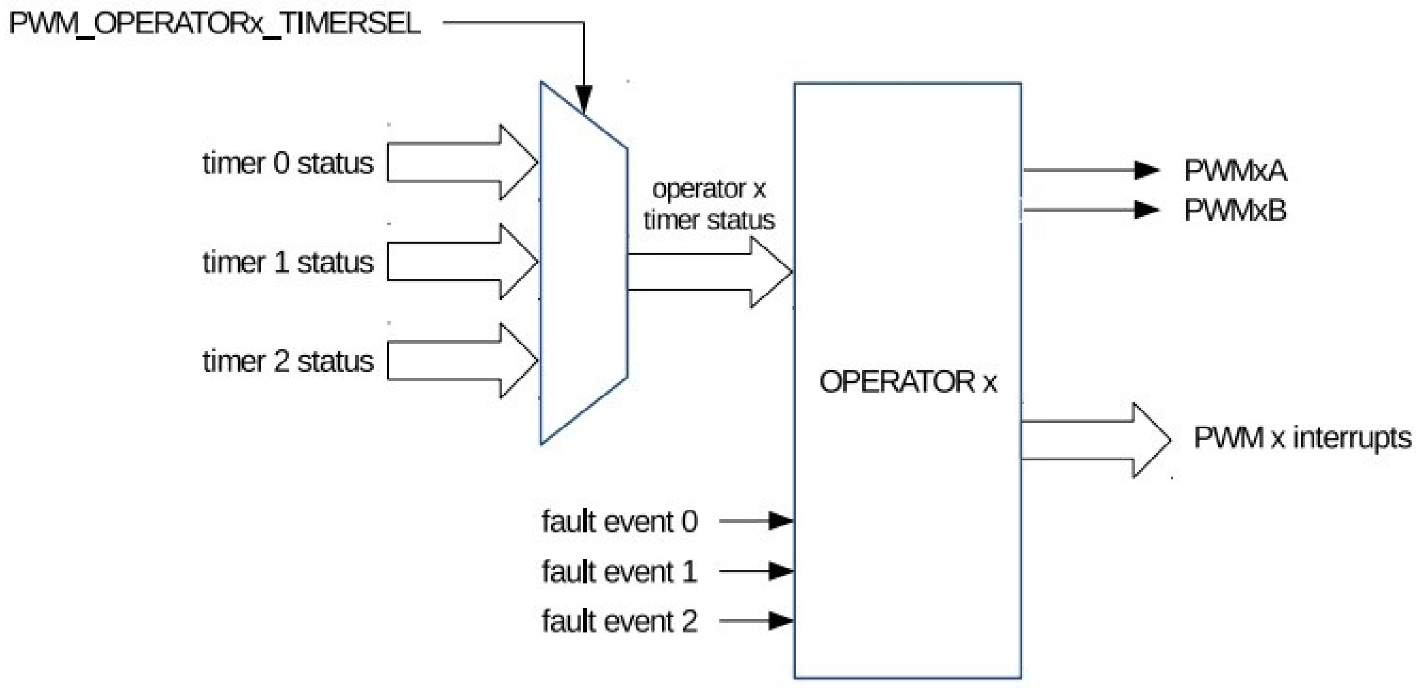


Figure 95: Operator Submodule

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

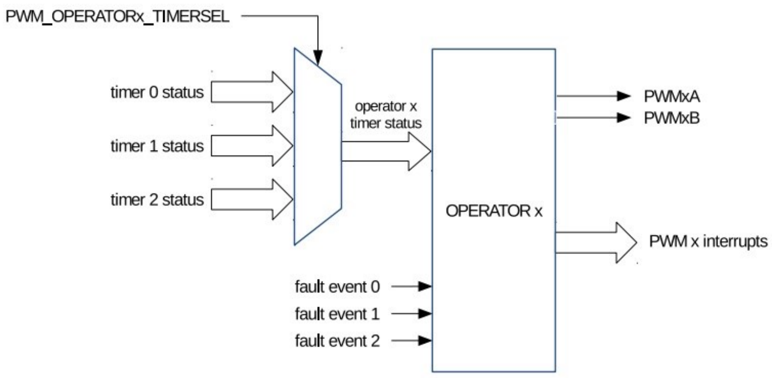


Figure 95: Operator Submodule

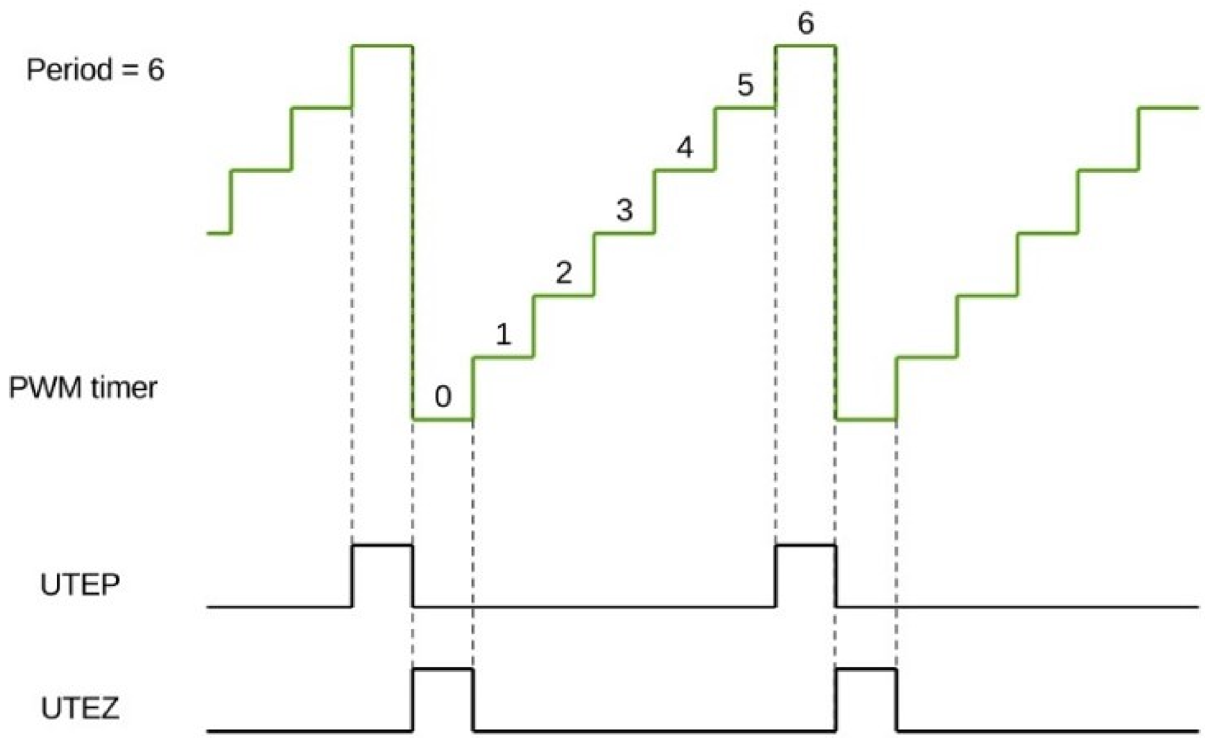
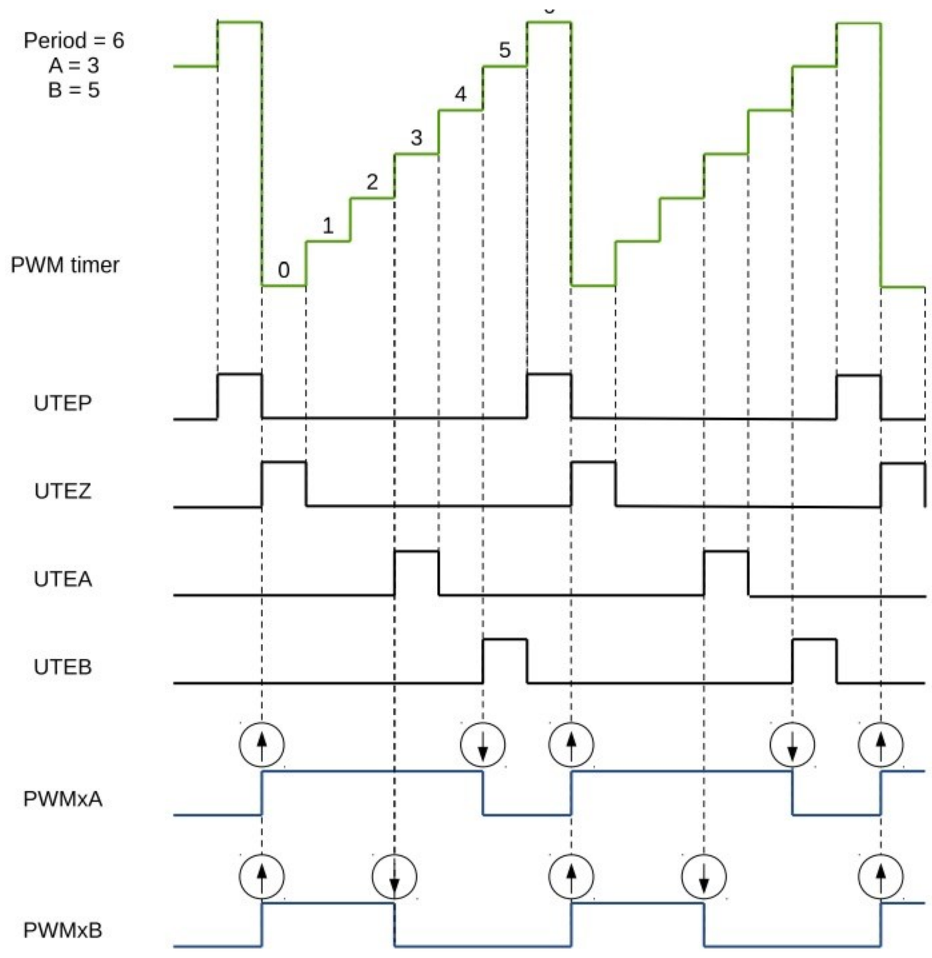


Figure 102: UTEP and UTEZ Generation in Count-Up Mode

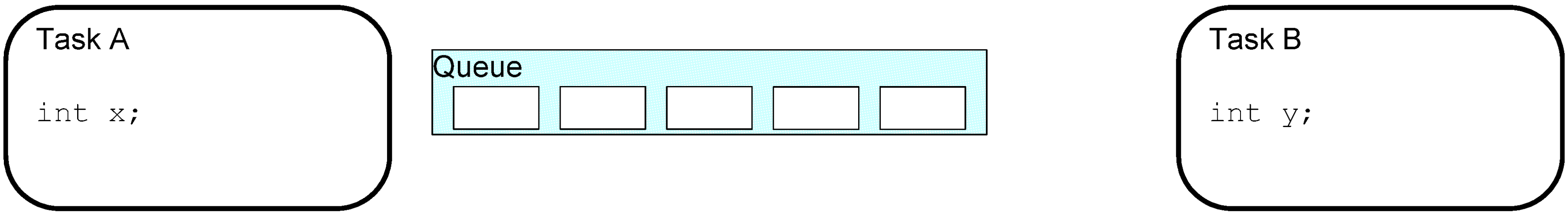
Motor Control Pulse Width Modulator (MCPWM) (Ch 17)



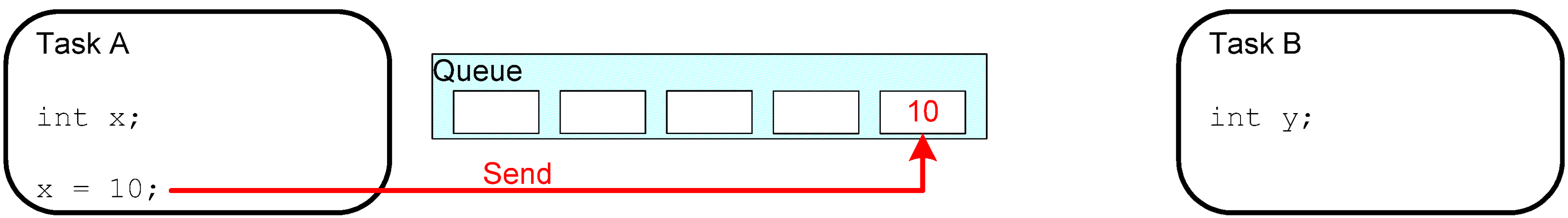
– UTEA: the PWM timer is counting up and its value is equal to register A.
– UTEB: the PWM timer is counting up and its value is equal to register B.
– DTEA: the PWM timer is counting down and its value is equal to register A.
– DTEB: the PWM timer is counting down and its value is equal to register B.

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.

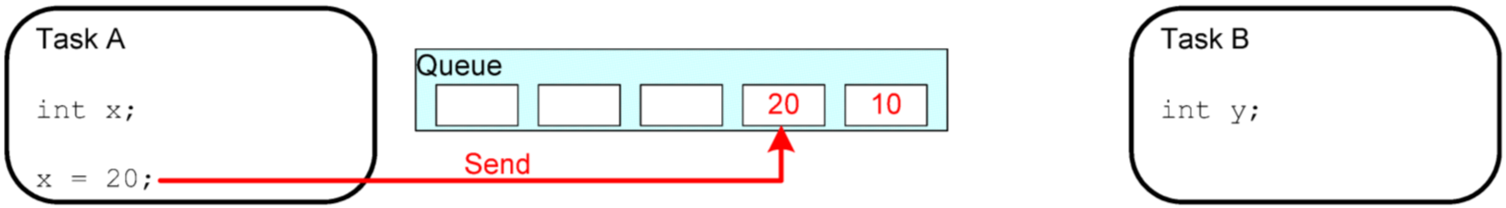


A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.

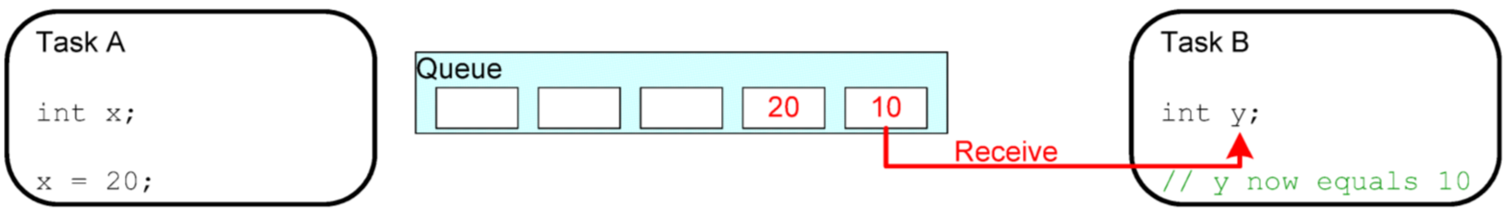


Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.

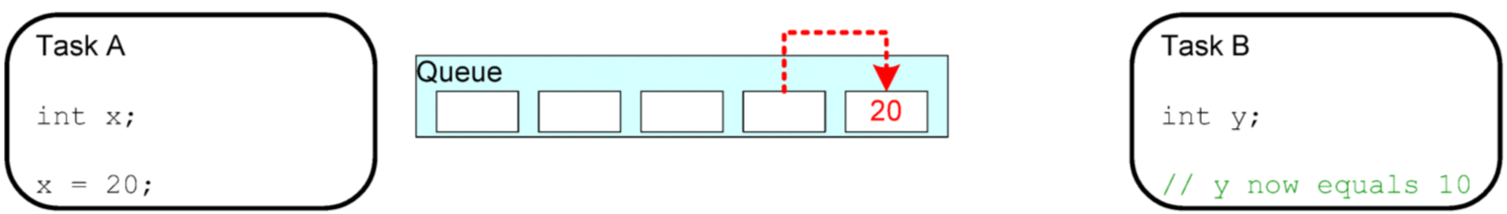
Queue in FreeRTOS



Task A changes the value of its local variable before writing it to the queue again. 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.

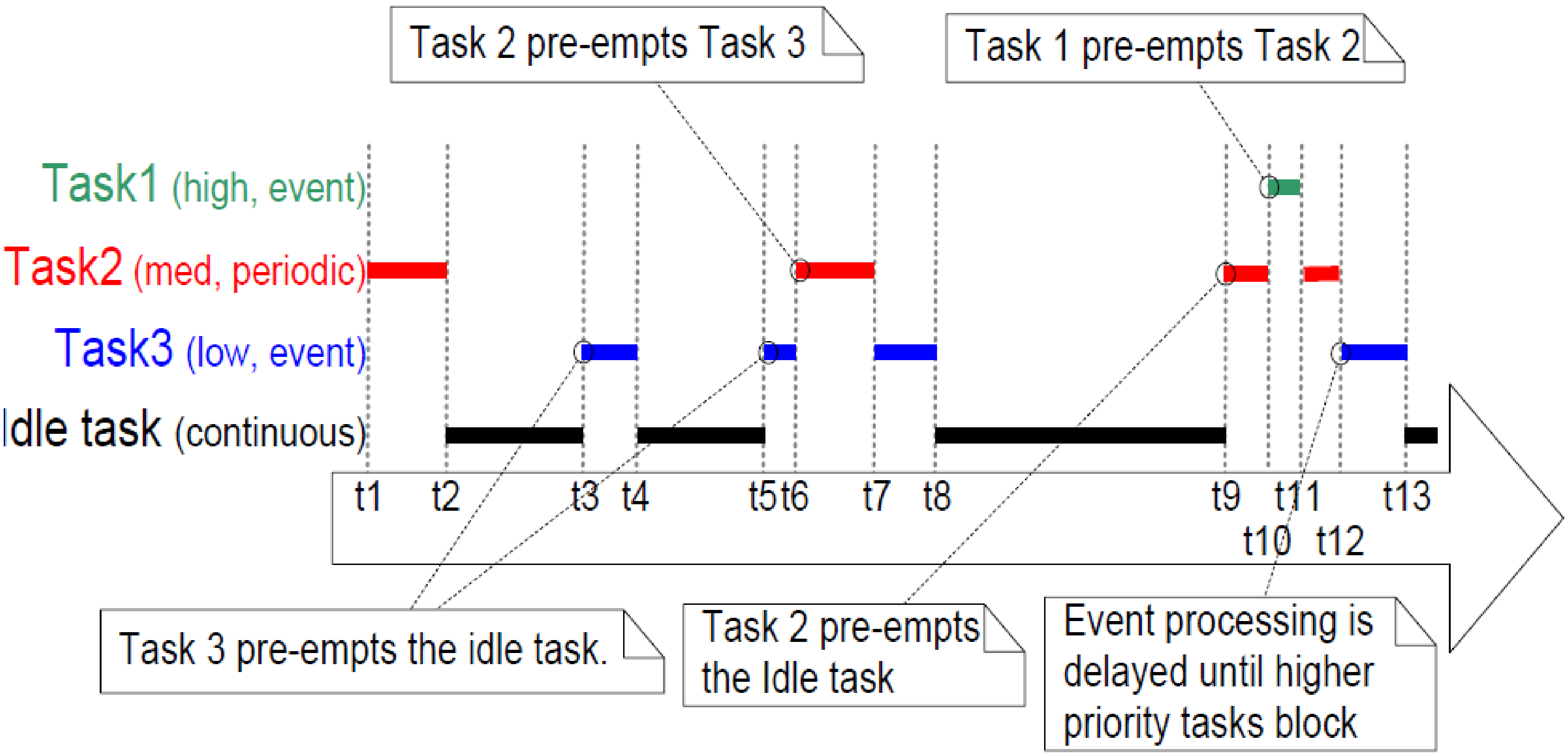


Task B reads (receives) from the queue into a different variable. The value received by Task B is the value from the head of the queue, which is the first value Task A wrote to the queue (10 in this illustration).



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



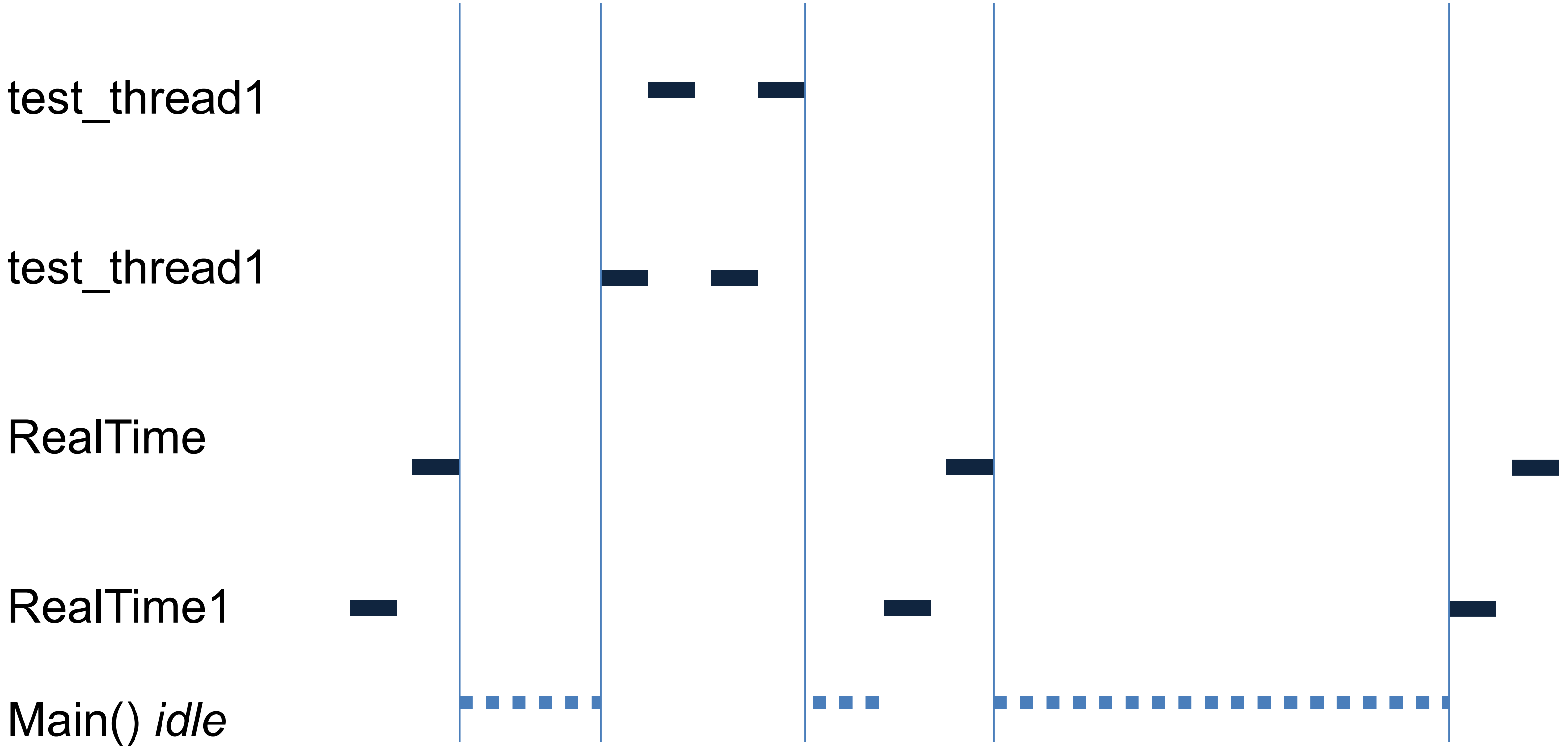
26. Execution pattern highlighting task prioritization and pre-emption in a hypothetical application in which each task has been assigned a unique priority

Process ID/Memory Management (Ch 28)

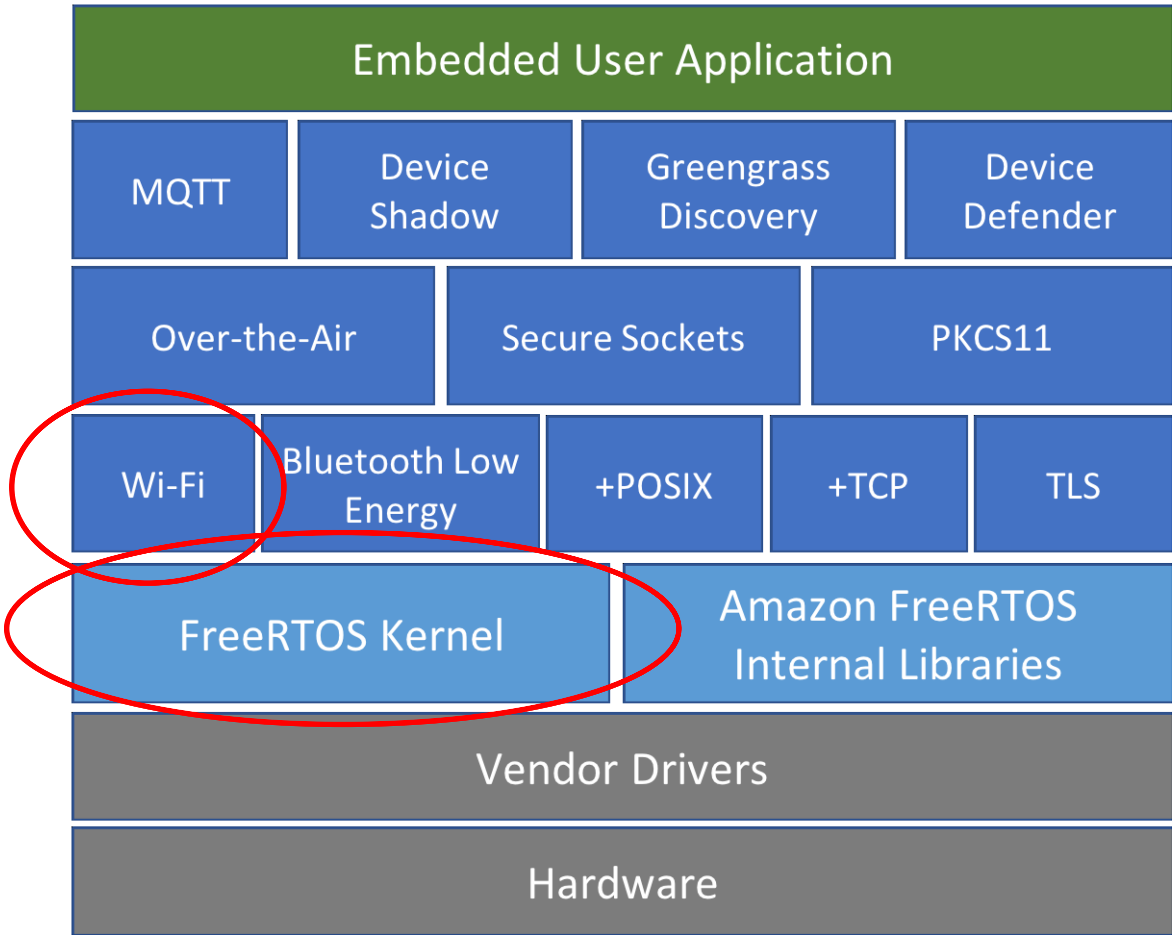
More specifically, when a code tries to access a MMU/MPU-protected memory region or peripheral, the MMU or MPU will receive the PID from the PID generator that is associated with the CPU on which the process is running.

??? void vTaskAllocateMPURegions(TaskHandle_t xTask, const MemoryRegion_t *const pxRegions)

Project Proposal: RTOS timer and threads example

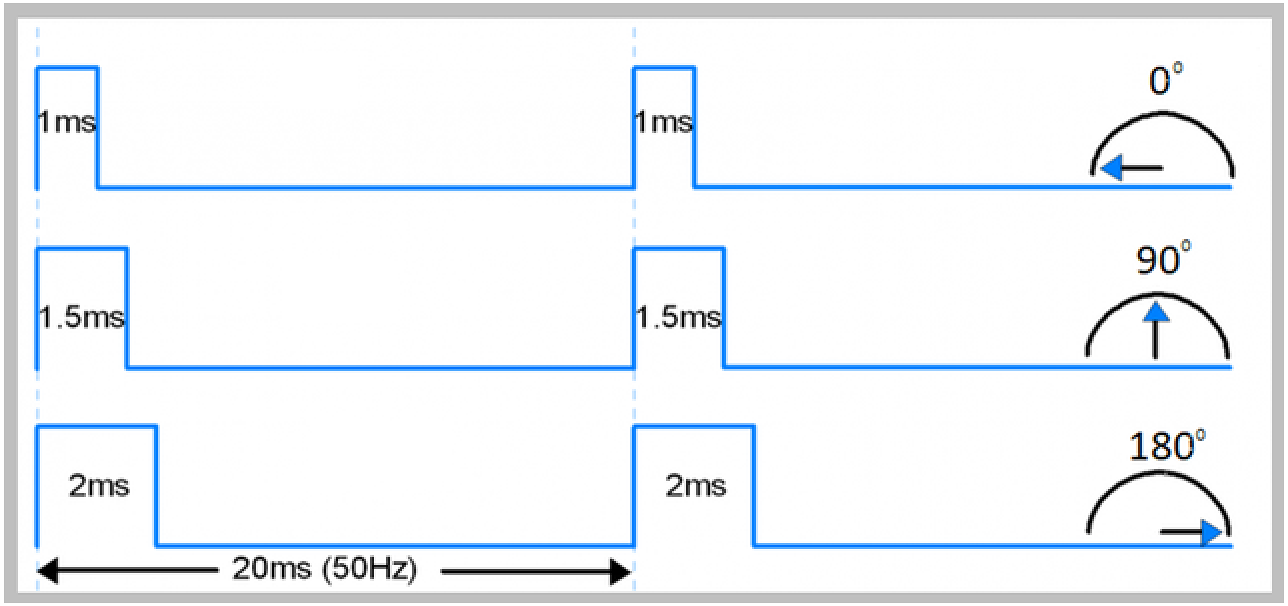


Instead of Ubuntu: FreeRTOS



MQTT: lightweight, publish-subscribe network protocol
The AWS IoT Greengrass Discovery library is used by your microcontroller devices to discover a Greengrass core on your network.

Servo PWM



https://www.instructables.com/id/PANTILT-Camera-With-ESP32/