

FreeRTOS

Real-time operating system for microcontrollers

Bedřich Said

409874@mail.muni.cz

Faculty of Informatics, Masaryk University

May 16, 2020

What you can find in this presentation

- FreeRTOS implementation in ESP32
- The basics of FreeRTOS in examples
- Concurrent data structures included in FreeRTOS

What is not included

- No deep theory and details about RTOS
- No other RTOS than FreeRTOS
- Nothing about how to compile and run FreeRTOS under your platform

Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

- New task creation

- Task properties

- Multiple tasks

- Jobs

- Timers

Sharing data among threads

- Critical sections

- Queues

FreeRTOS webpage

New blog post available
Gaurav Aggarwal describes run-time options when using FreeRTOS on ARM Cortex-M33 and M23 cores. [Learn more...](#) X

 Quality RTOS & Embedded Software
[Download FreeRTOS](#)

[Kernel](#) [Libraries](#) [Resources](#) [Community](#) [Support](#) [Email List](#) 

FreeRTOS™
Real-time operating system for microcontrollers

Developed in partnership with the world's leading chip companies over a 15-year period, and now downloaded every 175 seconds, FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of libraries suitable for use across all industry sectors. FreeRTOS is built with an emphasis on reliability and ease of use.

[Download FreeRTOS](#) [Getting Started](#)

<https://www.freertos.org/>

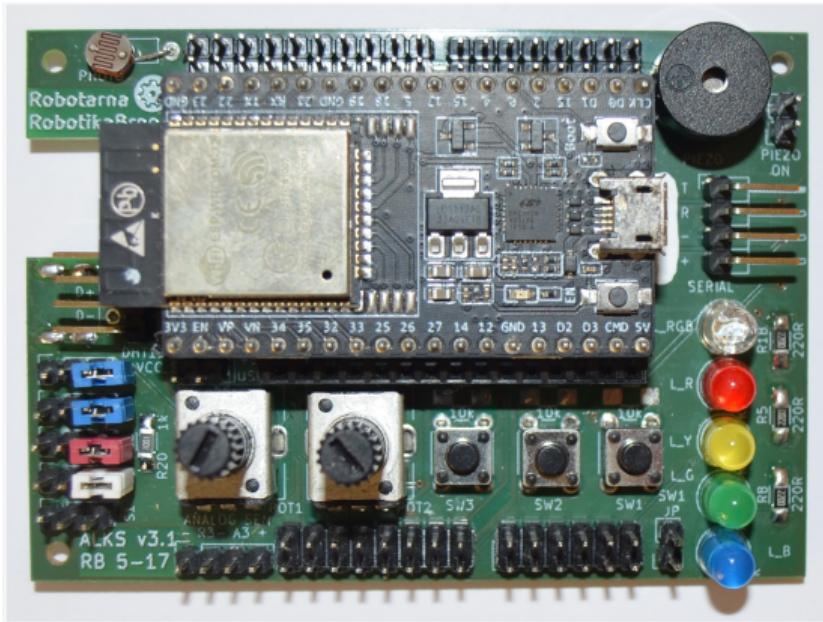
We will work with ESP32 dual core processor



www.espressif.com/en/products/hardware/esp32-devkitc/overview

Hardware used for the following examples

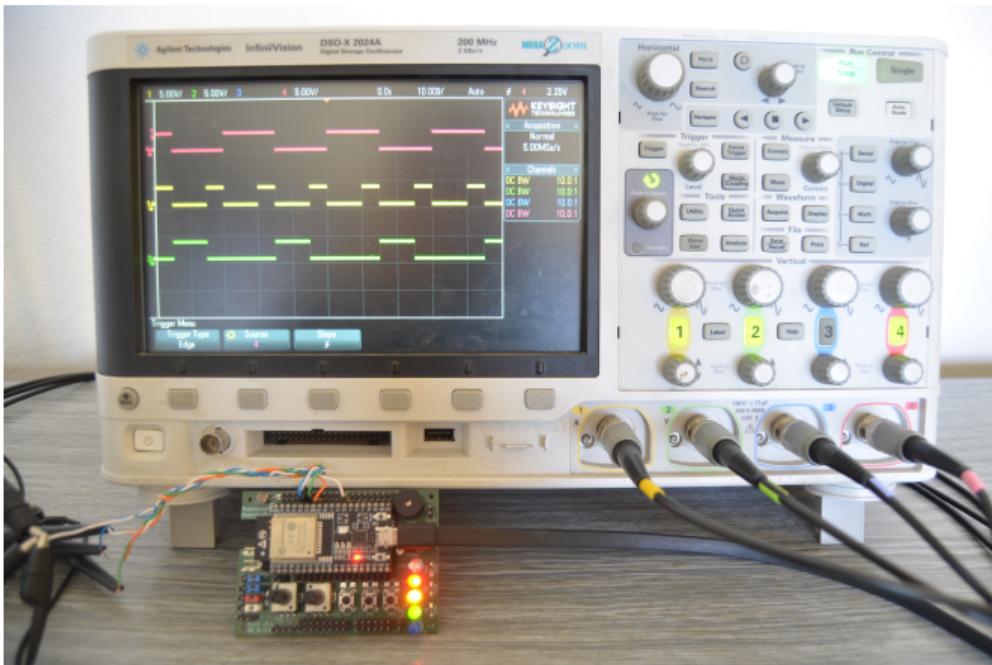
Arduino Learning Kit Starter with ESP32 microcontroller



<https://github.com/RoboticsBrno/ArduinoLearningKitStarter>

Hardware used for the following examples

... connected to oscilloscope

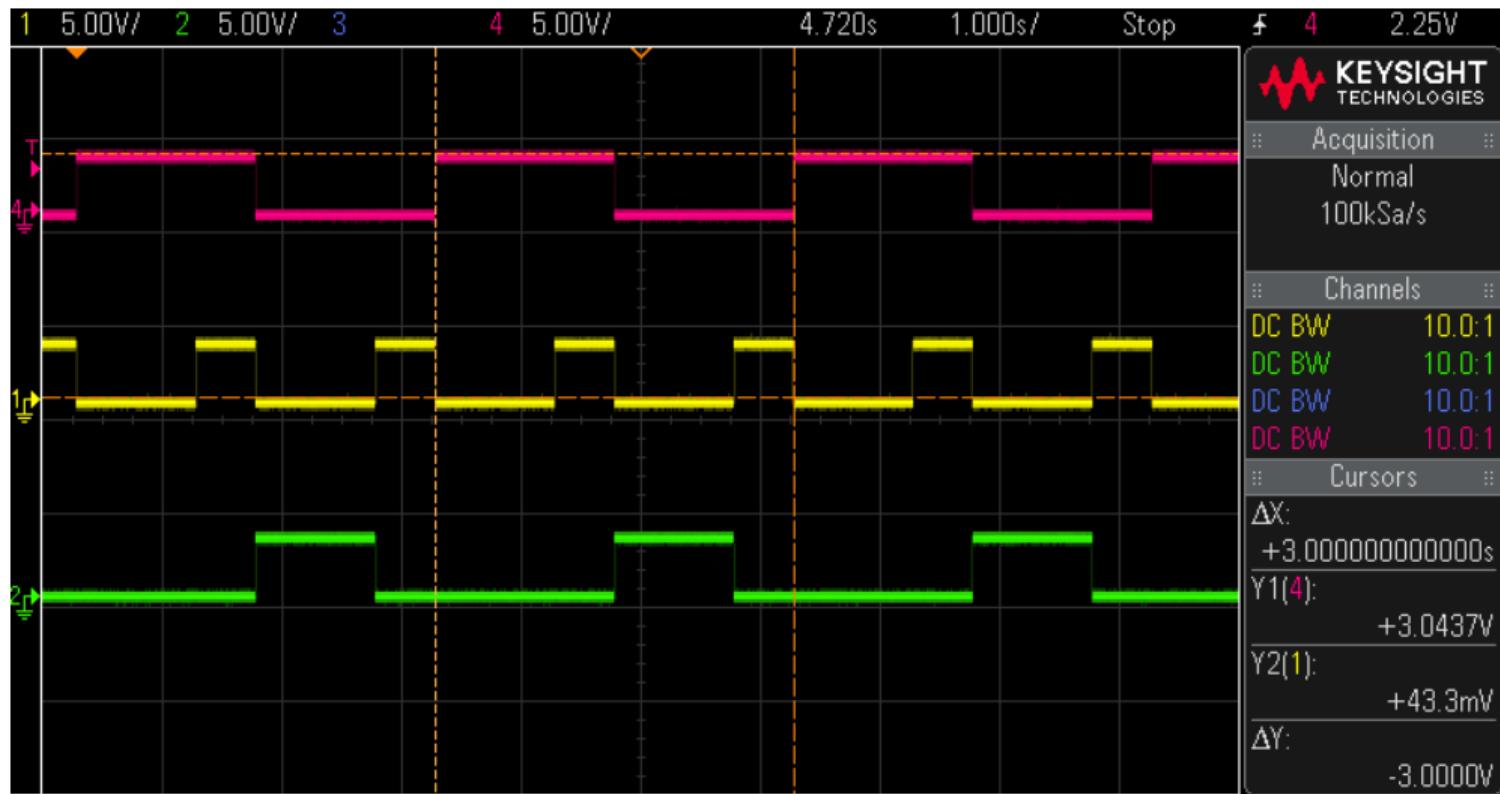


How the oscilloscope works

Presented in the video

- Oscilloscope measures voltage with time
- It displays a graph (X axis = time; Y axis = voltage)
- Our FreeRTOS tasks control LEDs
- We can see which task is running = which LED is blinking
- Fast changes can be seen by the oscilloscope

How the oscilloscope works



Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

- New task creation

- Task properties

- Multiple tasks

- Jobs

- Timers

Sharing data among threads

- Critical sections

- Queues

Minimal working example

```
1 #include <Arduino.h>
2
3 // called one time in the beginning
4 void setup() {
5     pinMode(L_R, OUTPUT); // Activate red LED
6 }
7
8 // repeats forever
9 void loop() {
10    digitalWrite(L_R, HIGH); // Red LED on
11    delay(1000); // wait 1000 ms
12    digitalWrite(L_R, LOW); // Red LED off
13    vTaskDelay(1000); // wait 1000 scheduler ticks
14 }
```

Minimal working example

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "OPEN EDITORS" and "RTOSTEST". The "src" folder contains "main.cpp" and "platformio.ini".
- Code Editor:** Displays the content of main.cpp, which includes the Arduino.h header and implements setup() and loop() functions.
- Terminal:** Shows the build and upload process for a FreeRTOS application:
 - Hash of data verified.
 - Compressed 8192 bytes to 47... .
 - Writing at 0x0000e000... (100 %)
 - Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds (effective 4501.3 kbit/s)...
 - Hash of data verified.
 - Compressed 211040 bytes to 106372... .
 - Writing at 0x00010000... (14 %)
 - Writing at 0x00014000... (28 %)
 - Writing at 0x00018000... (42 %)
 - Writing at 0x0001c000... (57 %)
 - Writing at 0x00020000... (71 %)
 - Writing at 0x00024000... (85 %)
 - Writing at 0x00028000... (100 %)
 - Wrote 211040 bytes (106372 compressed) at 0x00010000 in 9.5 seconds (effective 178.6 kbit/s)...
 - Hash of data verified.
- Status Bar:** Shows the terminal output: "=====[SUCCESS] Took 18.87 seconds =====". It also indicates "Terminal will be reused by tasks, press any key to close it."
- Bottom Bar:** Includes icons for file operations like Open, Save, Find, and Replace, along with status indicators for tabs, file count, and connection.

delay() vs. vTaskDelay()

```
1 // Busy wait for 1000 ms  
2 delay(1000);
```

VS.

```
1 // Delay a task for a given number of RTOS ticks  
2 vTaskDelay(1000);
```

VS.

```
1 // Delay a task for 1000 ms  
2 vTaskDelay(1000 / portTICK_PERIOD_MS);
```

<https://www.freertos.org/a00127.html>

delay() vs. vTaskDelay()

The screenshot shows the FreeRTOS website's navigation bar. The top navigation includes links for Kernel, Libraries, Resources, Community, and Support. Below this, under the 'Kernel' heading, there is a detailed list of API reference links, including 'Task Control' which is currently selected.

Kernel > API Reference > Task Control [API]

KERNEL

- [Home](#)
- [Getting Started](#)
- [FreeRTOS Books](#)
- [About FreeRTOS Kernel](#)
- [Developer Docs](#)
- [Secondary Docs](#)
- [Supported Devices](#)
- [API Reference](#)
 - [Task Creation](#)
 - [Task Control](#)
 - [vTaskDelay\(\)](#)
 - [vTaskDelayUntil\(\)](#)
 - [uxTaskPriorityGet\(\)](#)

Task Control

[API]

Modules

- [vTaskDelay](#)
- [vTaskDelayUntil](#)
- [uxTaskPriorityGet](#)
- [vTaskPrioritySet](#)
- [vTaskSuspend](#)
- [vTaskResume](#)
- [xTaskResumeFromISR](#)
- [xTaskAbortDelay](#)

<https://www.freertos.org/a00127.html>

Note: FreeRTOS naming convention

```
1 void vTaskDelay( const TickType_t xTicksToDelay );
2 BaseType_t xTaskCreate( ... );
```

- **v** – return type, here void

- u unsigned

- l long

- s short

- c char

- p pointer

- x variables of non stdint types

- **TaskDelay** – name in camel case

www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html

Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

New task creation

Task properties

Multiple tasks

Jobs

Timers

Sharing data among threads

Critical sections

Queues

New task creation

```
1 BaseType_t xTaskCreate(
2     TaskFunction_t pvTaskCode, // Called function
3     const char * const pcName, // Task name (string)
4     configSTACK_DEPTH_TYPE usStackDepth, // Stack size
5     void *pvParameters, // Parameters given to the task
6     UBaseType_t uxPriority, // Priority of the task
7     TaskHandle_t *pxCreatedTask // Task handle
8 );
```

<https://www.freertos.org/a00125.html>

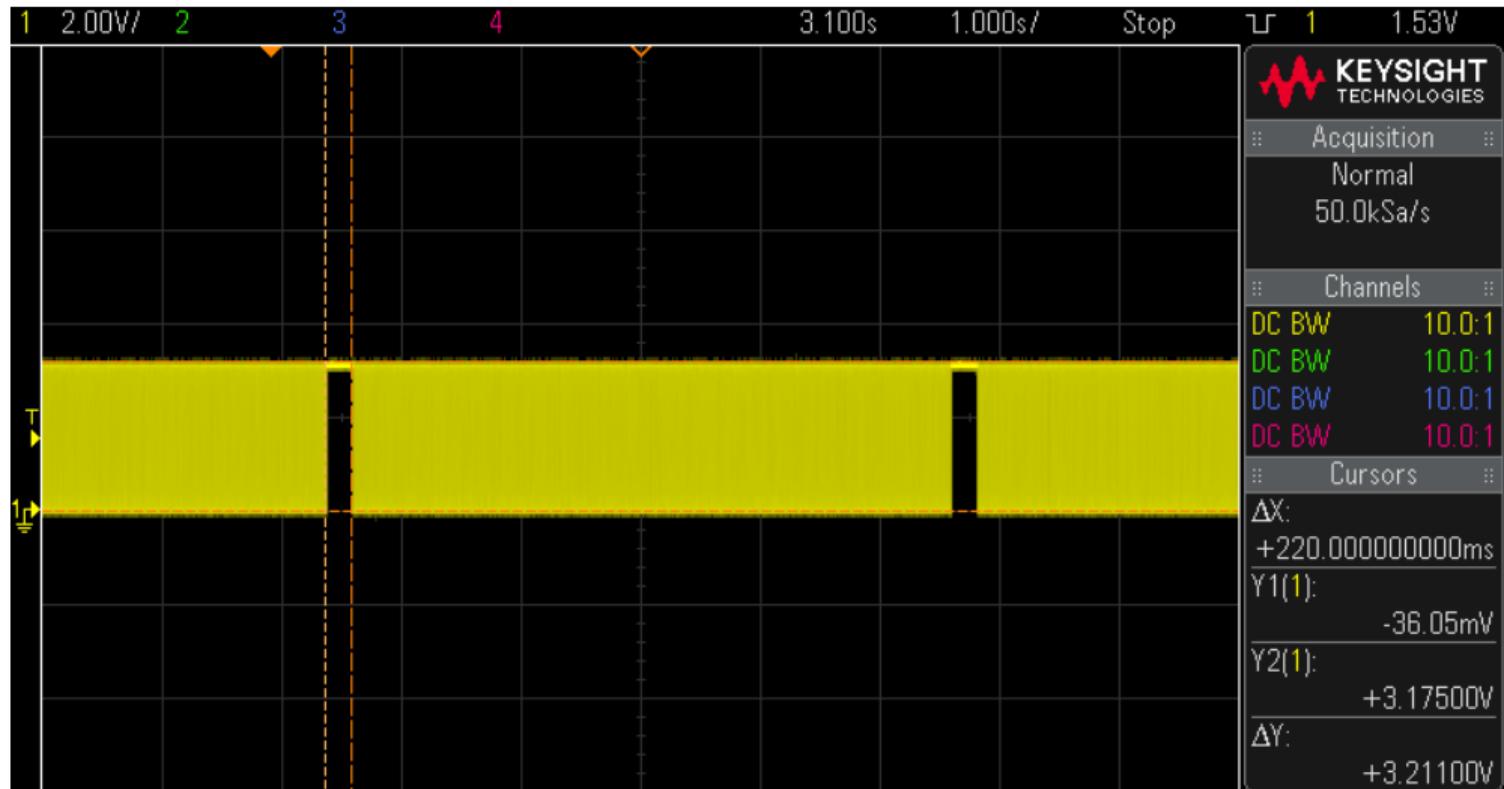
New task creation

Imagine a function

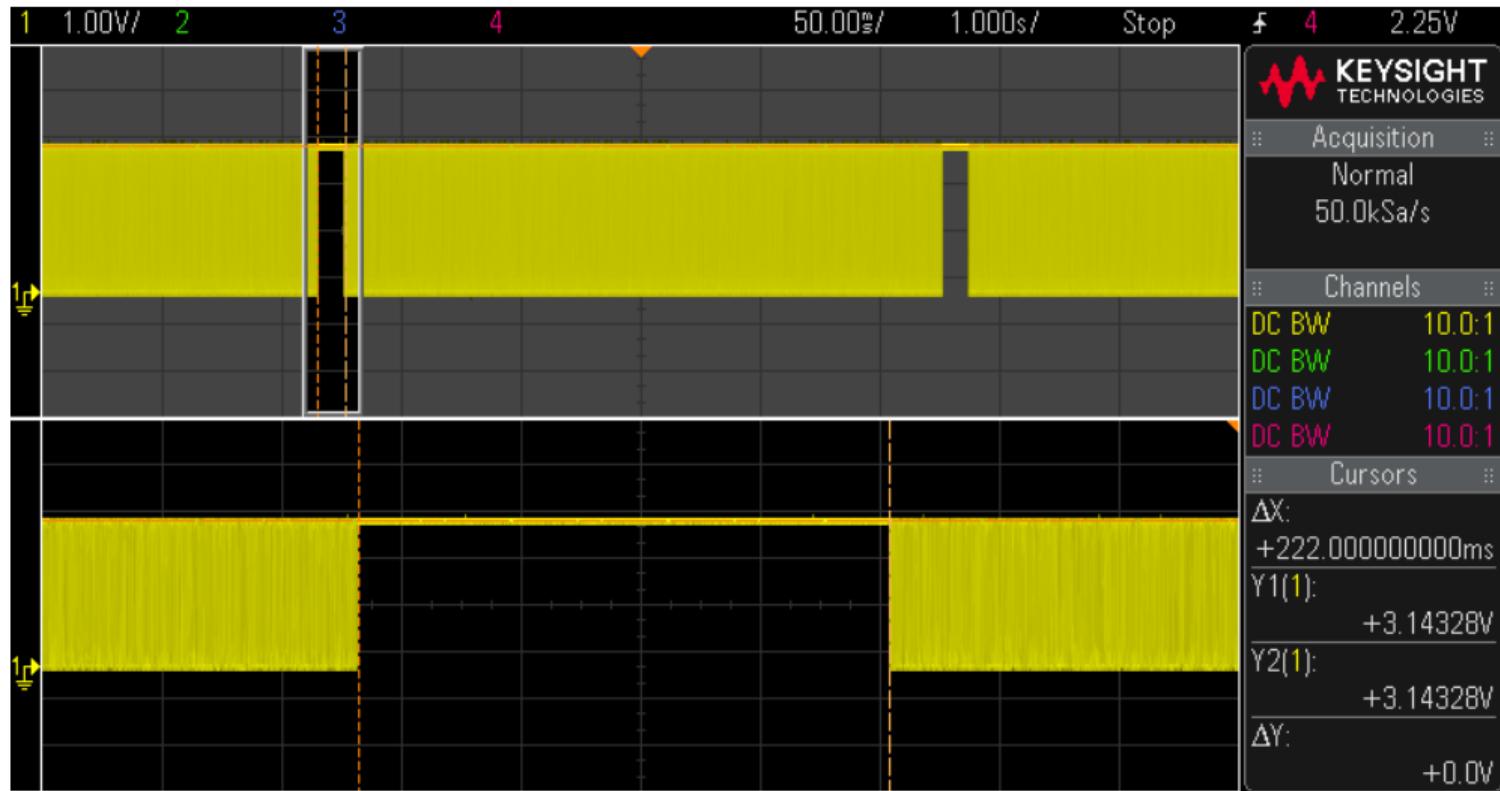
```
1 void blinkLedForever(int led) {  
2     for(;;) { // repeat forever  
3         digitalWrite(led, HIGH); // led on  
4         digitalWrite(led, LOW); // led off  
5     }  
6 }
```

```
1 void yellowTask(void * pvParameters) {
2     blinkLedForever(L_Y);
3 }
4
5 void setup() {
6     pinMode(L_Y, OUTPUT);
7     xTaskCreate(
8         yellowTask, // Function to implement the task
9         "Yellow task", // Name of the task
10        10000, // Stack size in words
11        NULL, // Task input parameter
12        1, // Priority of the task
13        NULL // Task handle.
14    );
15 }
```

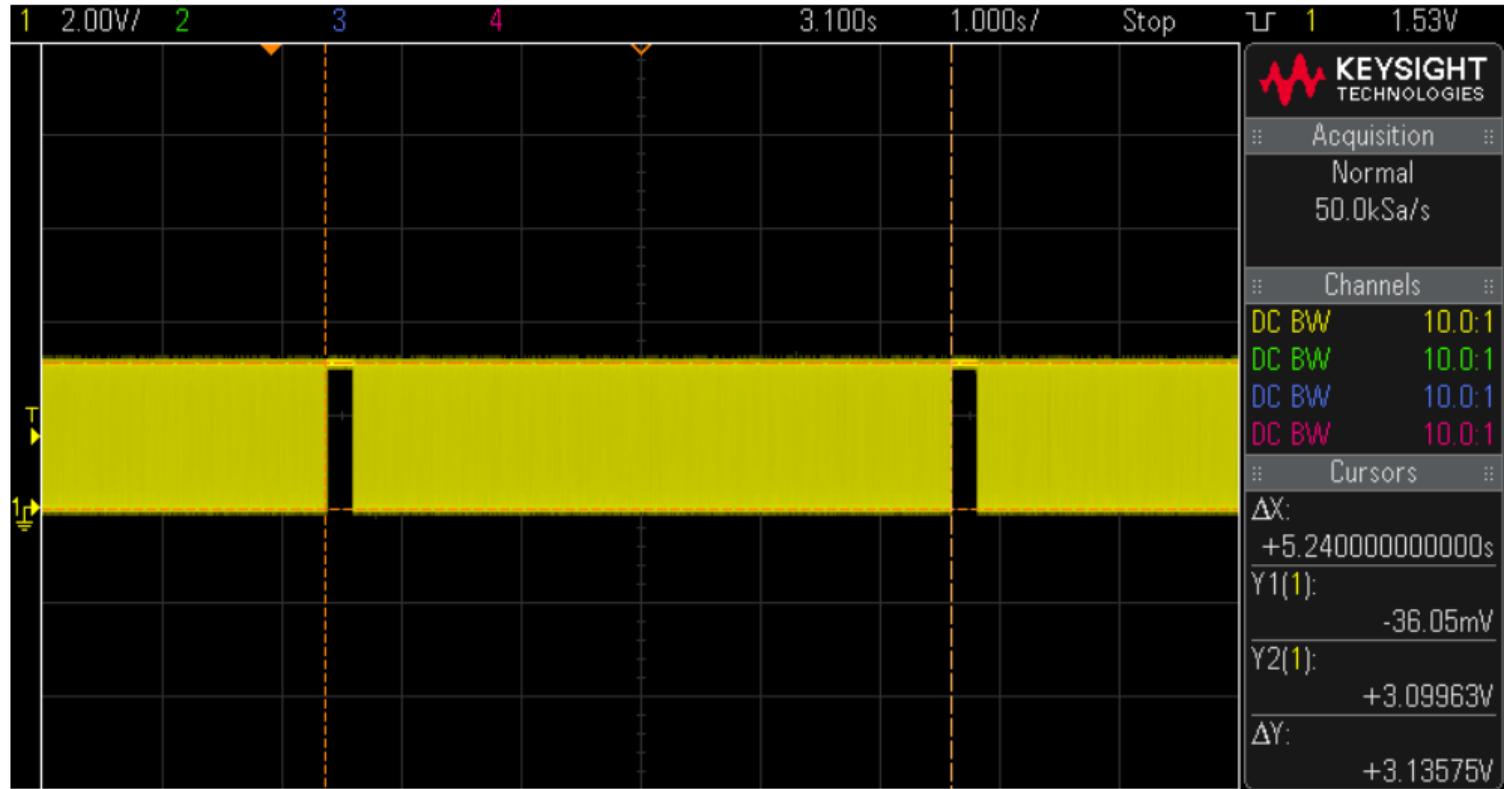
New task creation



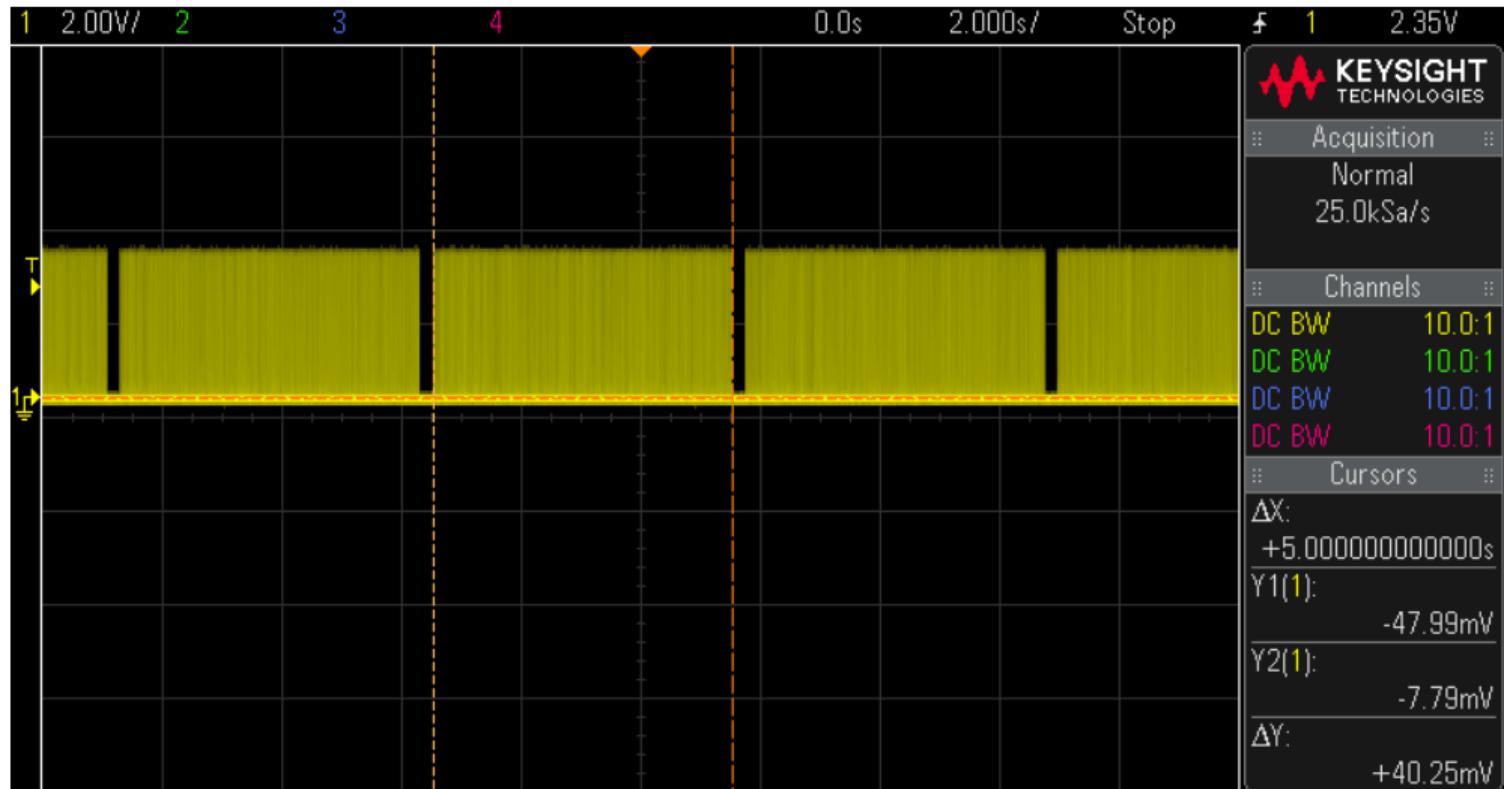
New task creation



New task creation



New task creation



New task creation

```
Terminal - COM4 - Lorris v933
Menu Terminal - COM4
Disconnect Pause Clear Text Send...  RTS  DTR

Rebooting...
ets Jun  8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x12 (SPI_FAST_FLASH_BOOT)
configip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:104
load:0x40078000,len:8896
load:0x40080400,len:5828
entry 0x400806ac

E (10218) task_wdt: Task watchdog got triggered. The following tasks did not reset the watchdog in time:
E (10218) task_wdt: - IDLE0 (CPU 0)
E (10218) task_wdt: Tasks currently running:
E (10218) task_wdt: CPU 0: newTask
E (10218) task_wdt: CPU 1: loopTask
E (10218) task_wdt: Aborting.
abort() was called at PC 0x400d356f on core 0

Backtrace: 0x4008b2e4:0x3ffbe170 0x4008b511:0x3ffbe190 0x400d356f:0x3ffbe1b0 0x400843d1:0x3ffbe1d0 0x40081093:0x3d

Rebooting...
ets Jun  8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x12 (SPI_FAST_FLASH_BOOT)
configip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:104
load:0x40078000,len:8896
load:0x40080400,len:5828
entry 0x400806ac
```

New task creation

Single core

- Scheduler is running on the same core.
- FreeRTOS has some watchdogs.
- If the scheduler (or task) does not trigger the watchdog on time, the watchdog resets the processor.
- Watchdog can be disabled (not recommended).

VS.

Multi core

- One task can occupy the whole core.
- There must be a reserved space for the scheduler at one core.
- Keep in mind the watchdogs.

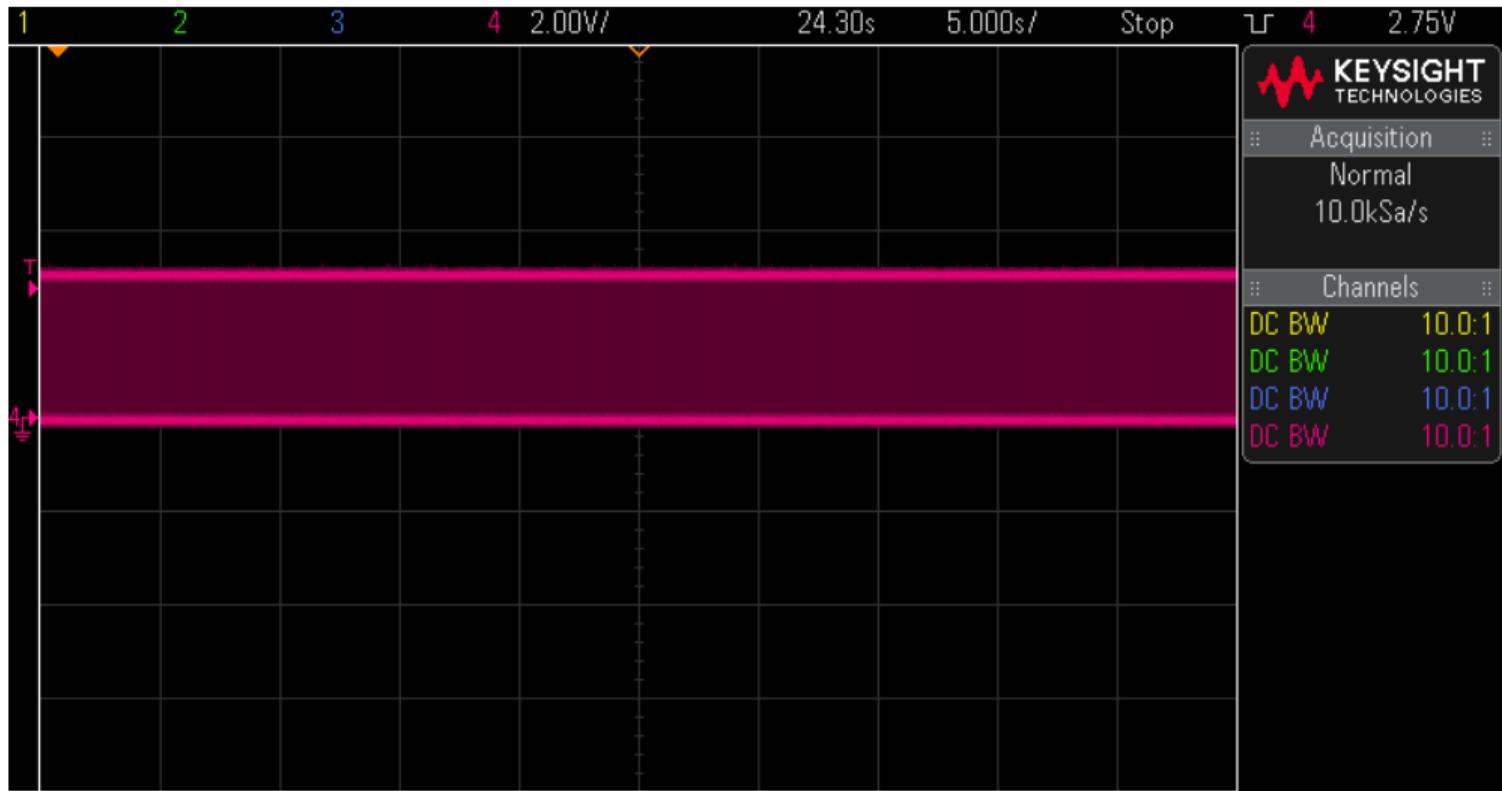
A task with waiting

```
1 void blinkLedForeverWithDelay(int led, uint32_t ticks) {  
2     for(;;) {  
3         digitalWrite(led, HIGH);  
4         vTaskDelay(ticks);  
5         digitalWrite(led, LOW);  
6         vTaskDelay(ticks);  
7     }  
8 }
```

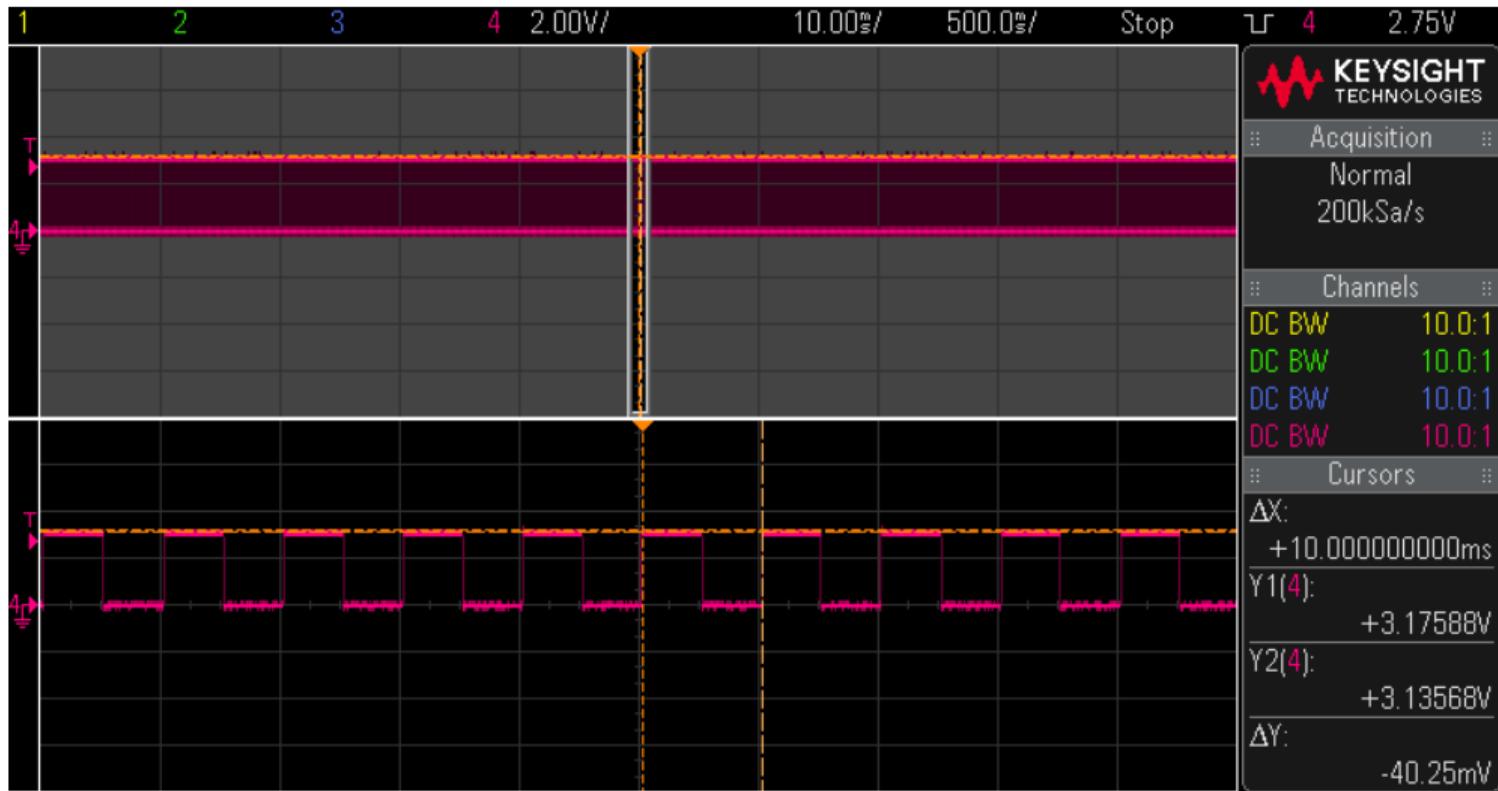
A task with waiting

```
1 void redTask(void * pvParameters) {
2     blinkLedForeverWithDelay(L_R, 5 /*ms*/);
3 }
4
5 void setup() {
6     esp_task_wdt_deinit();
7     pinMode(L_R, OUTPUT);
8     xTaskCreate(redTask, "redTask", 10000, NULL, 1, NULL);
9 }
```

A task with waiting



A task with waiting



The very very common mistakes

Do not forget to reserve a time for scheduler and context switching!

- Your tasks are interrupted when the scheduler is running.
- Context switching consumes time, too.

Do not forget the watch dogs!

- The CPU resets easily hide themselves behind the periodic tasks.

Do not forget the main task!

- If the main() function is still running, it is a task, too.

Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

New task creation

Task properties

Multiple tasks

Jobs

Timers

Sharing data among threads

Critical sections

Queues

Task management

Changes from the previous example

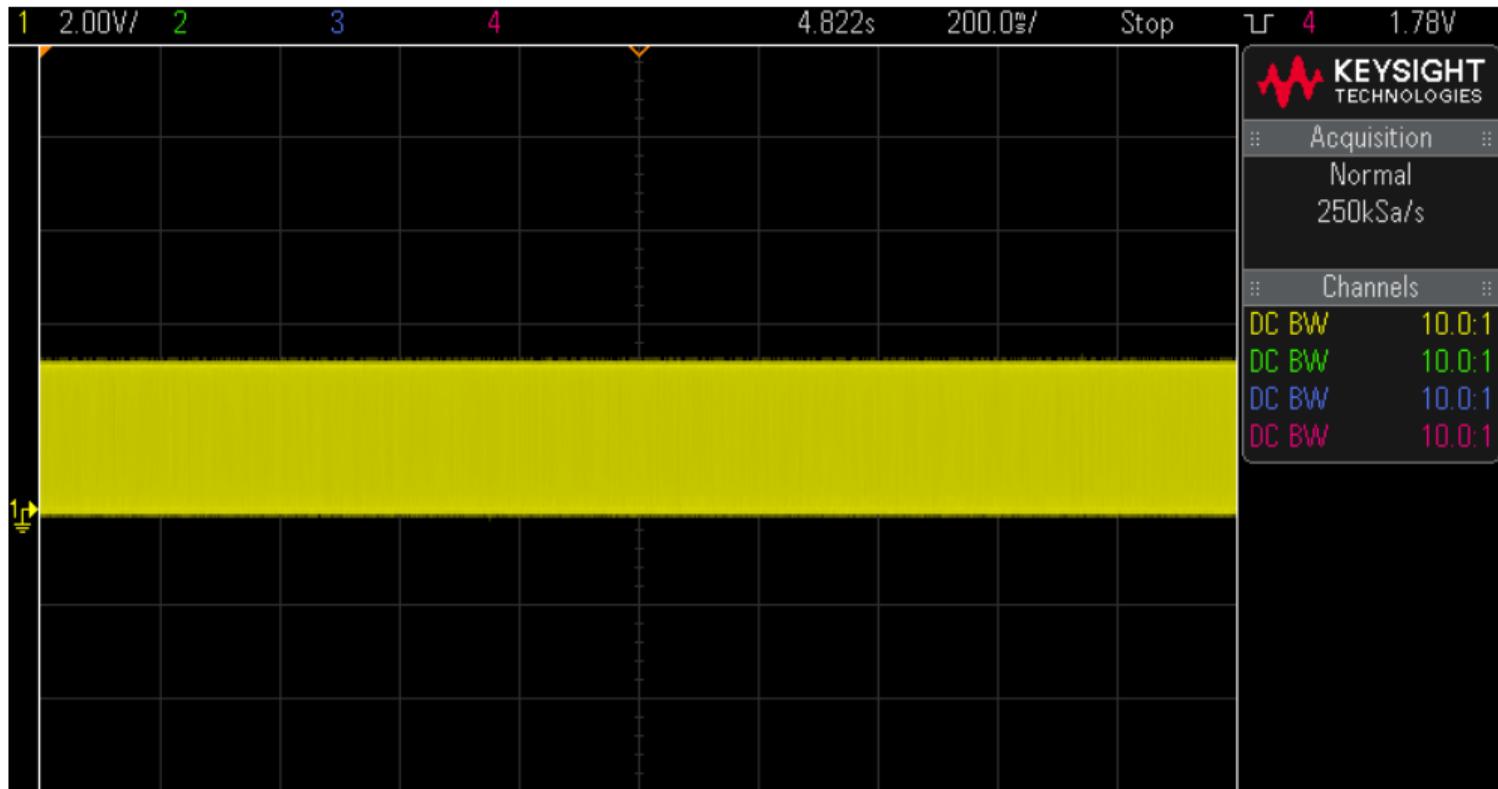
- Yellow task is pinned to Core 1.
- The scheduler is running on the Core 0.
- The task watch dog is disabled.

Remember previous functions

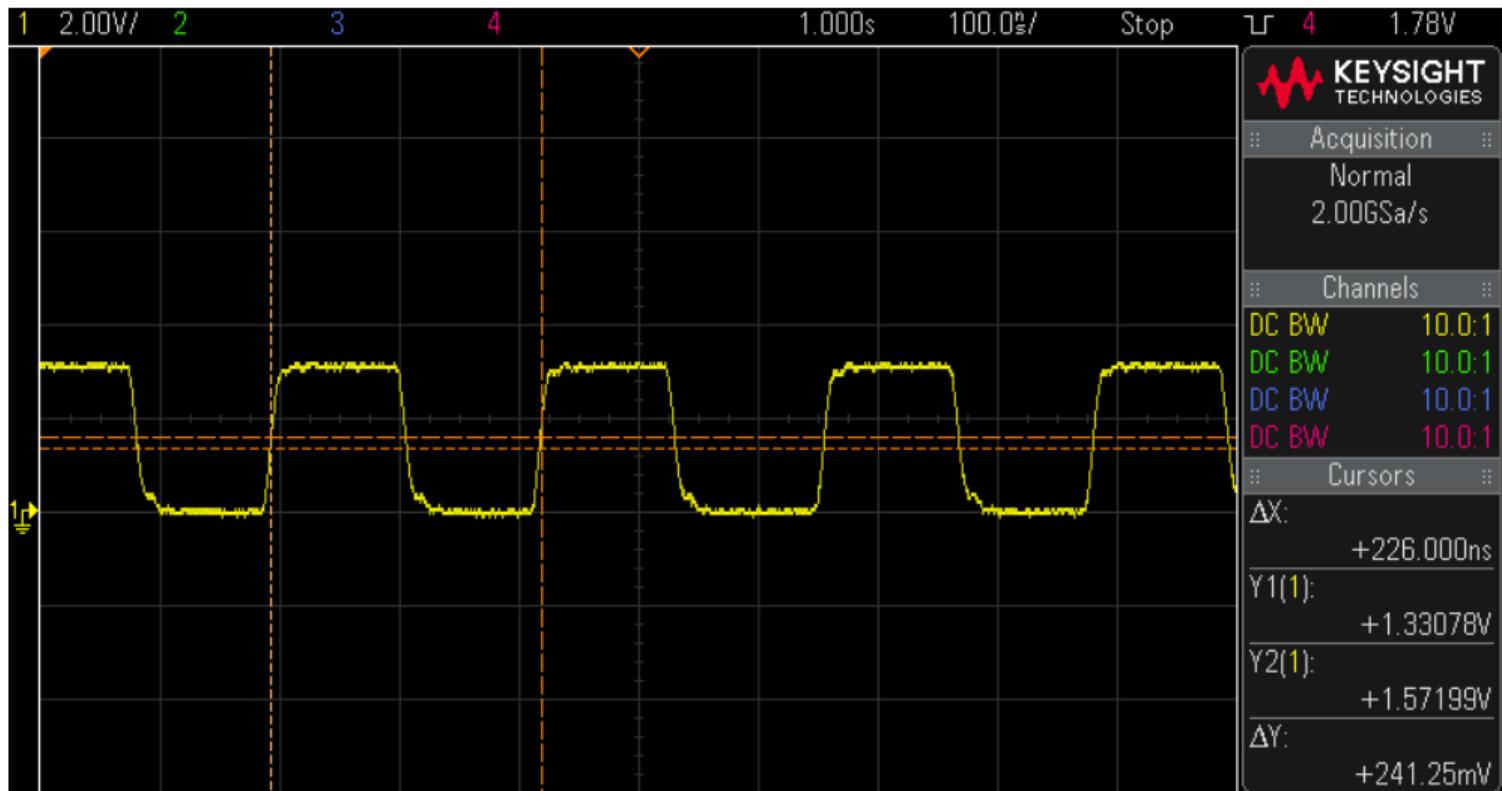
```
1 void blinkLedForever(int led) {
2     for(;;) { // repeat forever
3         digitalWrite(led, HIGH); // led on
4         digitalWrite(led, LOW); // led off
5     }
6 }
7
8 void yellowTask(void * pvParameters) {
9     blinkLedForever(L_Y);
10 }
```

```
1 void setup() {
2     esp_task_wdt_deinit();
3     pinMode(L_Y, OUTPUT);
4
5     xTaskCreatePinnedToCore(
6         yellowTask, // Function to implement the task
7         "yellowTask", // Name of the task
8         10000, // Stack size in words
9         NULL, // Task input parameter
10        2, // Priority of the task
11        NULL, // Task handle.
12        1 // CPU core ID
13    );
14 }
```

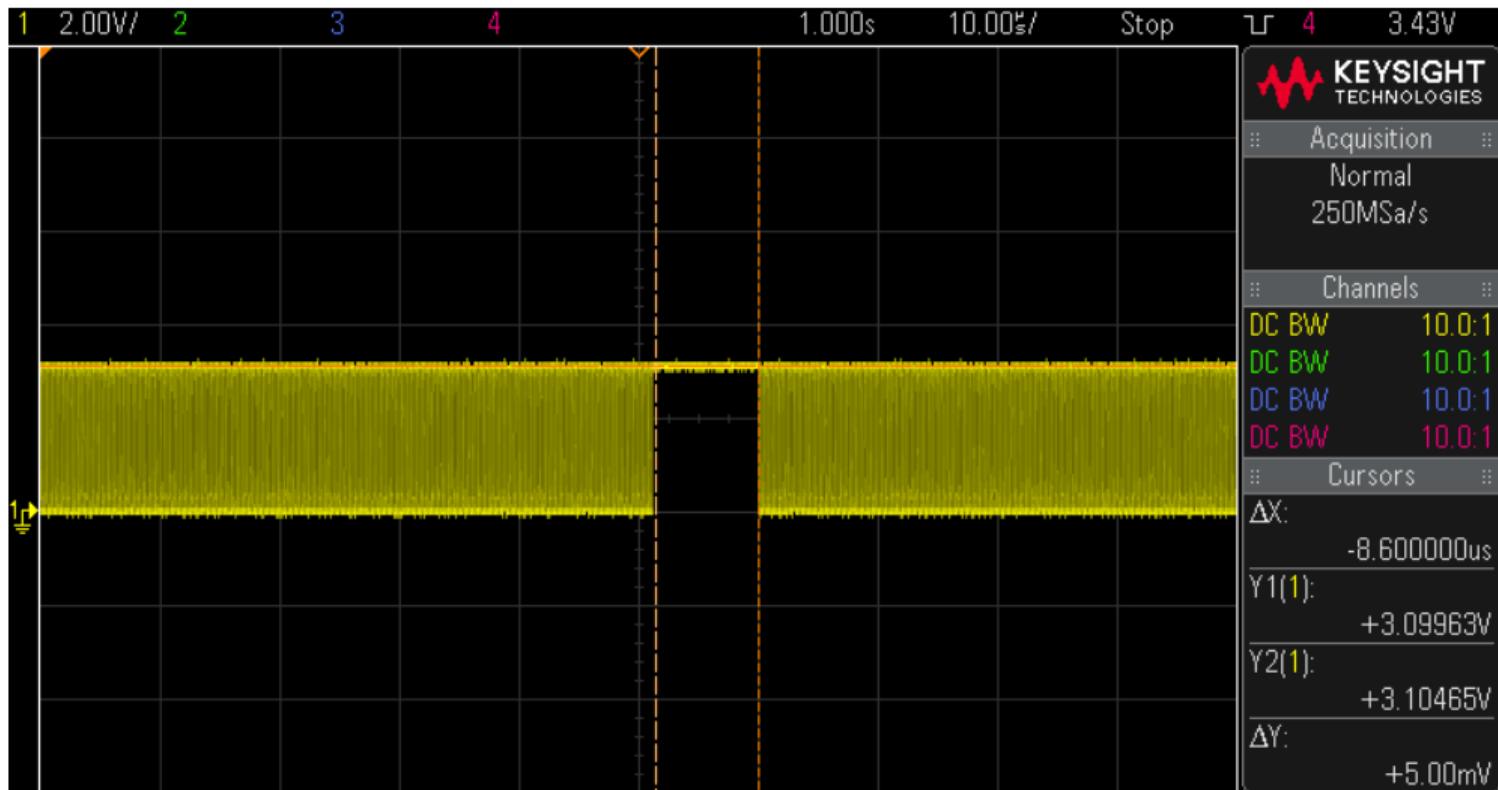
Task management



Task management



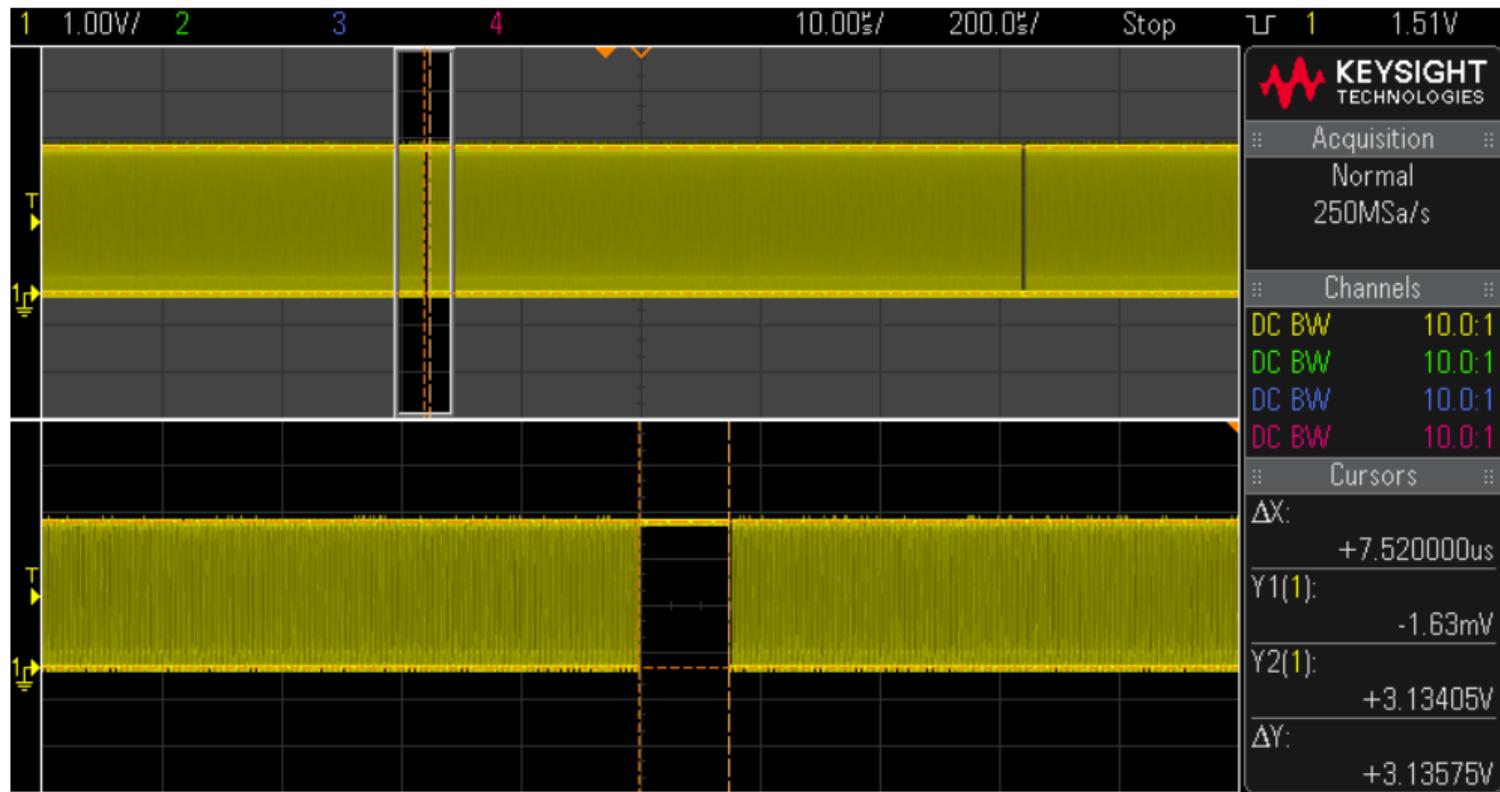
Task management



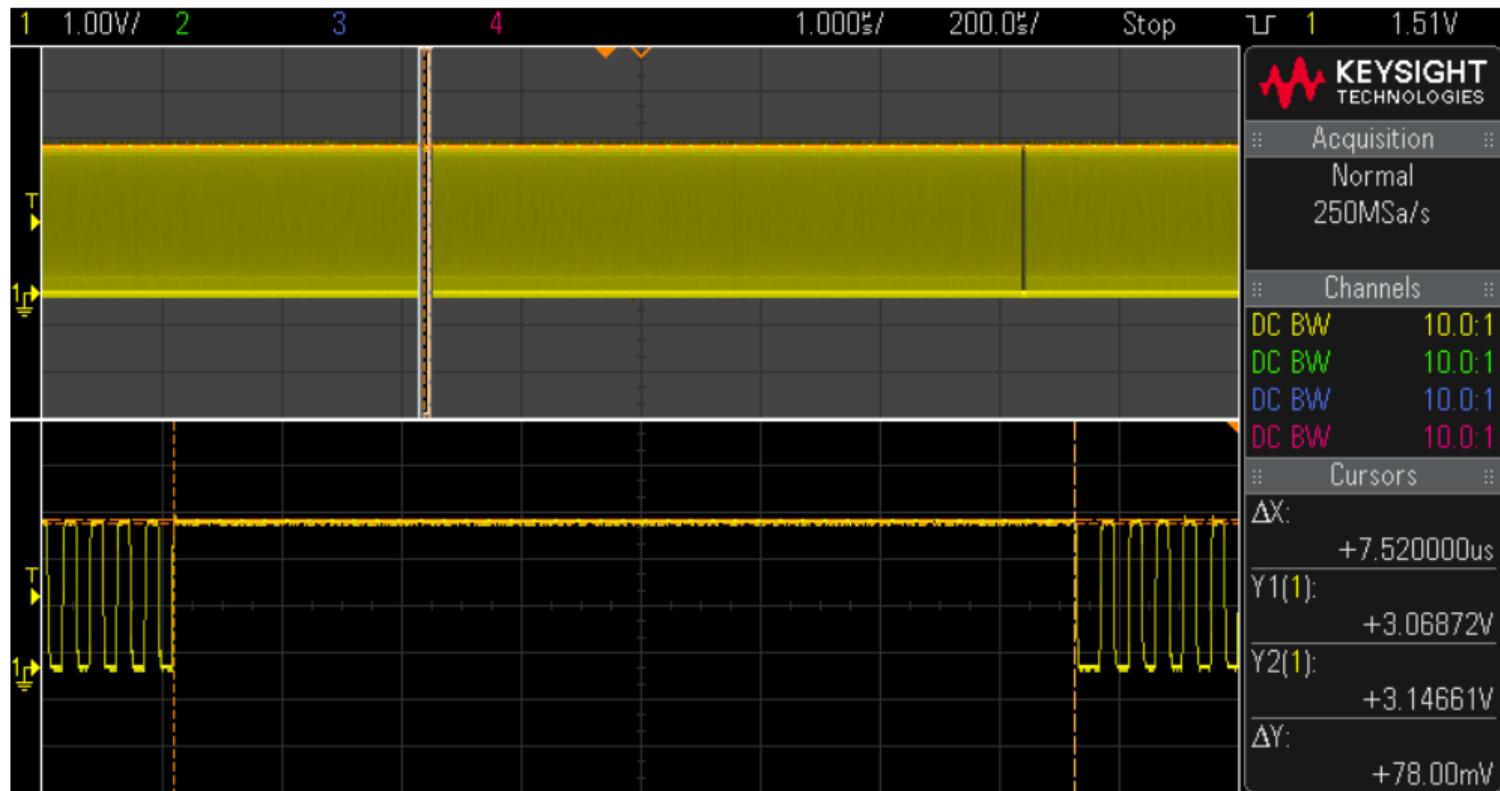
Task management



Task management



Task management



Task management

Properties of the task

- The task is interrupted by the scheduler and by defined SW or HW interrupts.
- If we want to run an uninterrupted thread on the CPU core,
do not use RTOS on this CPU core.

Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

New task creation

Task properties

Multiple tasks

Jobs

Timers

Sharing data among threads

Critical sections

Queues

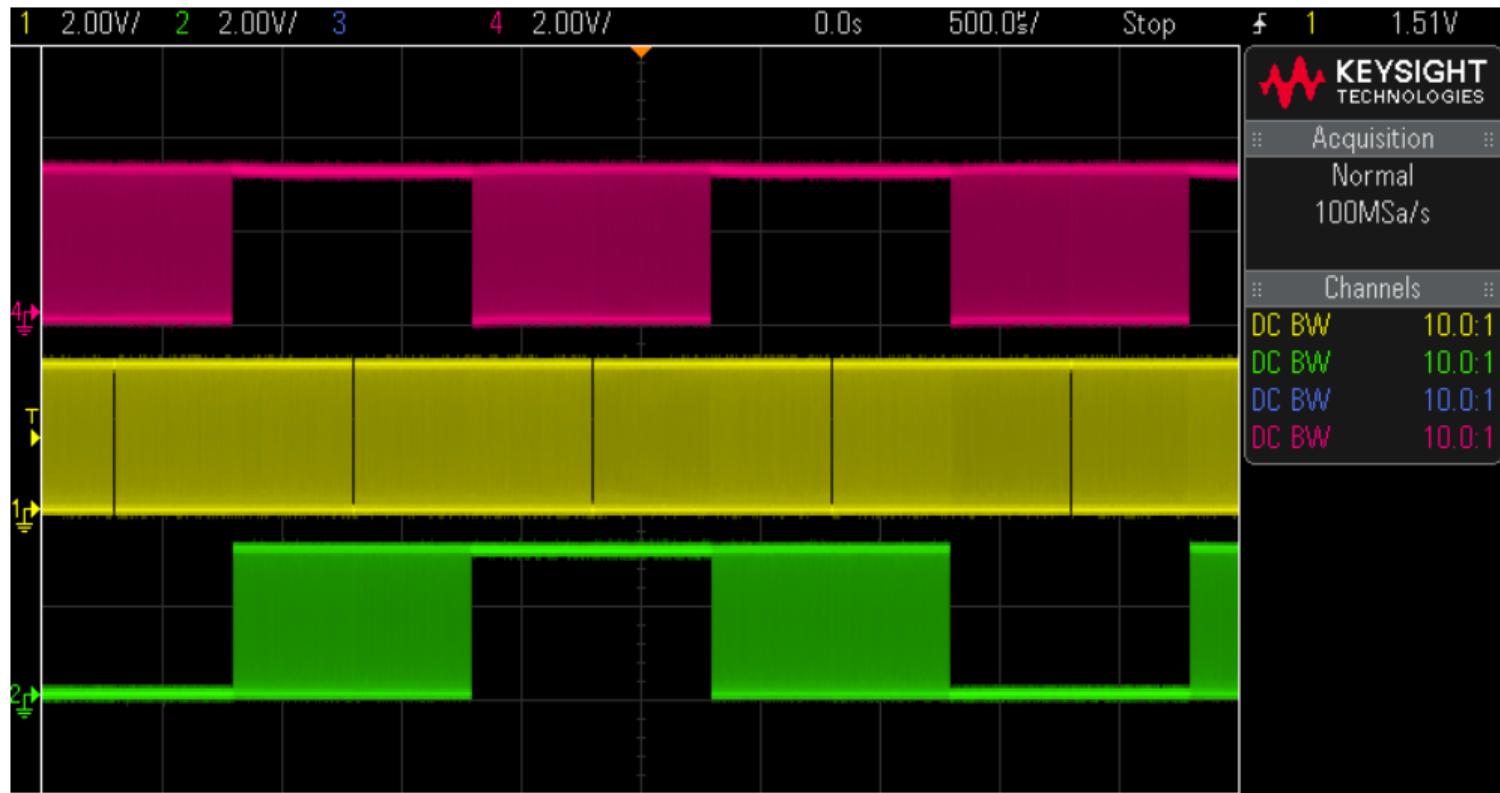
Multiple tasks

Warning: the watchdog will reset the CPU in this example

```
1 void redTask(void * pvParameters) {
2     blinkLedForever(L_R);
3 }
4
5 void yellowTask(void * pvParameters) {
6     blinkLedForever(L_Y);
7 }
8
9 void greenTask(void * pvParameters) {
10    blinkLedForever(L_G);
11 }
```

```
1 void setup() {
2     esp_task_wdt_deinit();
3     pinMode(L_R, OUTPUT);
4     pinMode(L_Y, OUTPUT);
5     pinMode(L_G, OUTPUT);
6
7     xTaskCreate(redTask, "redTask", 10000, NULL, 1, NULL);
8     xTaskCreate(yellowTask, "yellowTask", 10000, NULL, 1, NULL);
9     blinkLedForever(L_G);
10 }
```

Multiple tasks



Now let's change the task priorities

```
1 (...)  
2 xTaskCreate(redTask, "redTask", 10000, NULL, 2, NULL);  
3 xTaskCreate(yellowTask,"yellowTask",10000,NULL,3,NULL);  
4 blinkLedForever(L_G);  
5 (...)
```

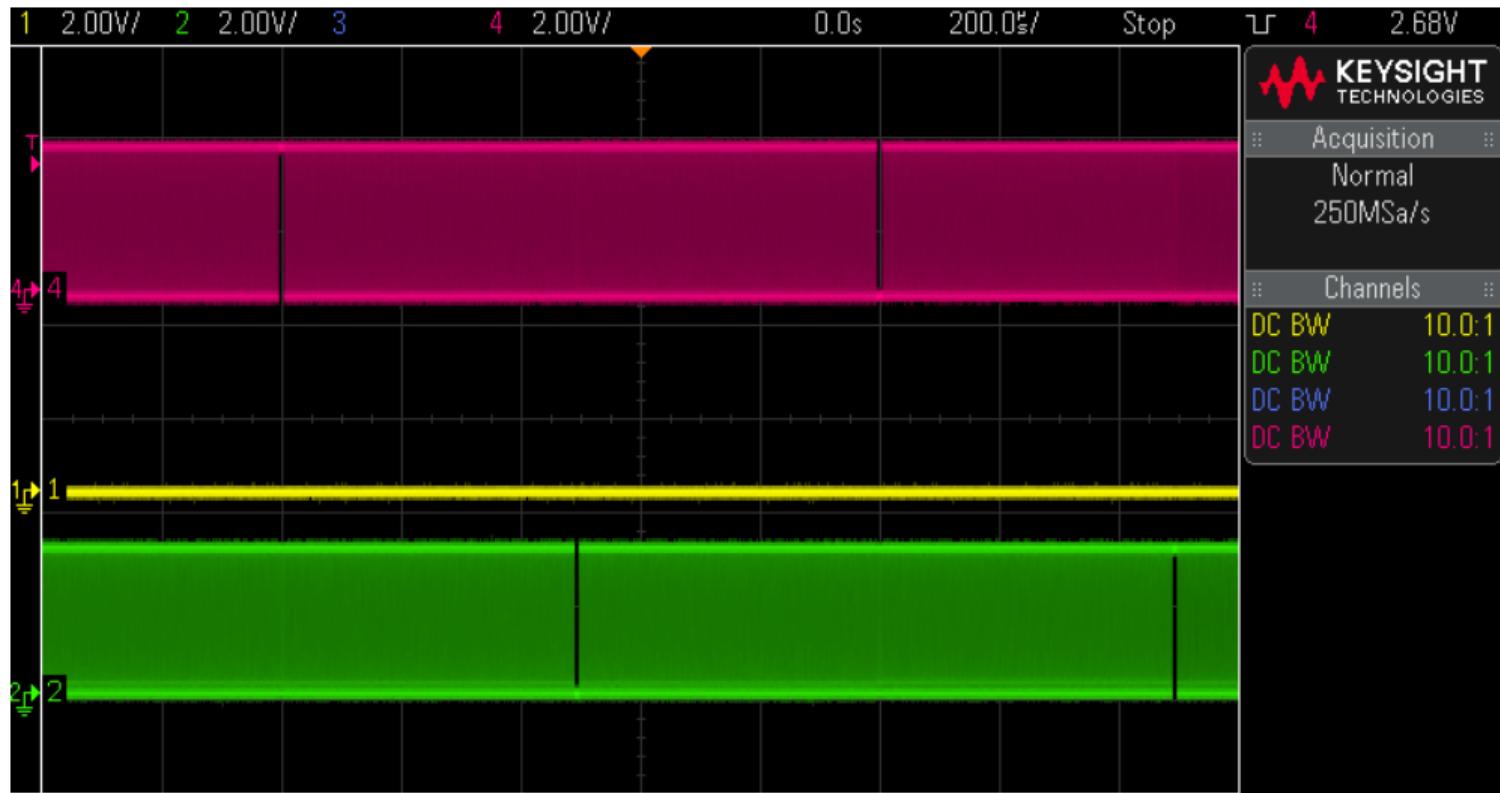
Multiple tasks



Now let's change the task priorities

```
1 (...)  
2 xTaskCreate(redTask, "redTask", 10000, NULL, 0, NULL);  
3 xTaskCreate(yellowTask,"yellowTask",10000,NULL,0,NULL);  
4 blinkLedForever(L_G);  
5 (...)
```

Multiple tasks



Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

New task creation

Task properties

Multiple tasks

Jobs

Timers

Sharing data among threads

Critical sections

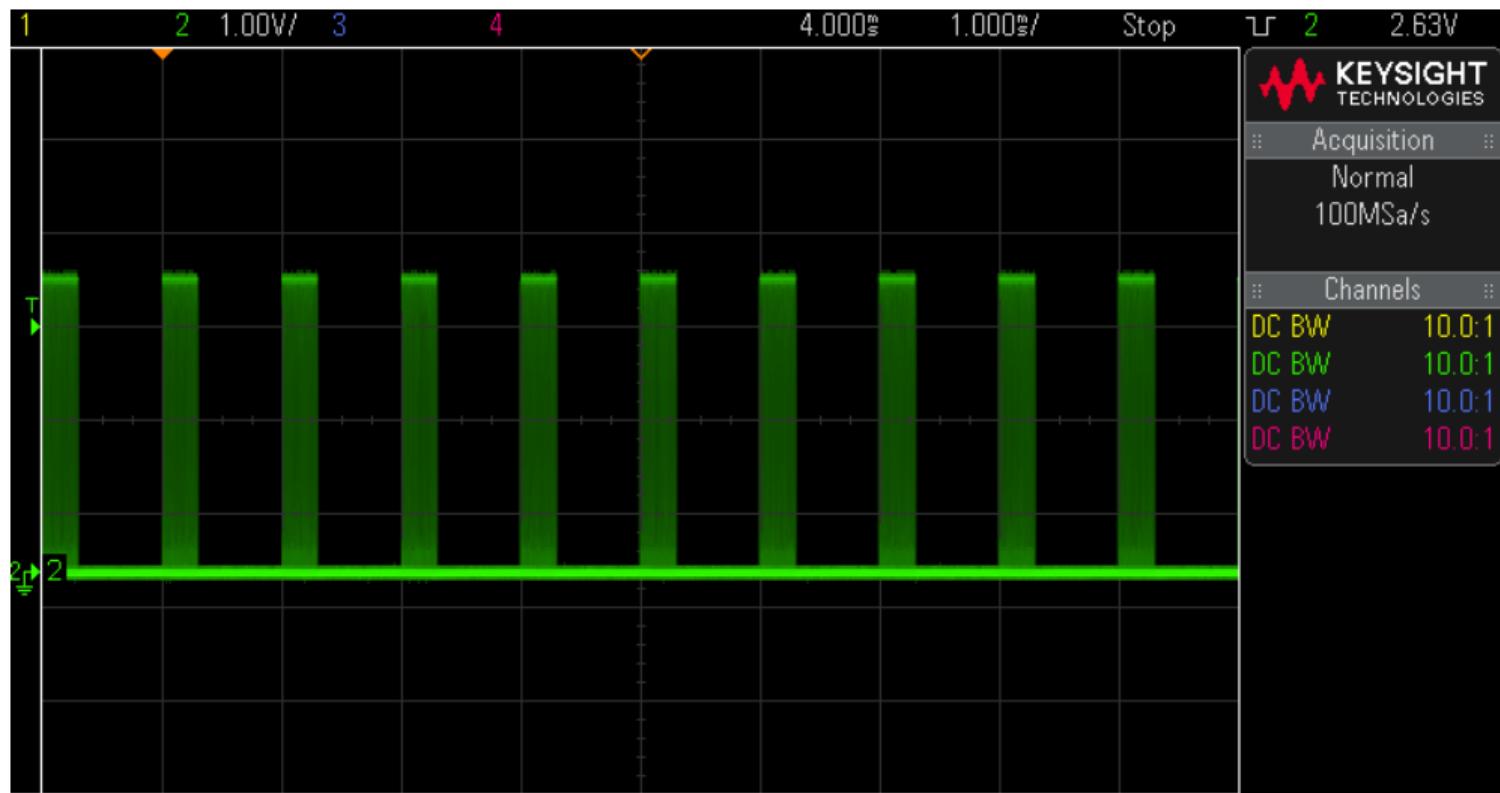
Queues

Tasks and jobs

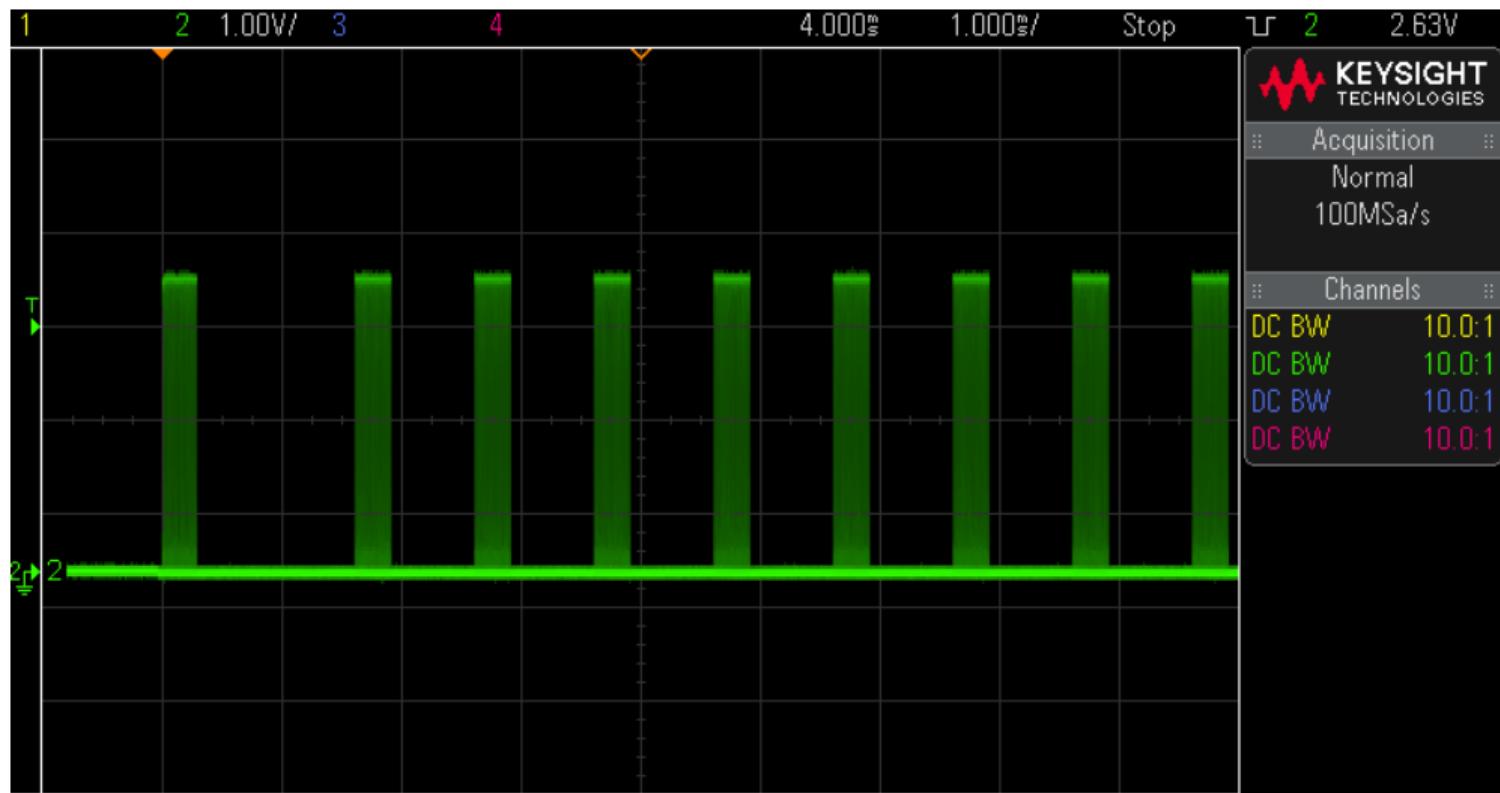
Imagine a function

```
1 void blinkLedForTime(uint8_t led,
2                         uint64_t run_us,
3                         uint64_t wait_ms) {
4     for(;;) {
5         // Blinking for specified time in microseconds
6         for(uint64_t t = micros(); t + run_us > micros();) {
7             digitalWrite(led, HIGH);
8             digitalWrite(led, LOW);
9         }
10        vTaskDelay(wait_ms);
11    }
12 }
```

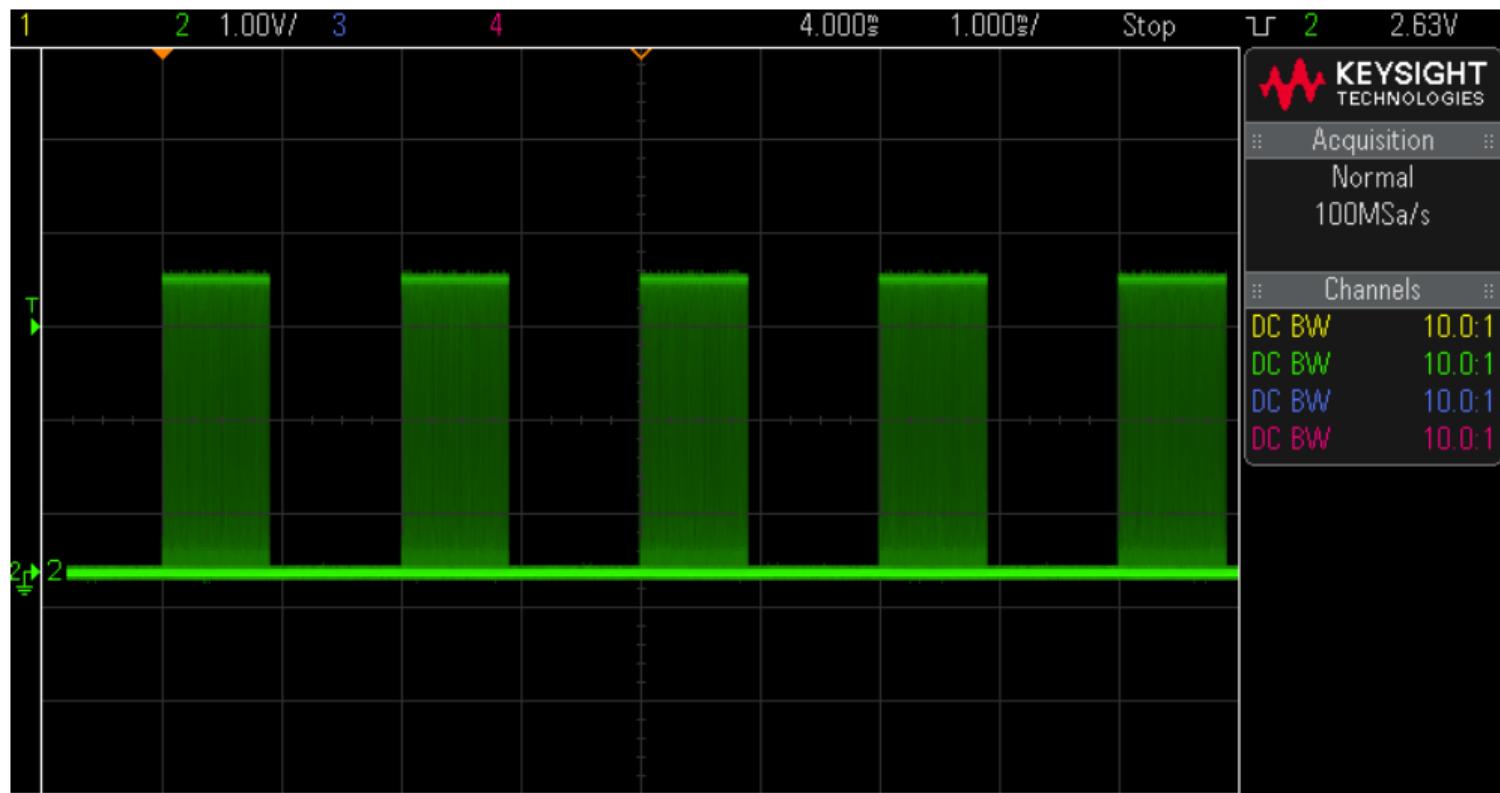
```
1 blinkLedForTime(L_G, 300, 1);
```



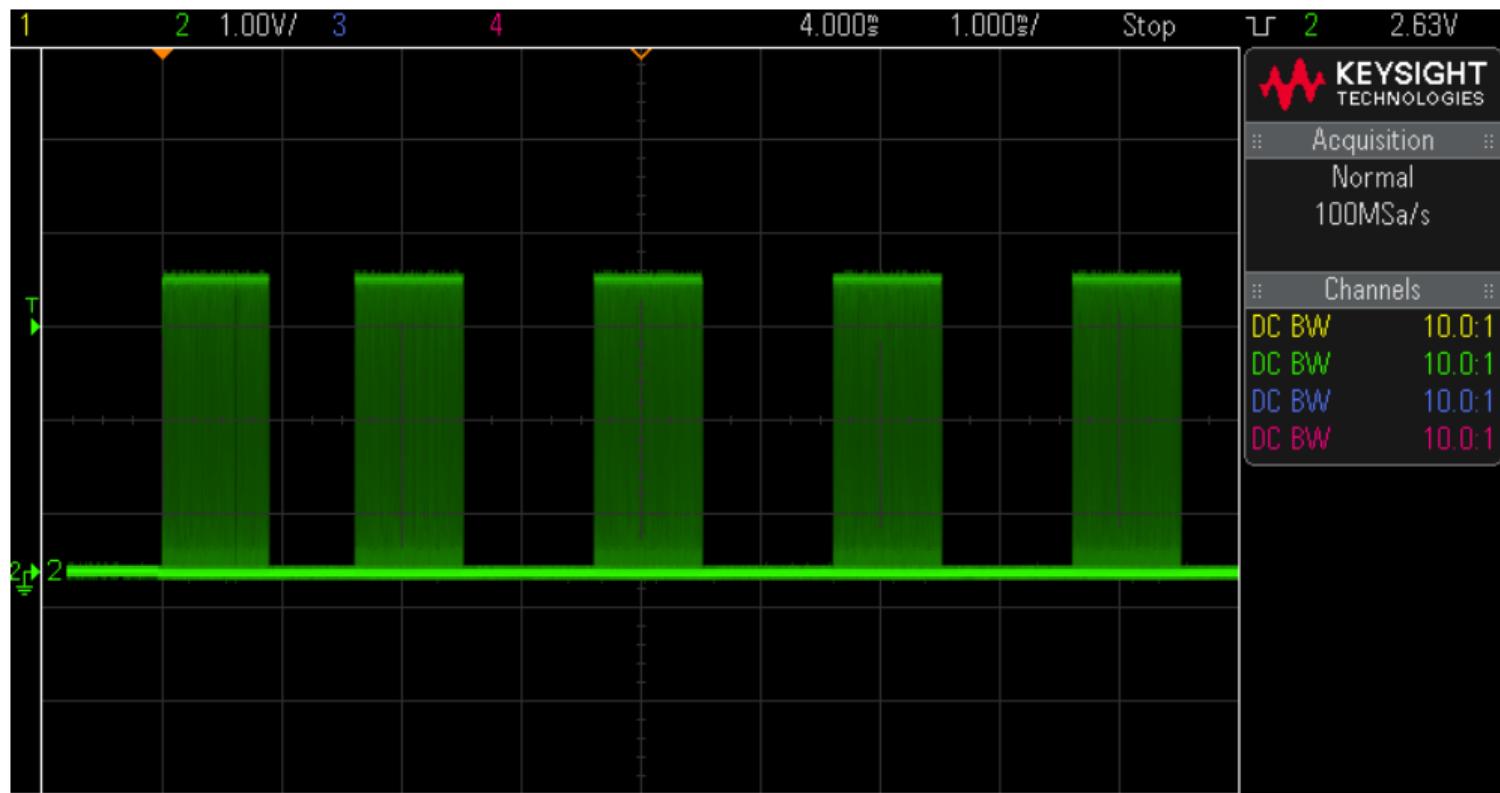
```
1 blinkLedForTime(L_G, 300, 1);
```



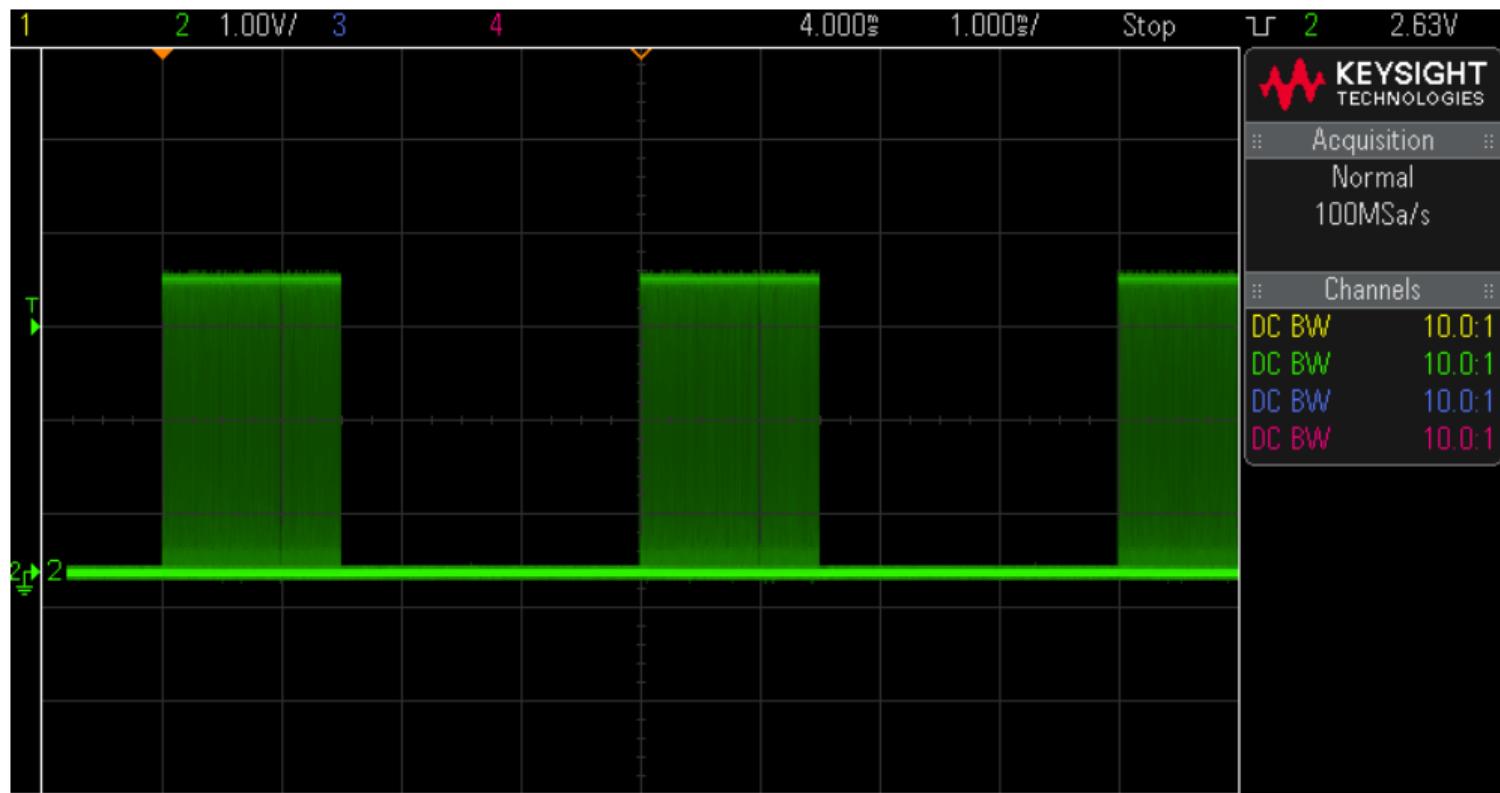
```
1 blinkLedForTime(L_G, 900, 1);
```



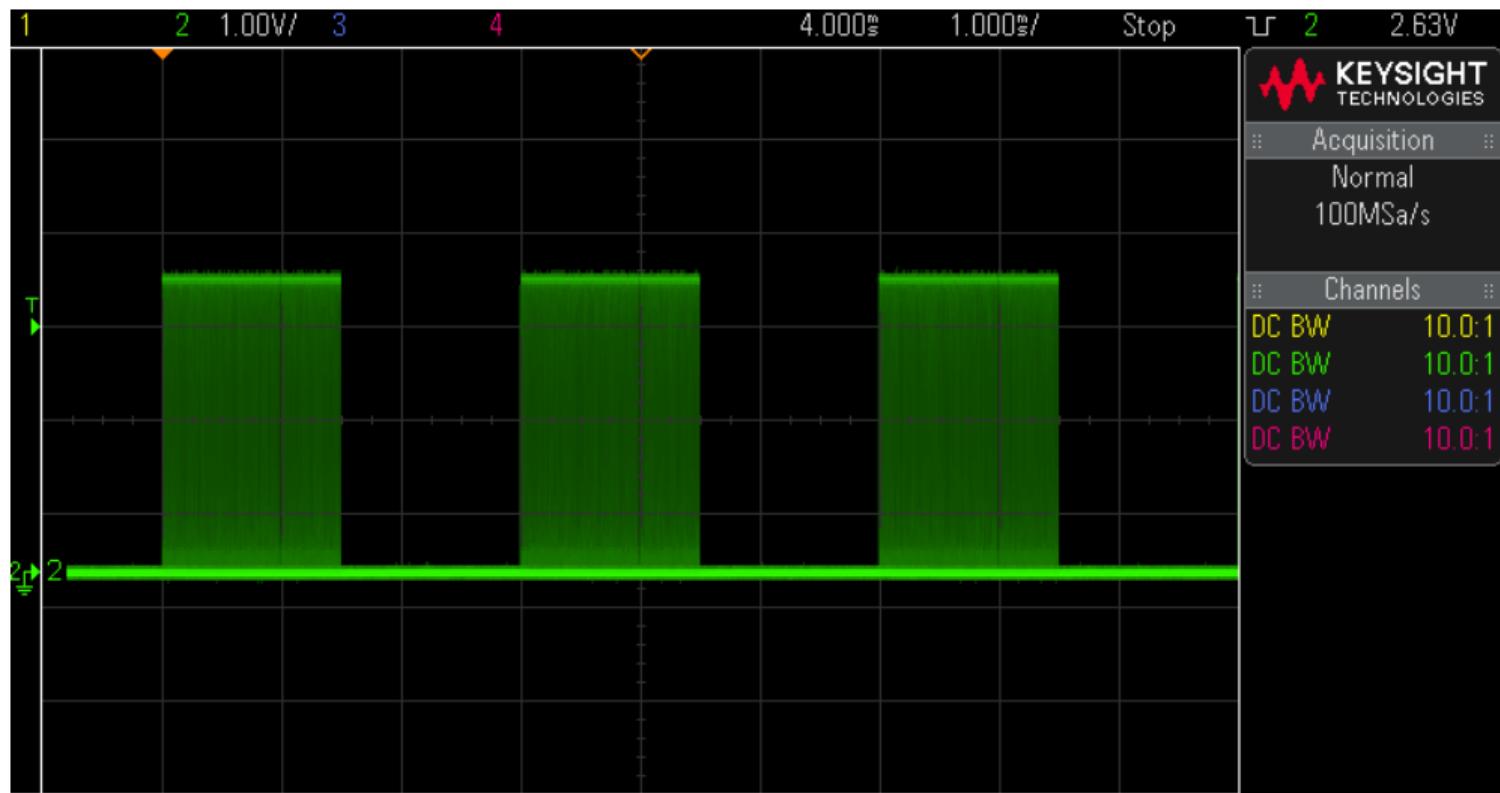
```
1 blinkLedForTime(L_G, 900, 1);
```



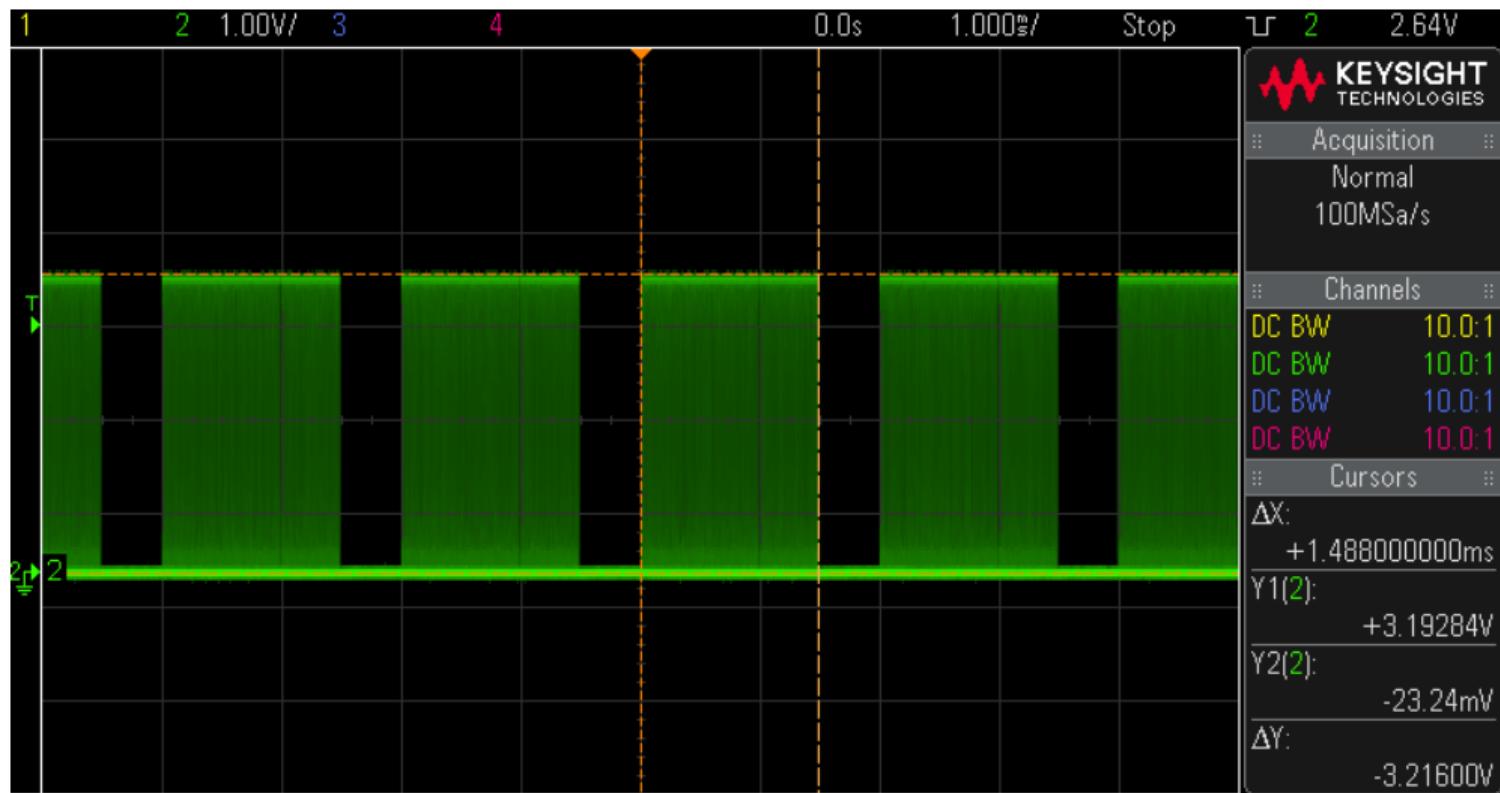
```
1 blinkLedForTime(L_G, 1500, 2);
```



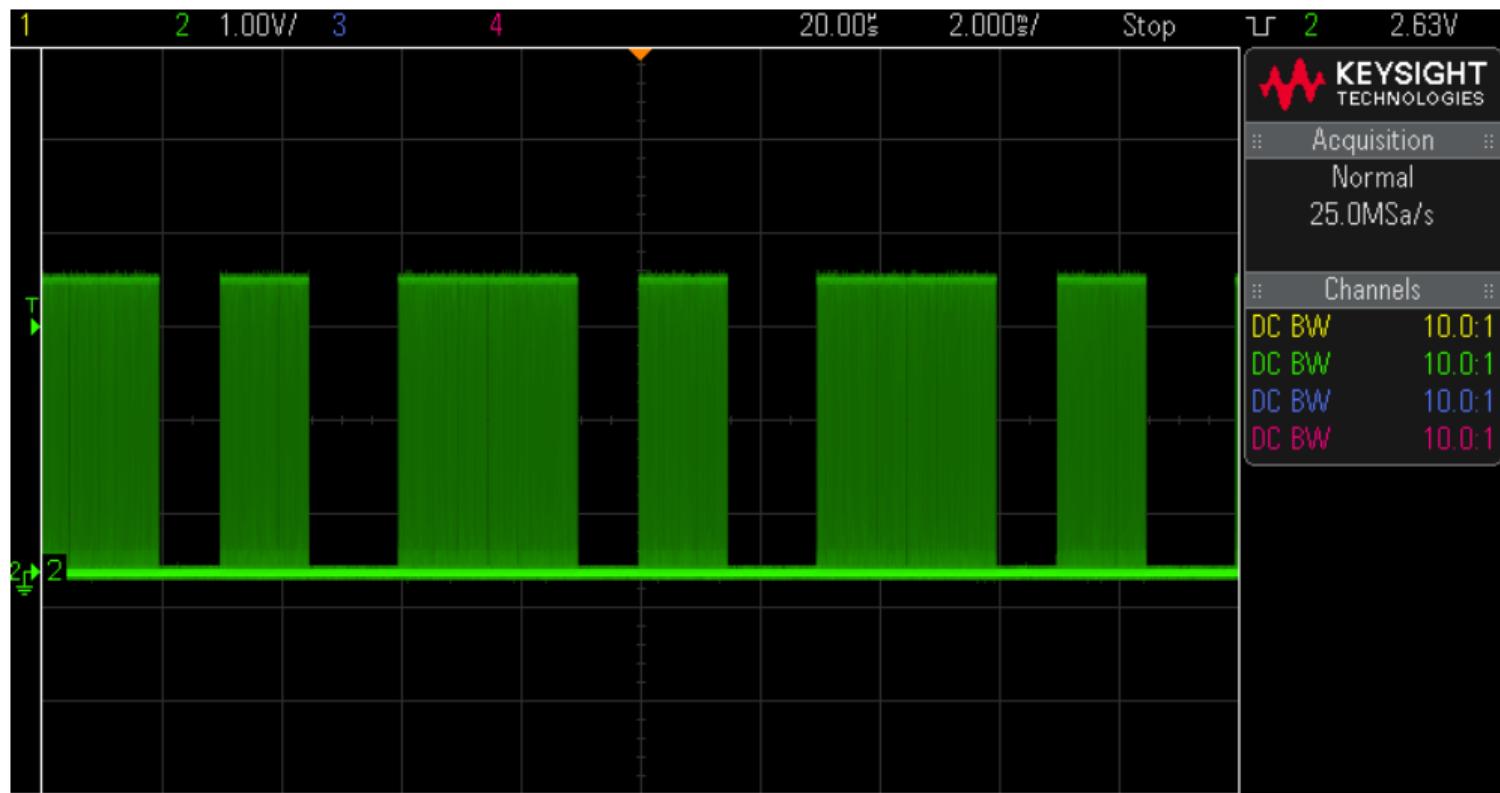
```
1 blinkLedForTime(L_G, 1500, 1);
```



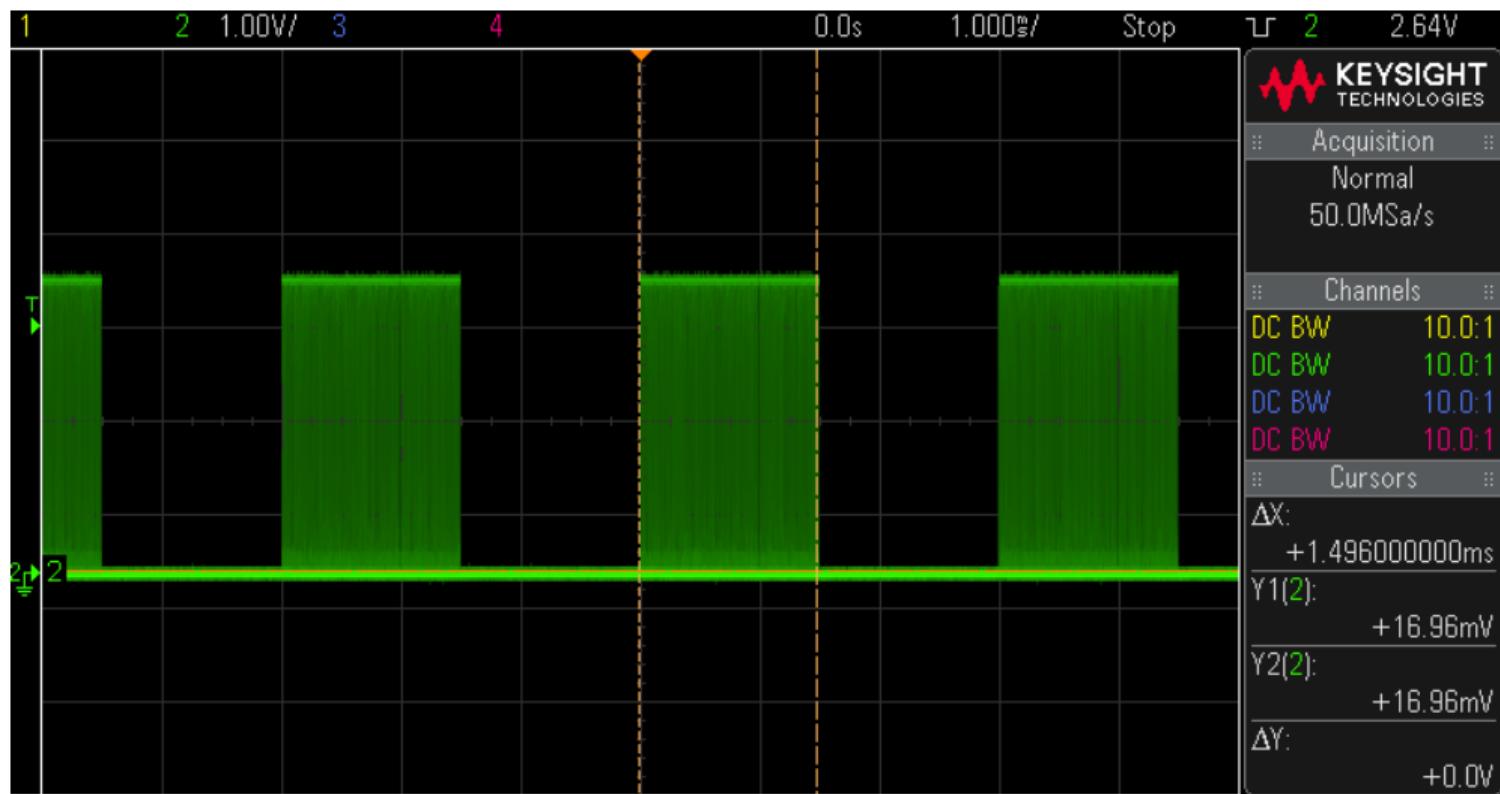
```
1 blinkLedForTime(L_G, 1488, 1);
```



```
1 blinkLedForTime(L_G, 1489, 1);
```



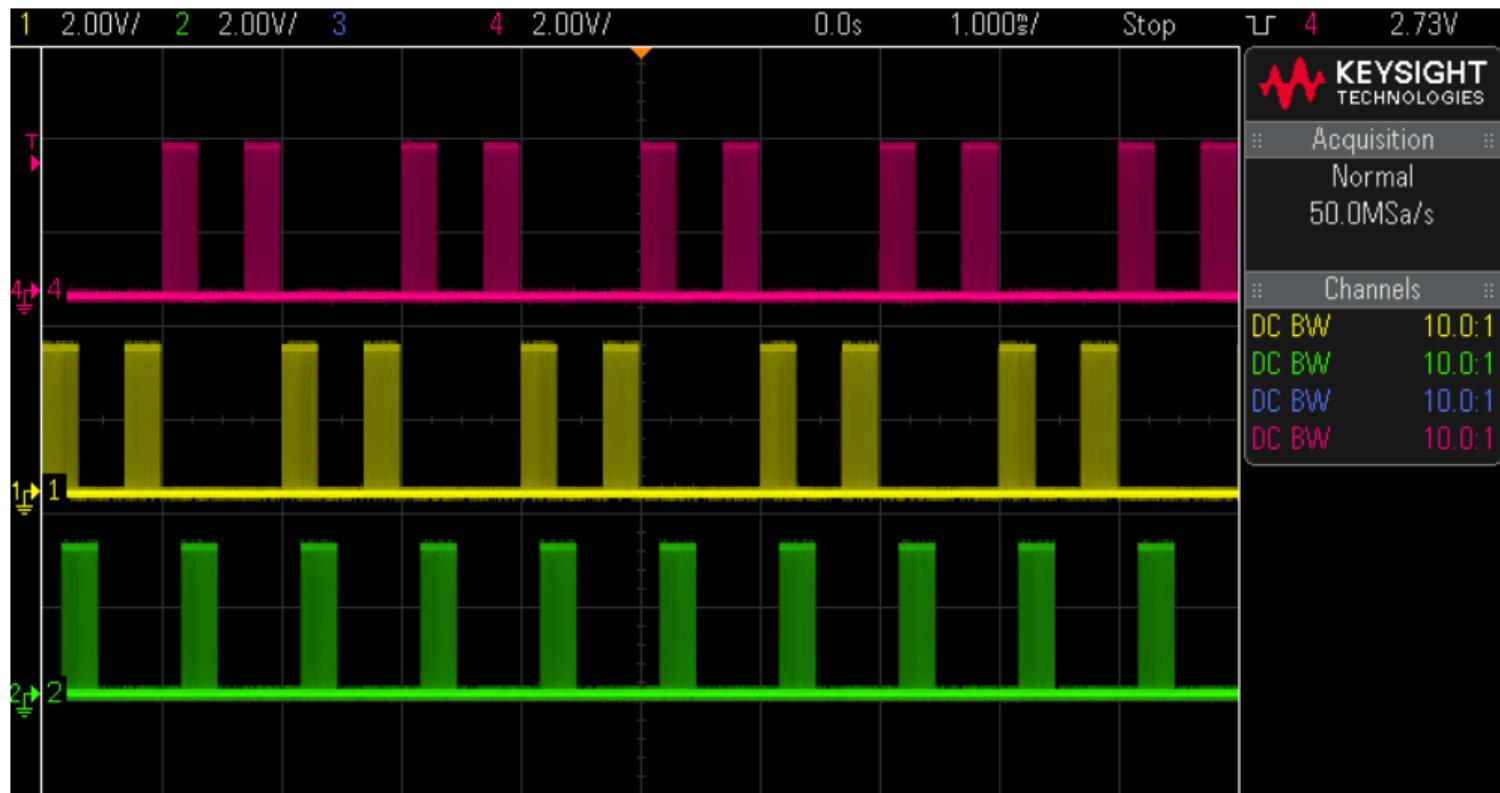
```
1 blinkLedForTime(L_G, 1490, 1);
```



3 periodic tasks on 2 CPU cores

```
1 void redTask(void * pvParameters) {
2     blinkLedForTime(L_R, 300, 1);
3 }
4
5 void yellowTask(void * pvParameters) {
6     blinkLedForTime(L_Y, 300, 1);
7 }
8
9 void setup() {
10 (...) 
11     xTaskCreate(redTask, "redTask", 10000, NULL, 3, NULL, 0);
12     xTaskCreate(yellowTask, "yellowTask", 10000, NULL, 2, NULL, 0);
13     blinkLedForTime(L_G, 300, 1);
14 }
```

3 periodic tasks on 2 CPU cores



Task control functions

- `xTaskCreate(...)`
- `xTaskCreatePinnedToCore(...)`
- `vTaskDelete(...)`
- `vTaskDelay(...)`
- `xTaskDelayUntil(...)`
- `vTaskSuspend(...)`
- `vTaskResume(...)`
- `taskYIELD()`
- ...

Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

New task creation

Task properties

Multiple tasks

Jobs

Timers

Sharing data among threads

Critical sections

Queues

Timers

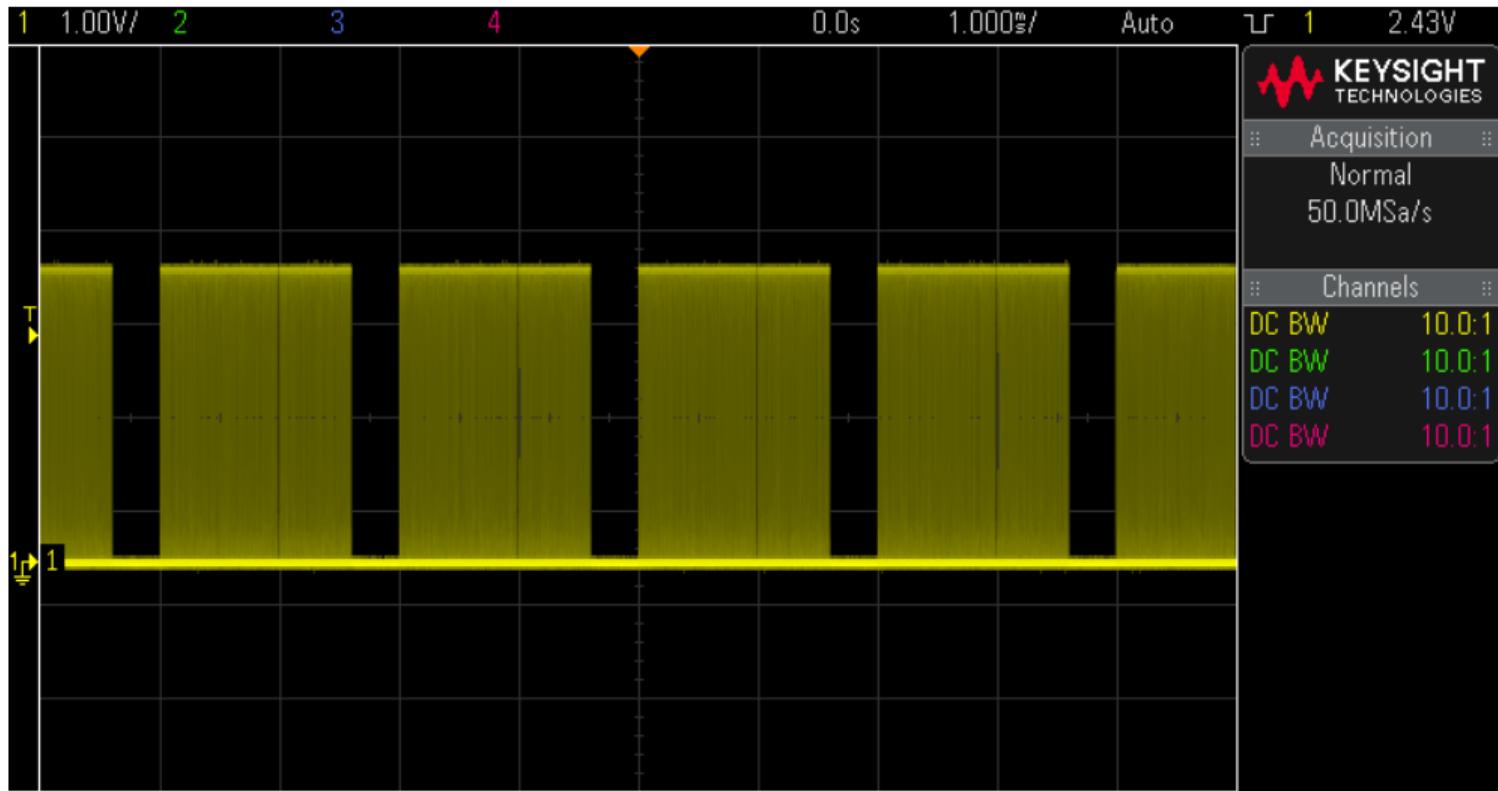
- Good for periodic tasks
- Limited number of hardware timers
- <https://www.freertos.org/RTOS-software-timer.html>

```
1 TimerHandle_t xTimerCreate(
2     const char * const pcTimerName,
3     const TickType_t xTimerPeriod,
4     const UBaseType_t uxAutoReload,
5     void * const pvTimerID,
6     TimerCallbackFunction_t pxCallbackFunction
7 );
```

Timers

```
1 void vTimerCallback(TimerHandle_t xTimer) {
2     blinkLedOnce(L_Y, 1600);
3 }
4
5 void setup() {
6     pinMode(L_Y, OUTPUT);
7     TimerHandle_t xTimer = xTimerCreate(
8         "Timer1", 2, pdTRUE, ( void * ) 0, vTimerCallback
9     );
10    xTimerStart(xTimer, 0);
11 }
```

Timers



Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

- New task creation

- Task properties

- Multiple tasks

- Jobs

- Timers

Sharing data among threads

- Critical sections

- Queues

Imagine a function

```
1 void blinkLedForTimeCritical(
2     uint8_t led, uint64_t run_us,
3     uint64_t wait_ms, portMUX_TYPE *myMutex
4 ) {
5     for(;;) {
6 //     portENTER_CRITICAL(myMutex);
7         blinkLedOnce(led, run_us);
8 //     portEXIT_CRITICAL(myMutex);
9         vTaskDelay(wait_ms);
10    }
11 }
```

Imagine a function

```
1 void blinkLedForTimeCritical(
2     uint8_t led, uint64_t run_us,
3     uint64_t wait_ms, portMUX_TYPE *myMutex
4 ) {
5     for(;;) {
6         portENTER_CRITICAL(myMutex);
7         blinkLedOnce(led, run_us);
8         portEXIT_CRITICAL(myMutex);
9         vTaskDelay(wait_ms);
10    }
11 }
```

Critical sections

```
1 portMUX_TYPE myMutex = portMUX_INITIALIZER_UNLOCKED;
2
3 void redTask(void * pvParameters) {
4     blinkLedForTimeCritical(L_R, 300, 1, &myMutex);
5 }
6
7 void yellowTask(void * pvParameters) {
8     blinkLedForTimeCritical(L_Y, 300, 1, &myMutex);
9 }
```

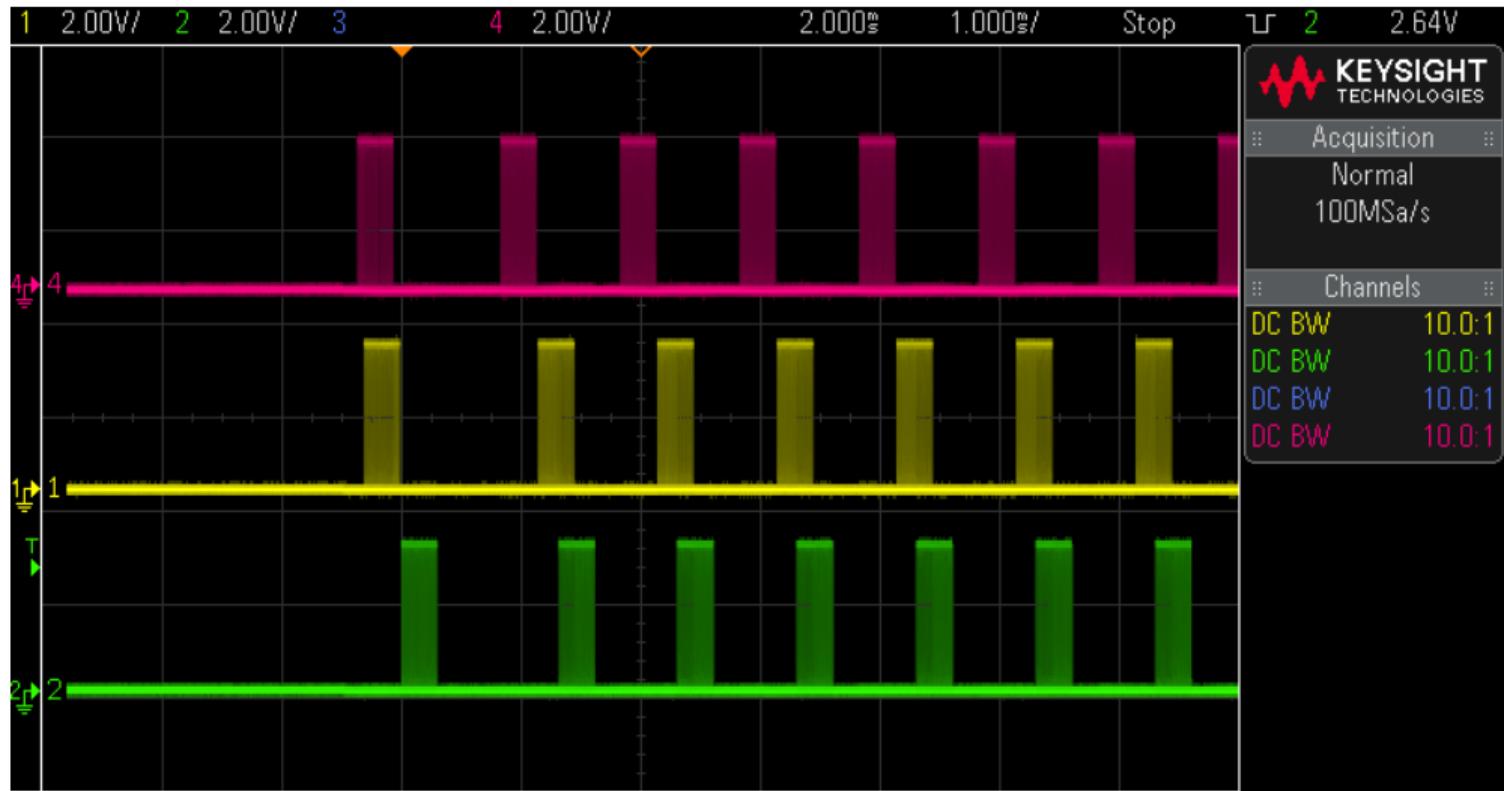
https://www.freertos.org/taskENTER_CRITICAL_taskEXIT_CRITICAL.html

Critical sections

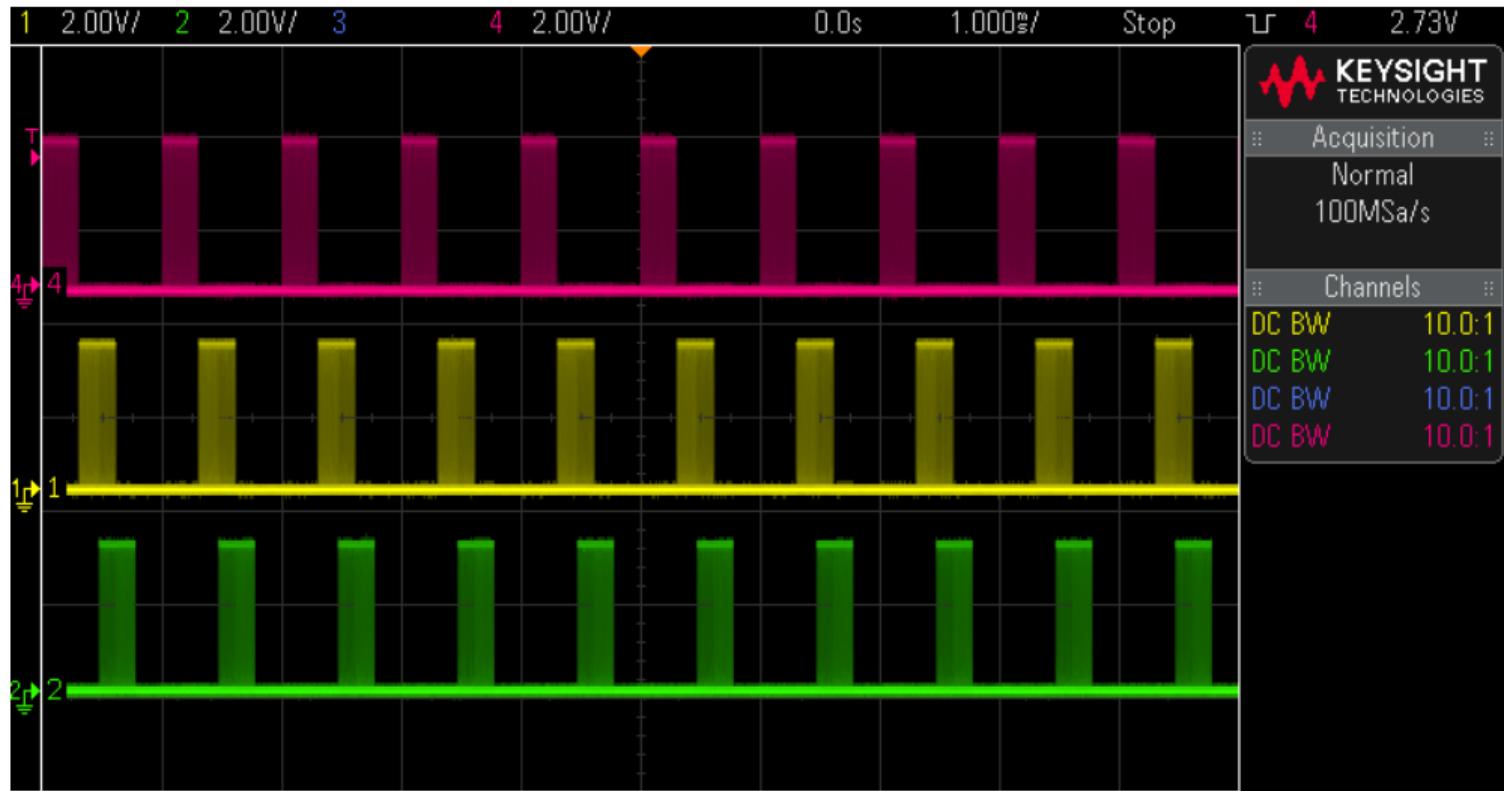
```
1 void setup() {
2     initLEDs();
3
4     xTaskCreate(redTask, "redTask", 10000, NULL, 3, NULL);
5     xTaskCreate(yellowTask, "yellowTask", 10000, NULL, 2, NULL);
6     blinkLedForTimeCritical(L_G, 300, 1, &myMutex);
7 }
```

https://www.freertos.org/taskENTER_CRITICAL_taskEXIT_CRITICAL.html

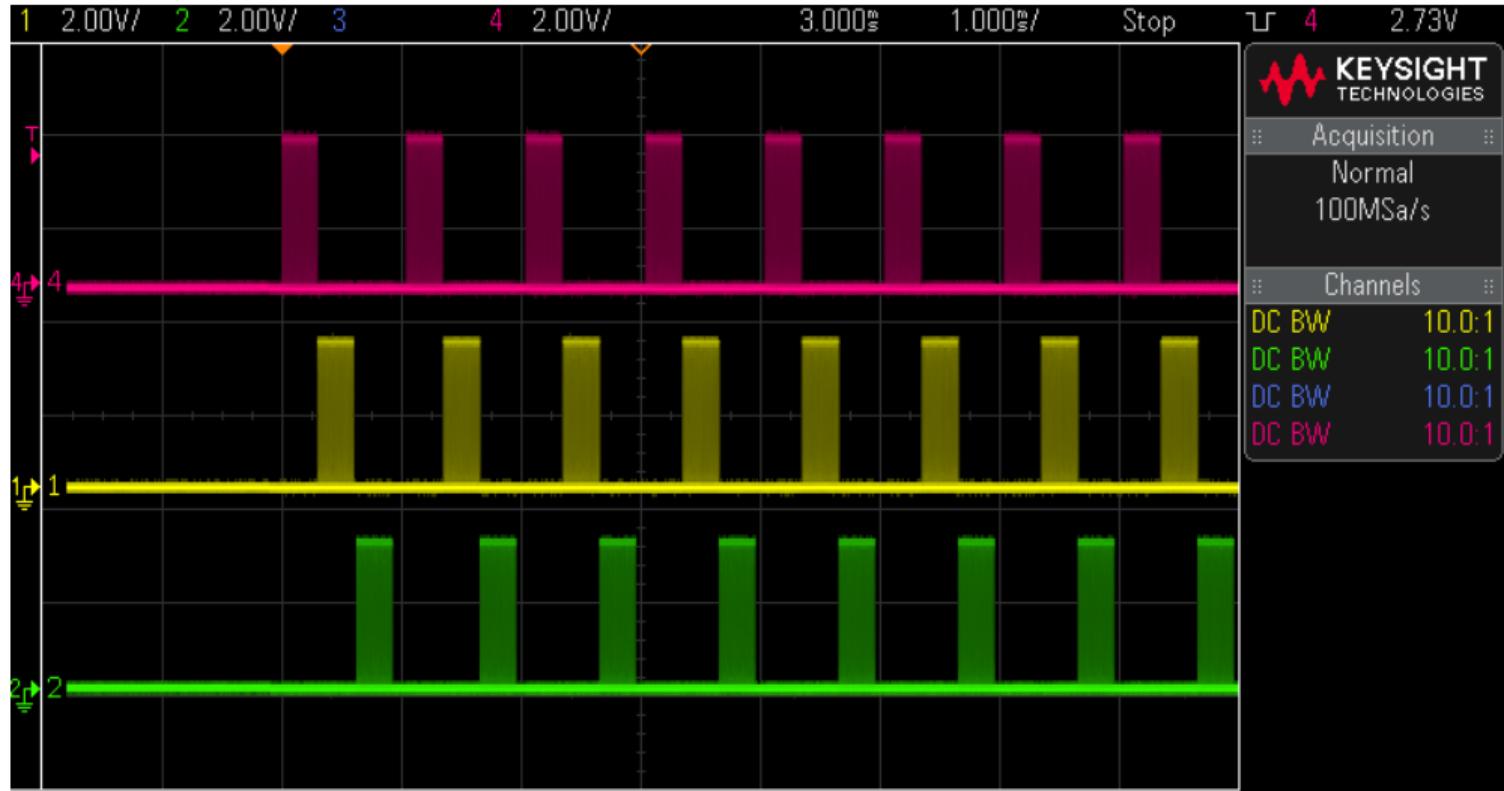
No critical sections



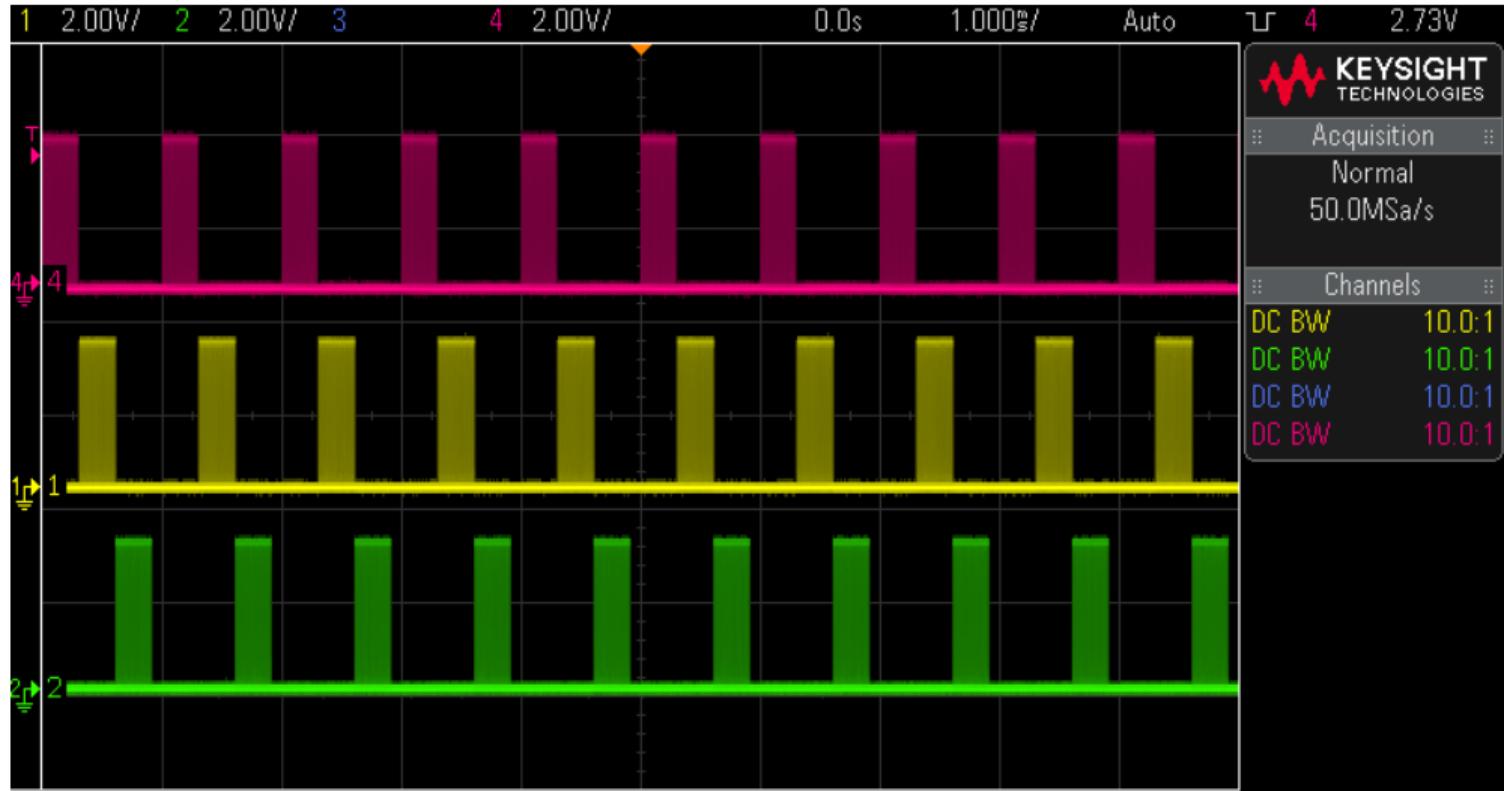
No critical sections



Critical sections



Critical sections



Overview

FreeRTOS and used hardware

Minimal working example

Tasks and jobs

- New task creation

- Task properties

- Multiple tasks

- Jobs

- Timers

Sharing data among threads

- Critical sections

- Queues

Queues

- Producer/consumer layout
- More tasks can use the same queue simultaneously

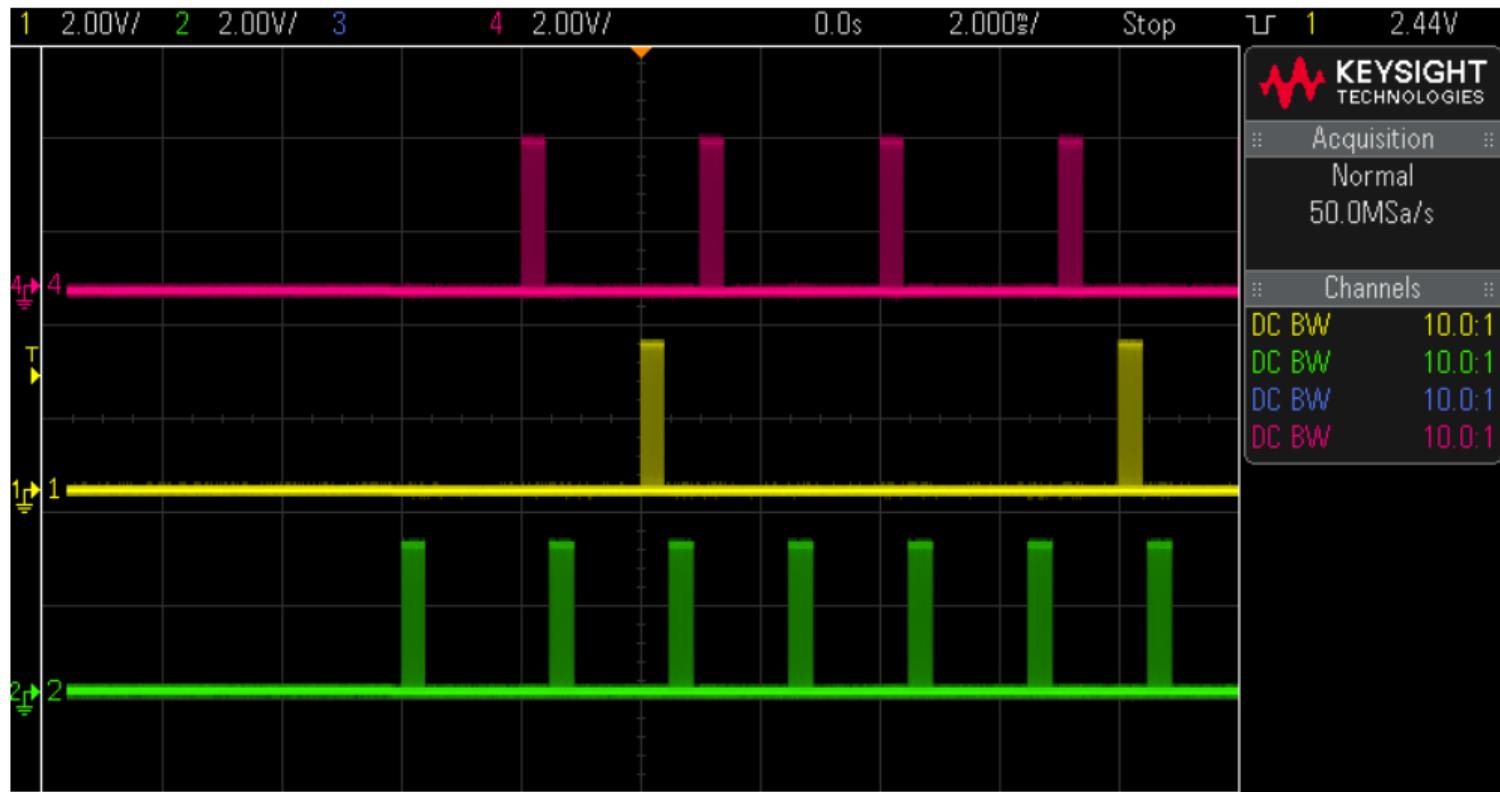
```
1 QueueHandle_t myQueue = xQueueCreate(10, sizeof(uint32_t));  
2  
3 xQueueSend(myQueue, (void*) &data, (TickType_t) 0);  
4  
5 xQueueReceive(myQueue, &data, (TickType_t) 0);
```

<https://www.freertos.org/Embedded-RTOS-Queues.html>

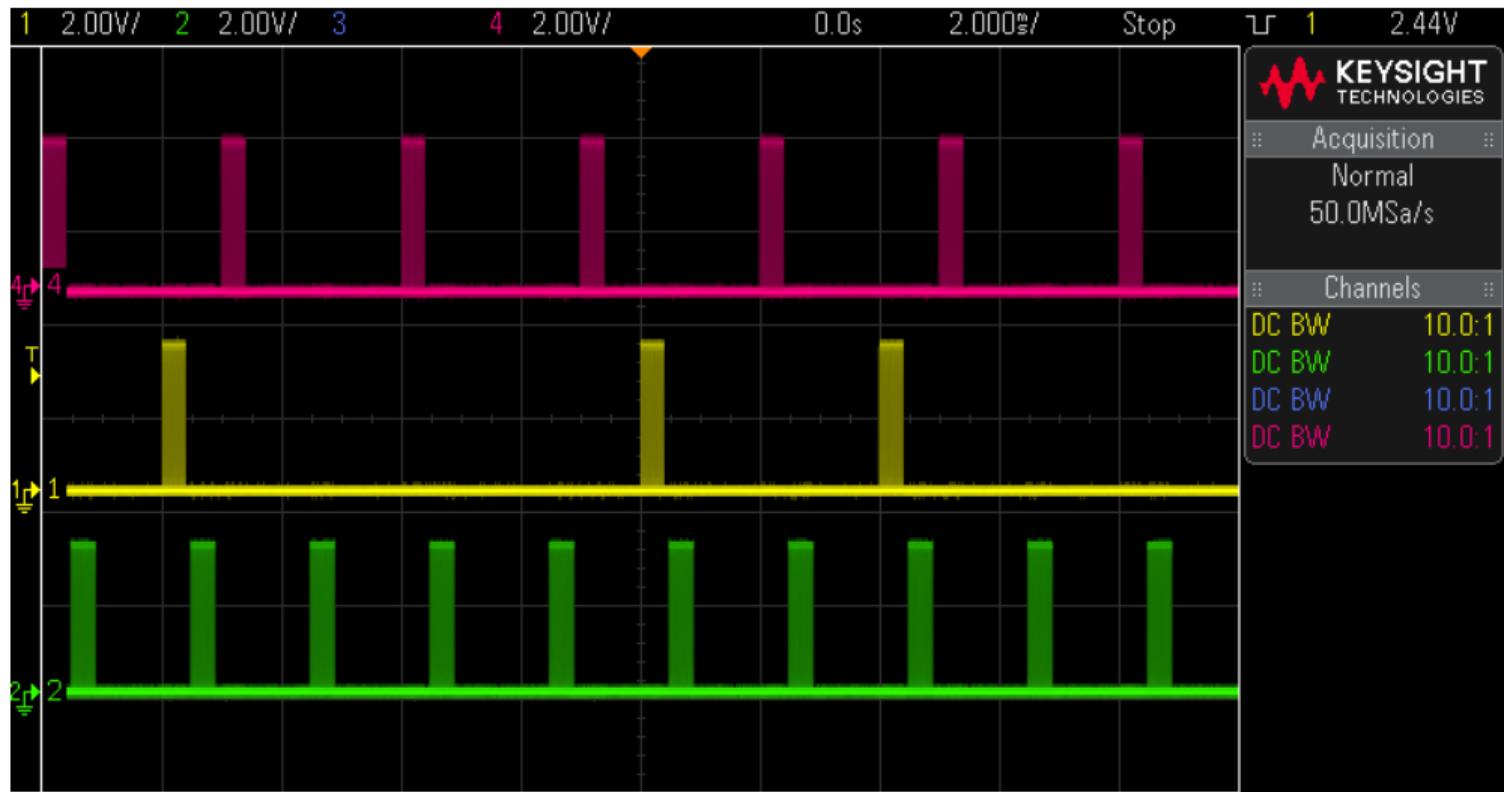
Queues (example)

- Green task sends every 2 ms
- Red task listens every 3 ms
- Yellow task listens every 4 ms

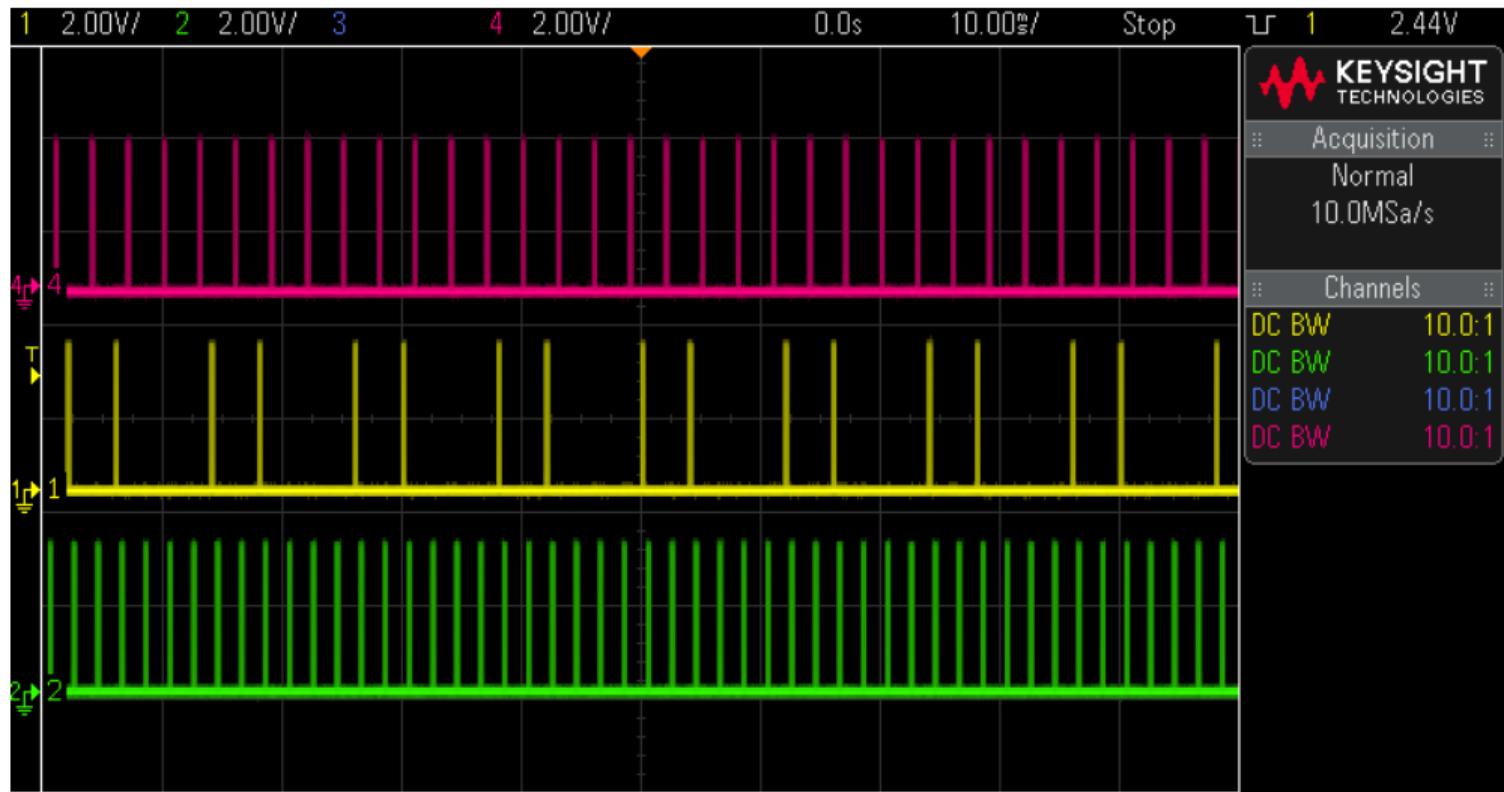
Queues (example)

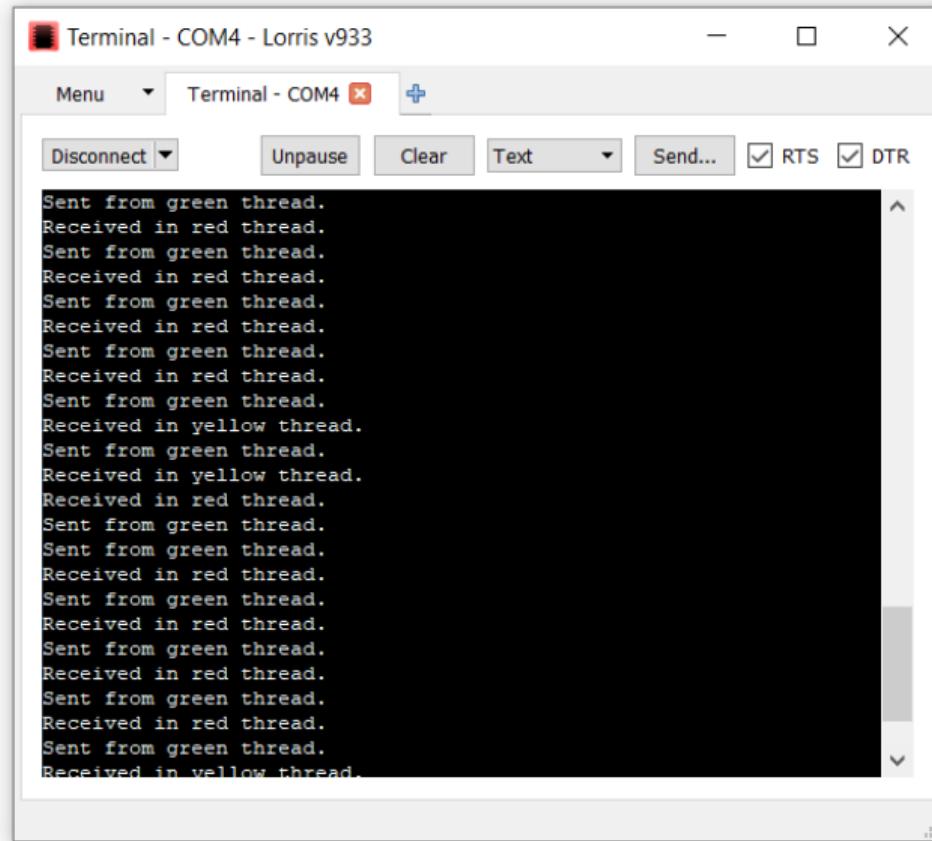


Queues (example)

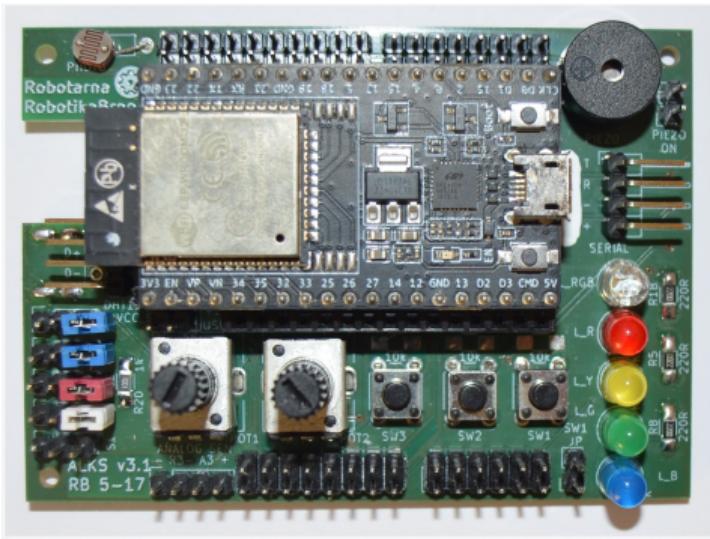
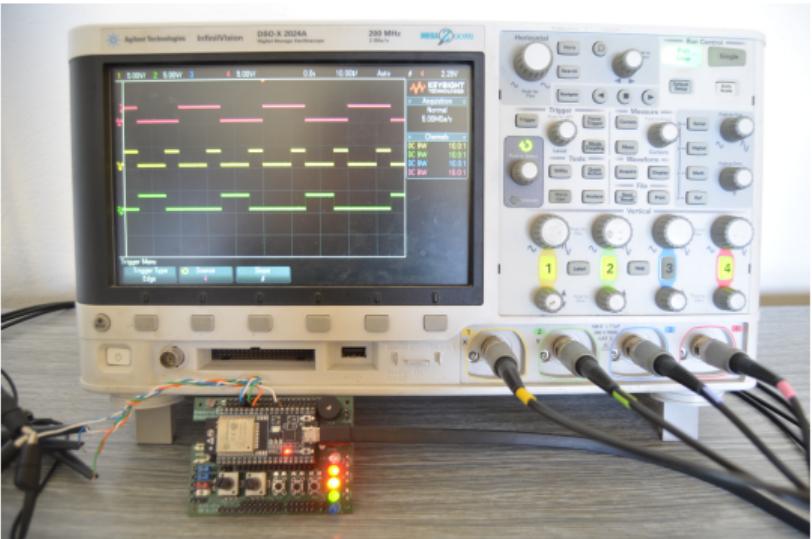


Queues (example)





Thank you!



Questions?

Bedřich Said: 409874@mail.muni.cz

MUNI
FACULTY
OF INFORMATICS