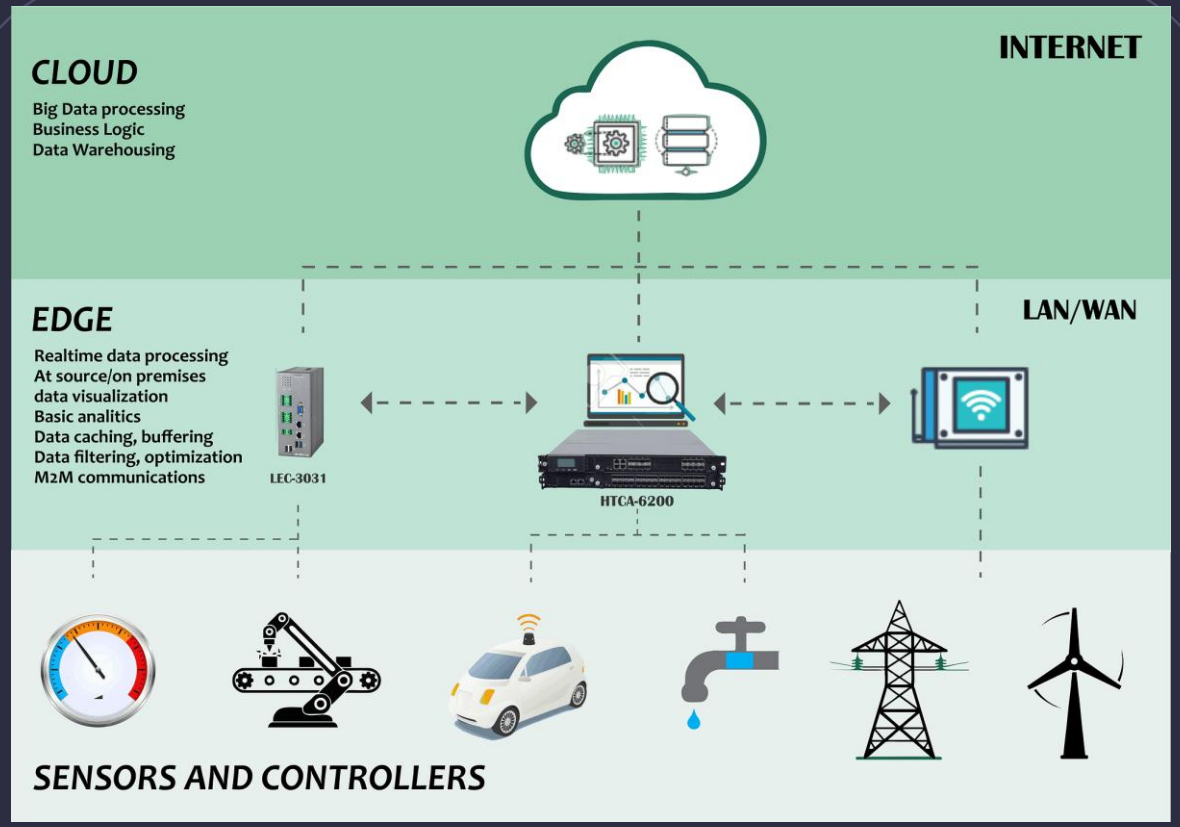
Internet of Things

System Design with Sensors
FreeRTOS Applications

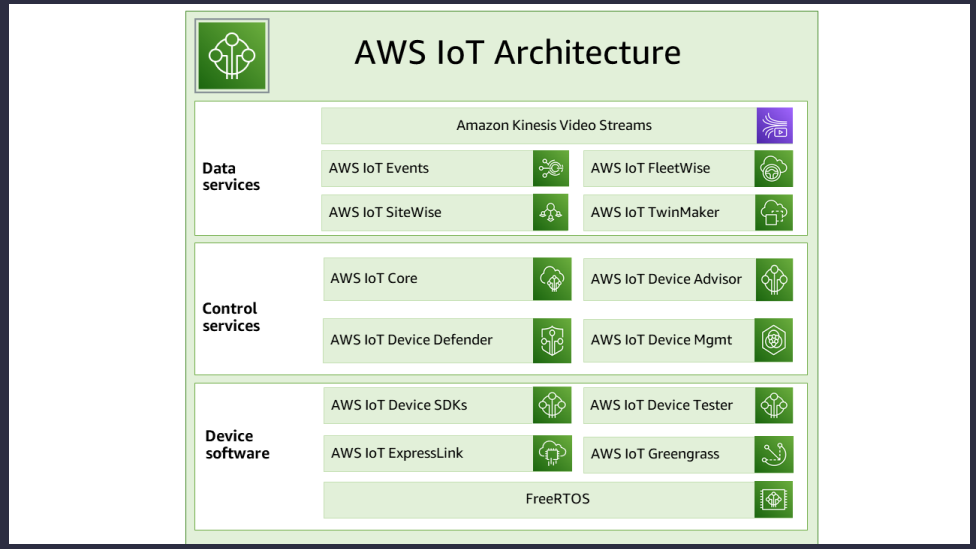
Fenerbahce University

How IoT Systems Work (End-to-End)

A practical mental model: device → network → edge/gateway → cloud → dashboard → actions



Edge–Fog–Cloud reference diagram



Example: AWS IoT “how it works” overview

System Design with Sensors II - FreeRTOS

How to Build an Embedded System?

- Buttons
- Sensors
  - Temperature
  - IMU
  - GPS
  - …
- Communication Interfaces
  - UART
  - SPI
  - I2C
  - Ethernet
  - …
- RF Transceiver

- LEDs
- Motors
- Communication Interfaces
  - UART
  - SPI
  - I2C
  - Ethernet
  - …
- RF Transceiver

System Design with Sensors II - FreeRTOS

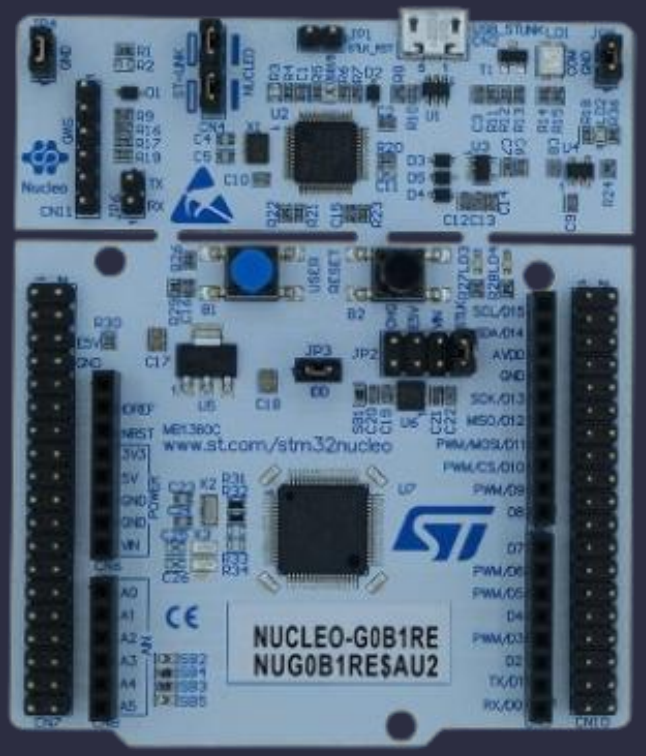
STM32 Nucleo Development Board

- STM32 Nucleo Development Board

- STM32 microcontroller in LQFP64 package
- Three LEDs:
  - USB communication (LD1), user LED (LD2), power LED (LD3)
- Two push-buttons: USER and RESET
- Two types of extension resources
  - ARDUINO® Uno V3 connectivity
- ST morpho extension pin headers for full access to all STM32 I/Os
- Three different interfaces supported on USB:
  - Virtual COM port
  - Mass storage
  - Debug port

- For details:

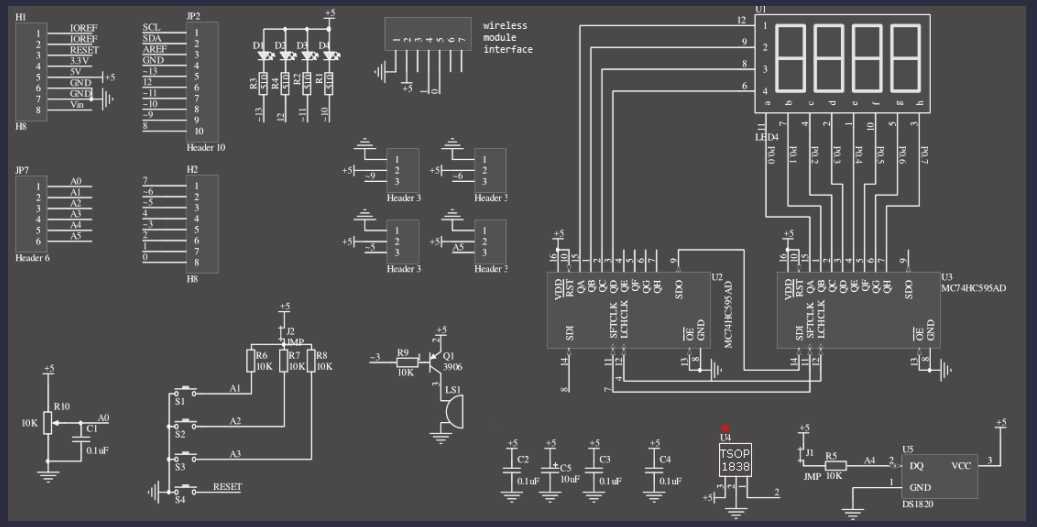
https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf



System Design with Sensors II - FreeRTOS

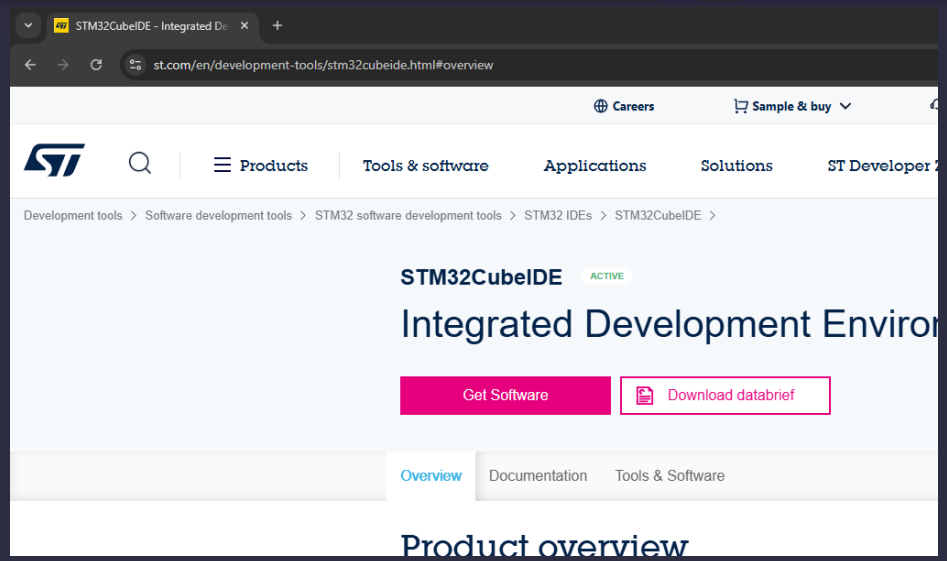
- STM32 Nucleo Development Board

- MPU, STM32 Nucleo with Extension Board



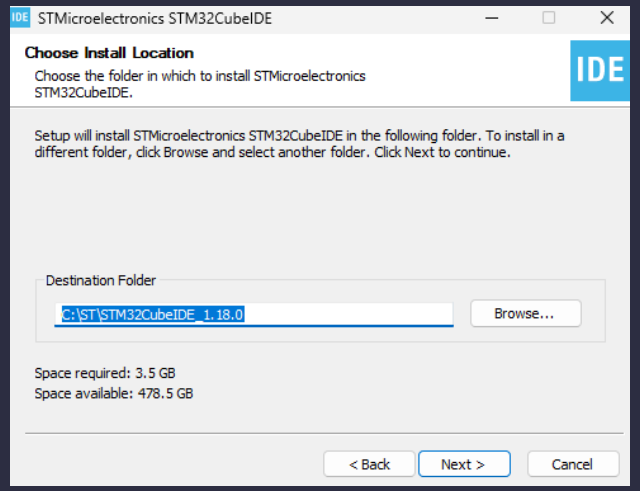
System Design with Sensors II - FreeRTOS

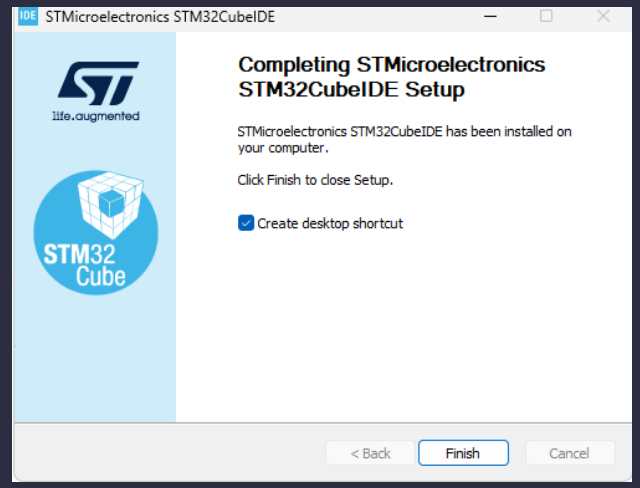
- STM32 Nucleo Development Board
  - Download STM32CubeIDE
  - https://www.st.com/en/development-tools/stm32cubeide.html#overview



System Design with Sensors II - FreeRTOS

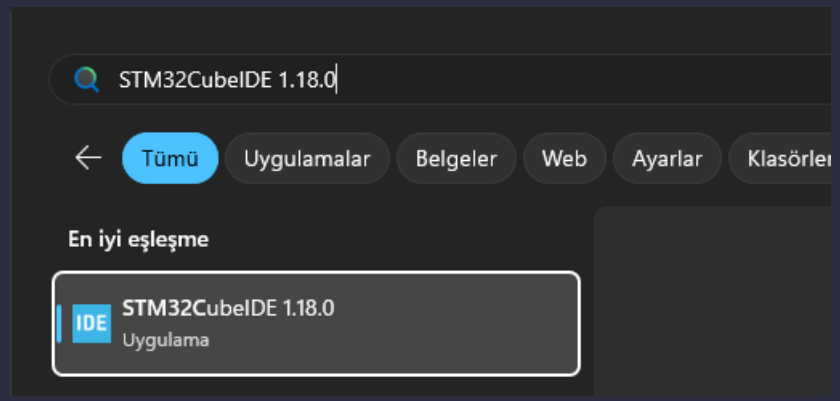
- STM32 Nucleo Development Board
  - Install STM32CubeIDE

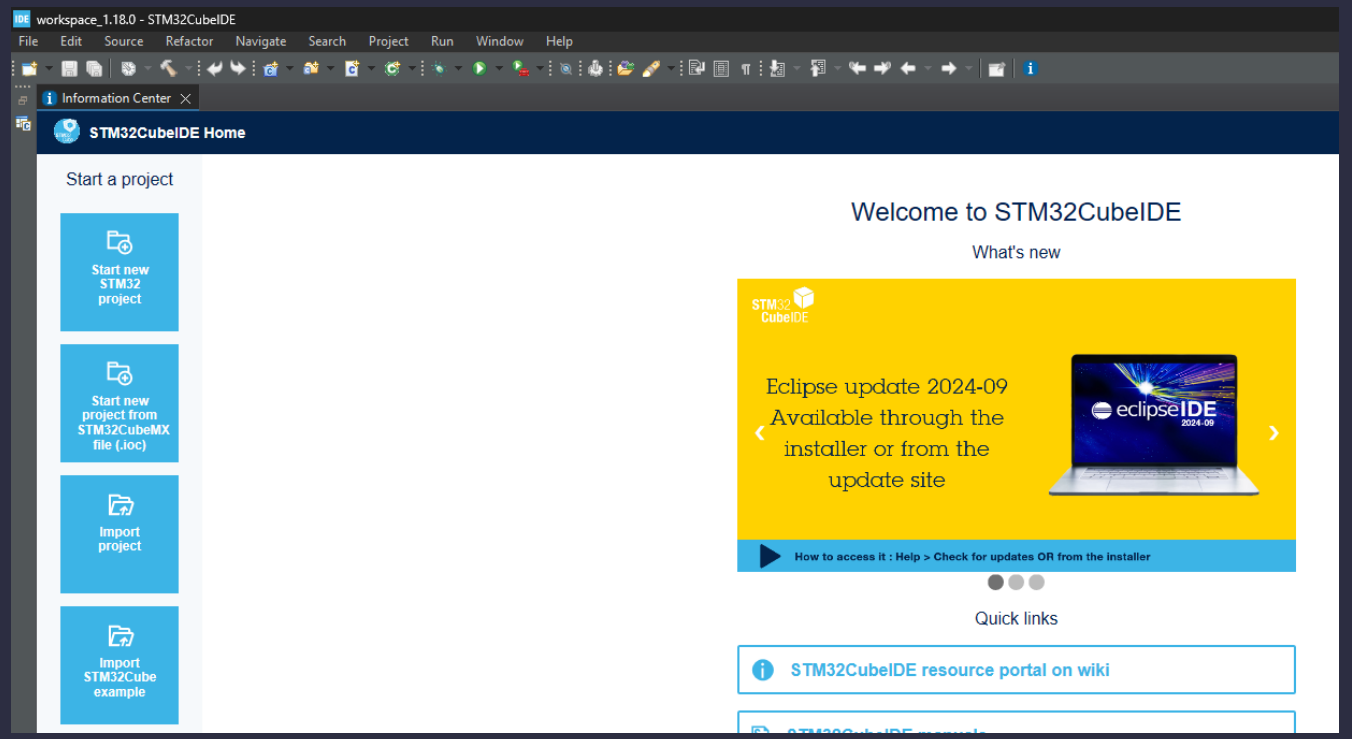




System Design with Sensors II - FreeRTOS

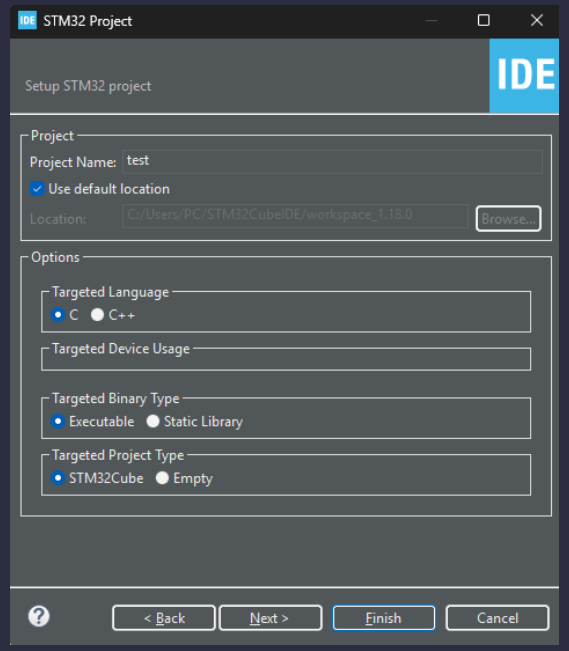
- STM32 Nucleo Development Board
  - Start STM32CubeIDE

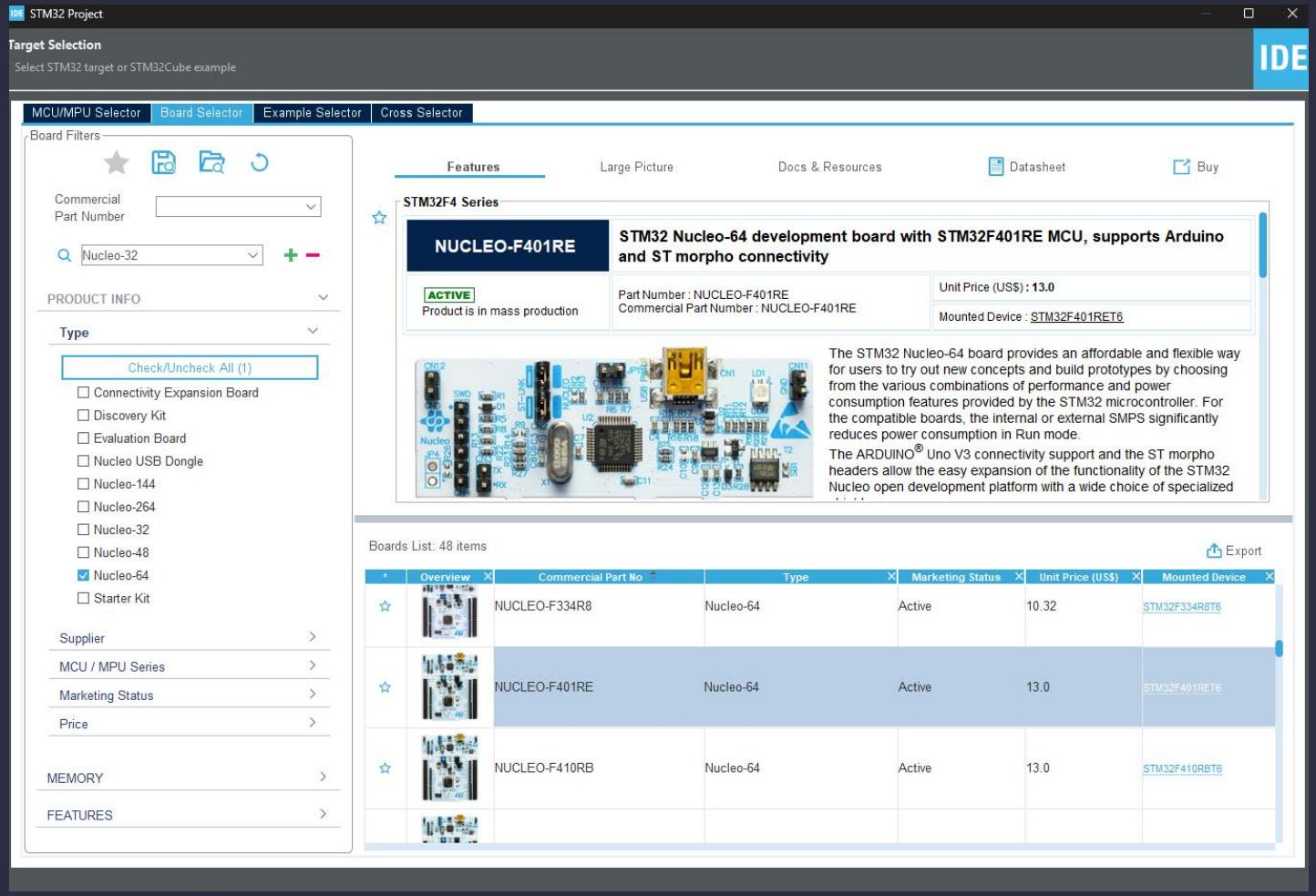




System Design with Sensors II - FreeRTOS

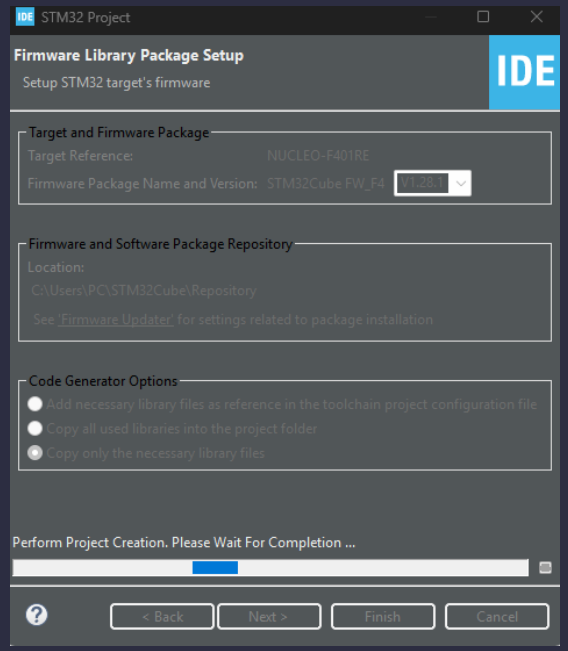
- STM32 Nucleo Development Board
  - Create Project
  - Select Development Board
  - Nucleo-F401RE

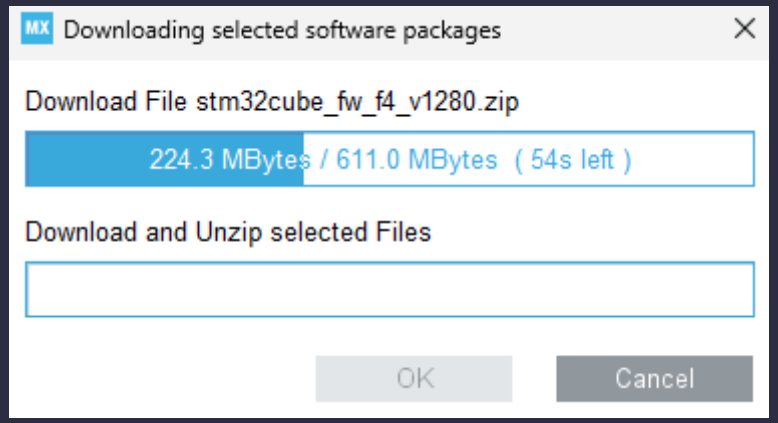




System Design with Sensors II - FreeRTOS

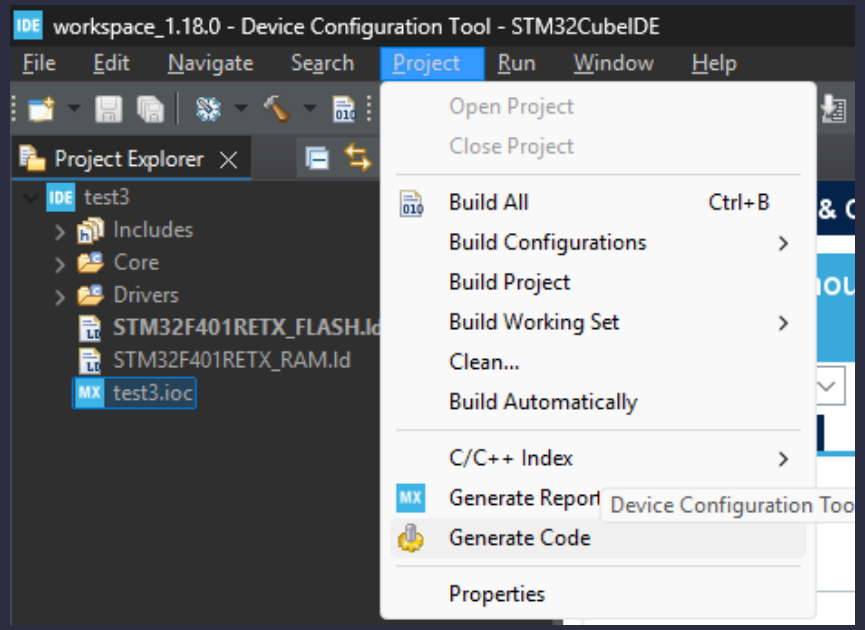
- STM32 Nucleo Development Board
  - Create Project
  - Select Development Board
  - Nucleo-F401RE

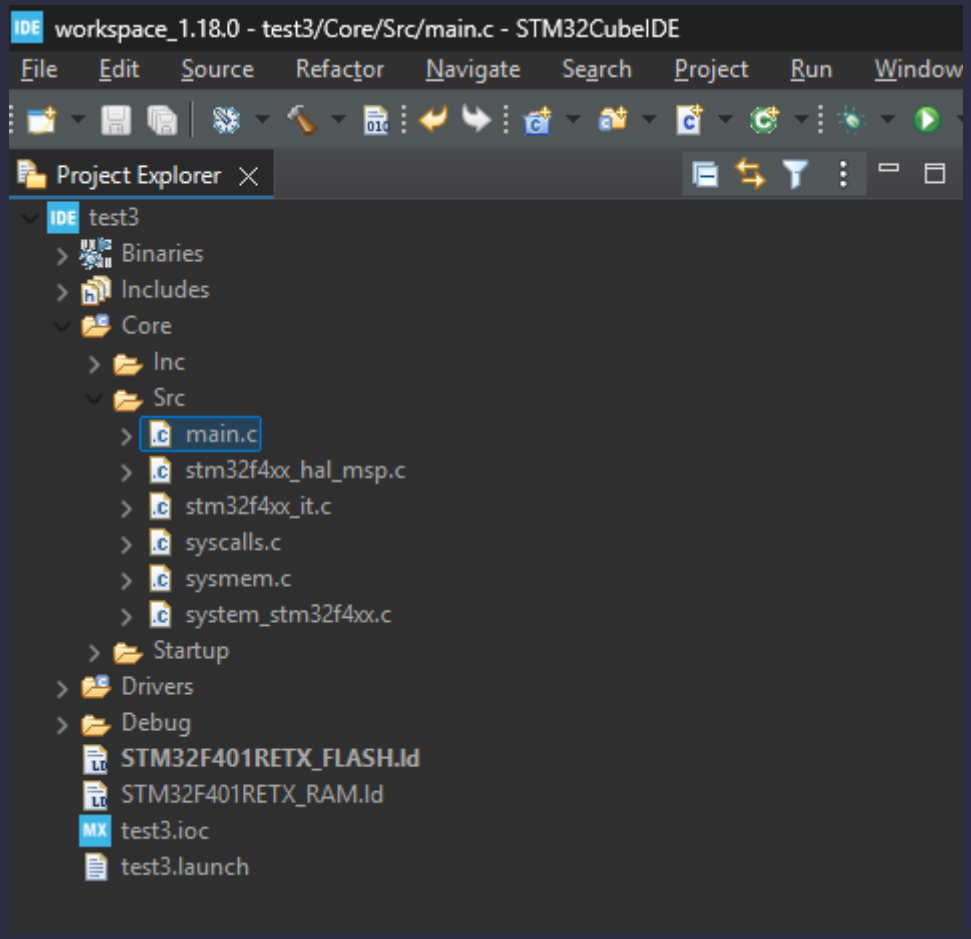




System Design with Sensors II - FreeRTOS

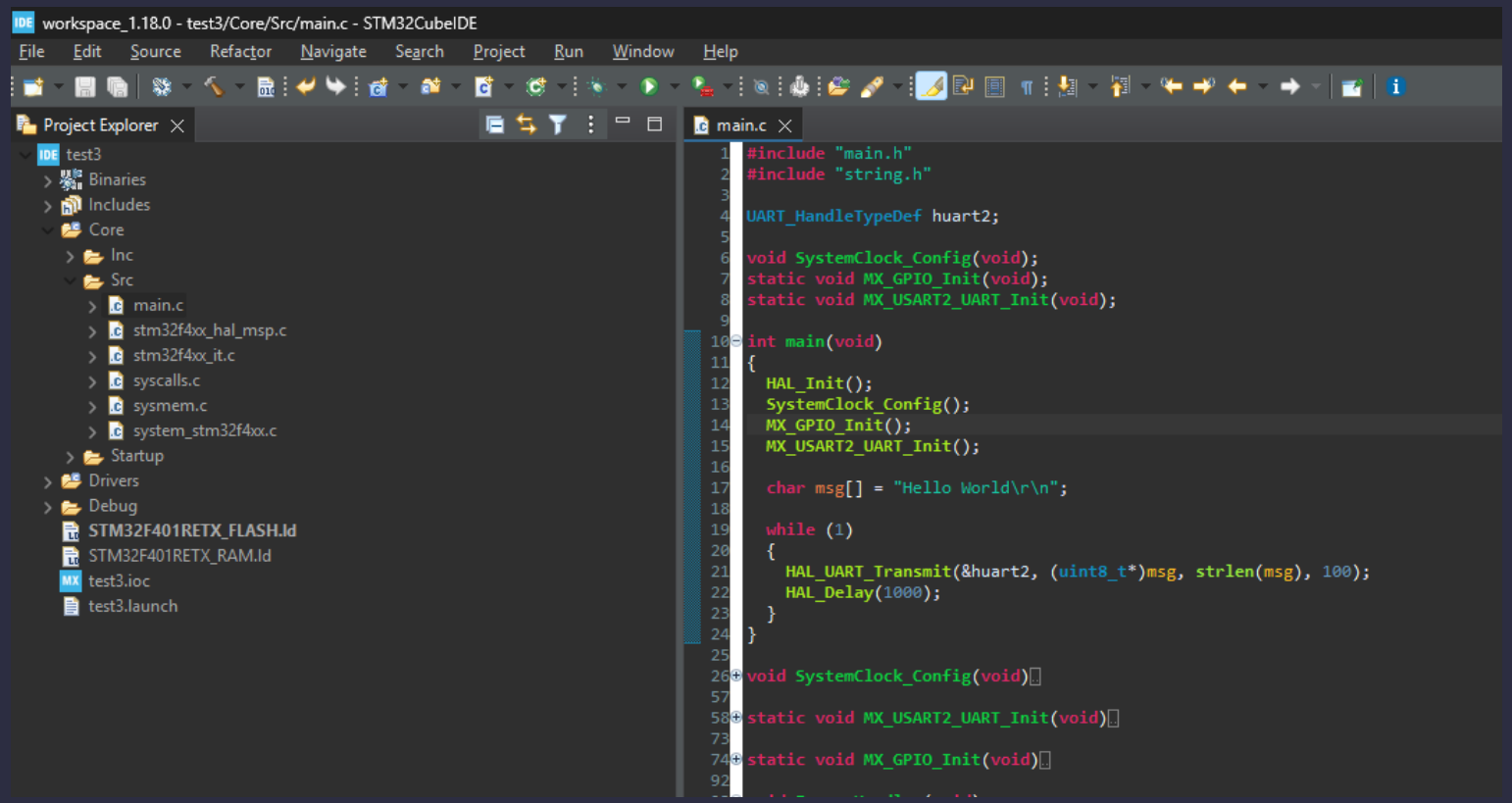
- STM32 Nucleo Development Board
  - Generate Code





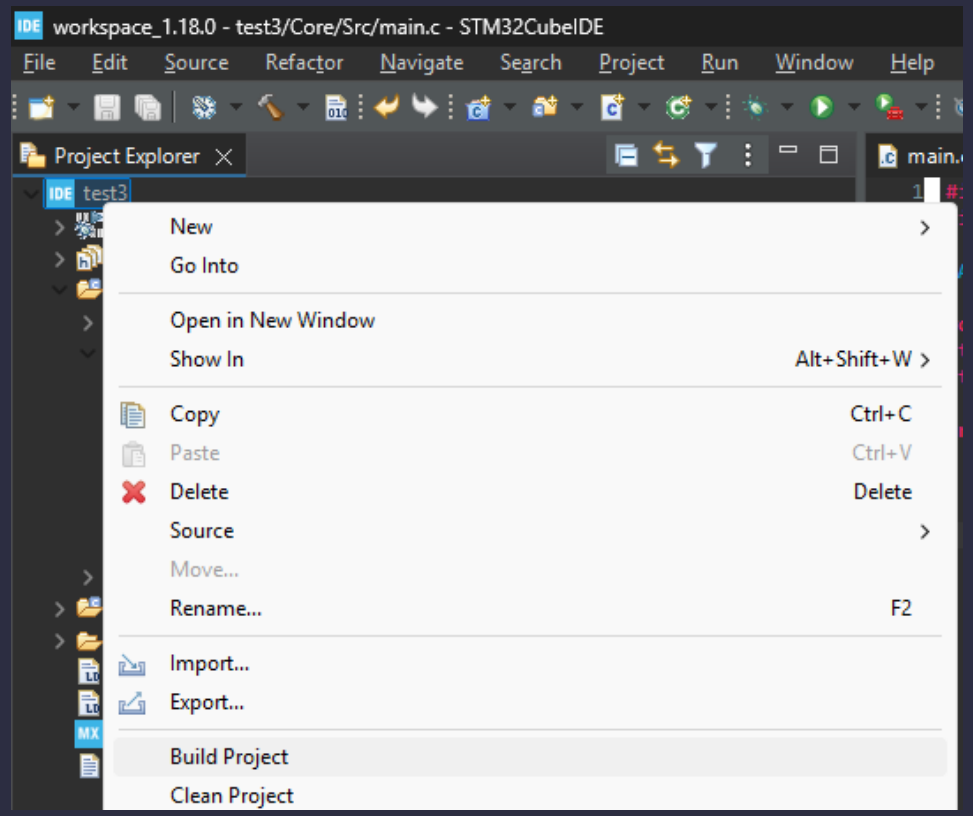
System Design with Sensors II - FreeRTOS

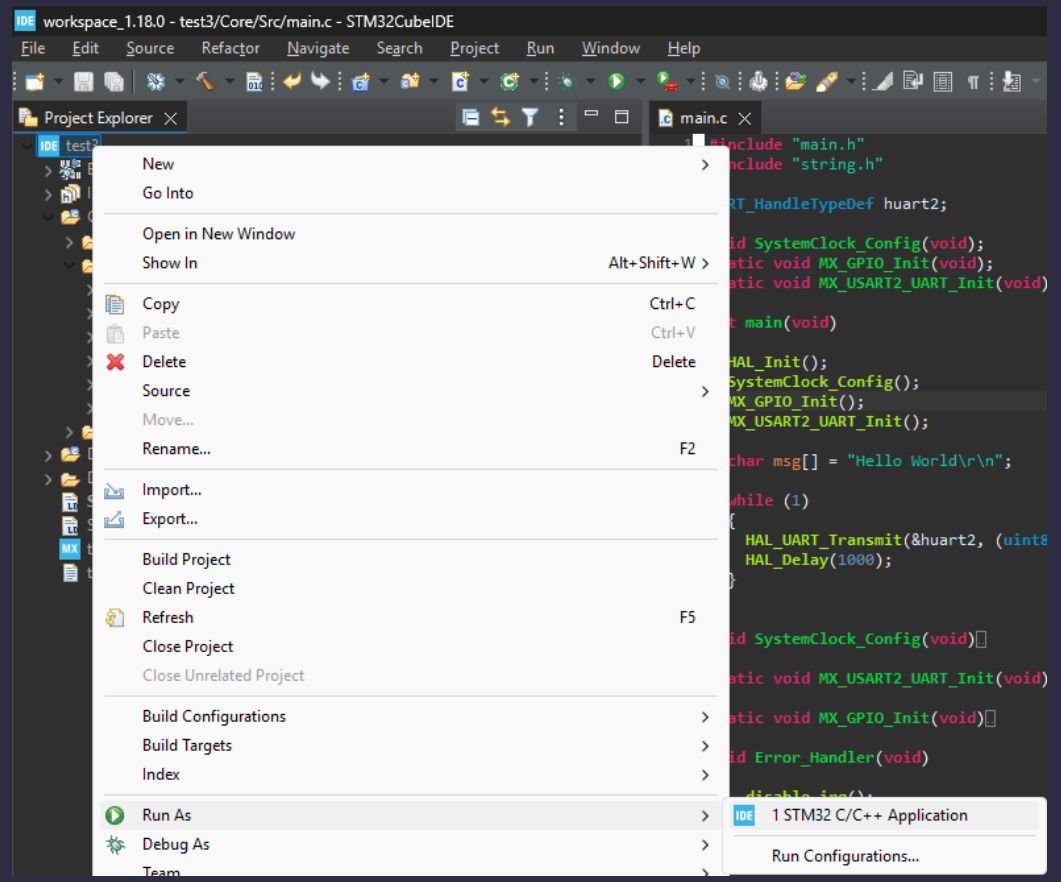
- STM32 Nucleo Development Board
  - Type Code



System Design with Sensors II - FreeRTOS

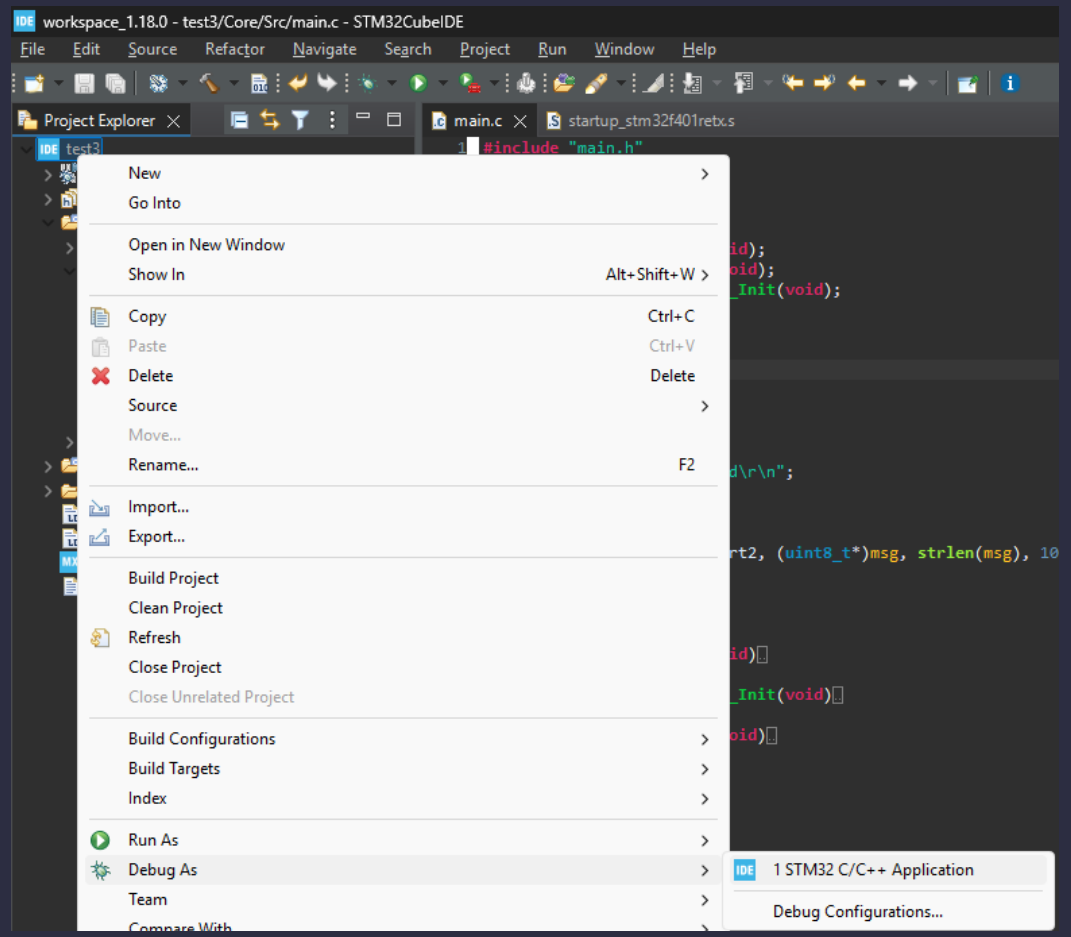
- STM32 Nucleo Development Board
  - Compile and Run the Code





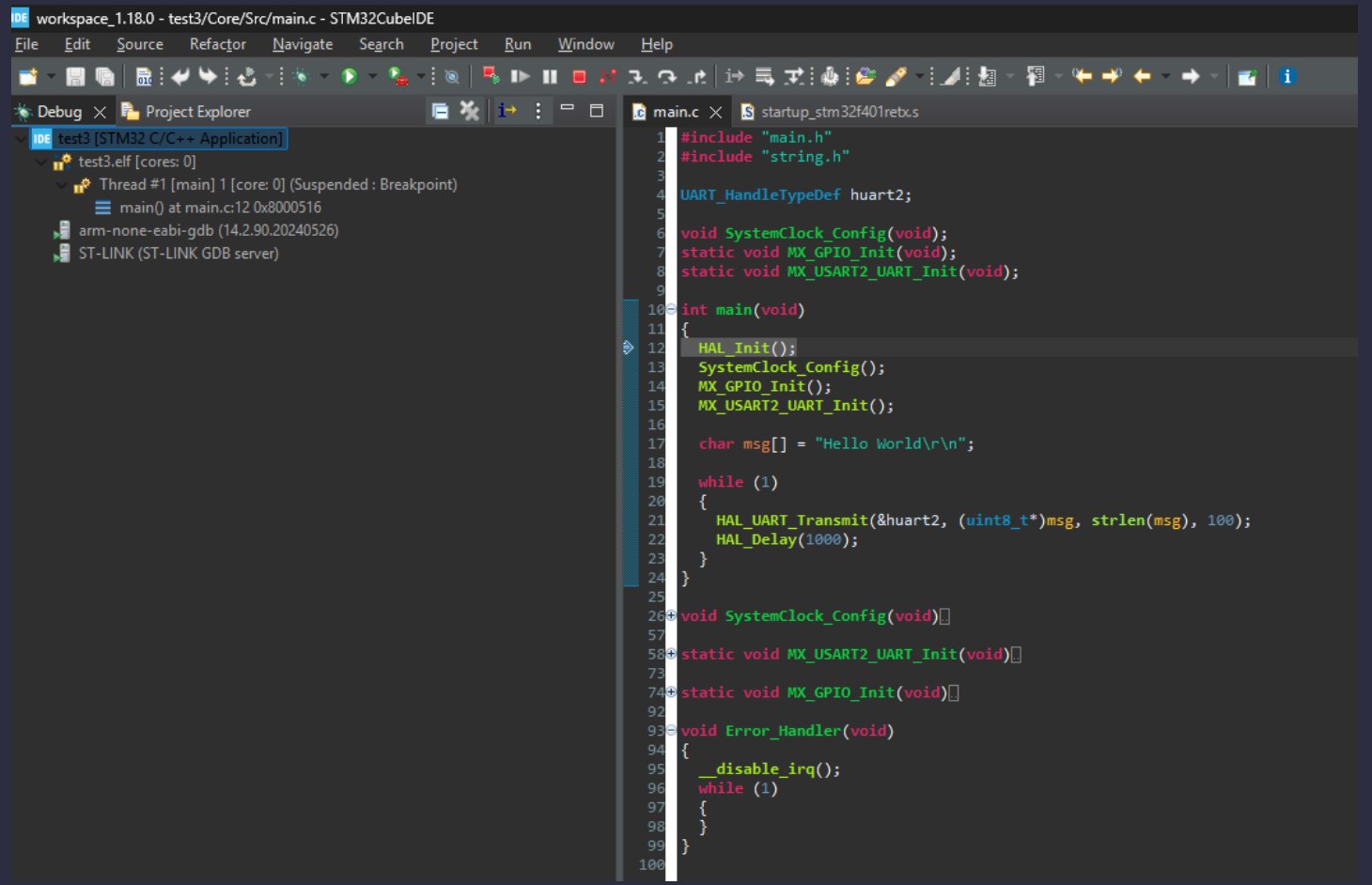
System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Debug



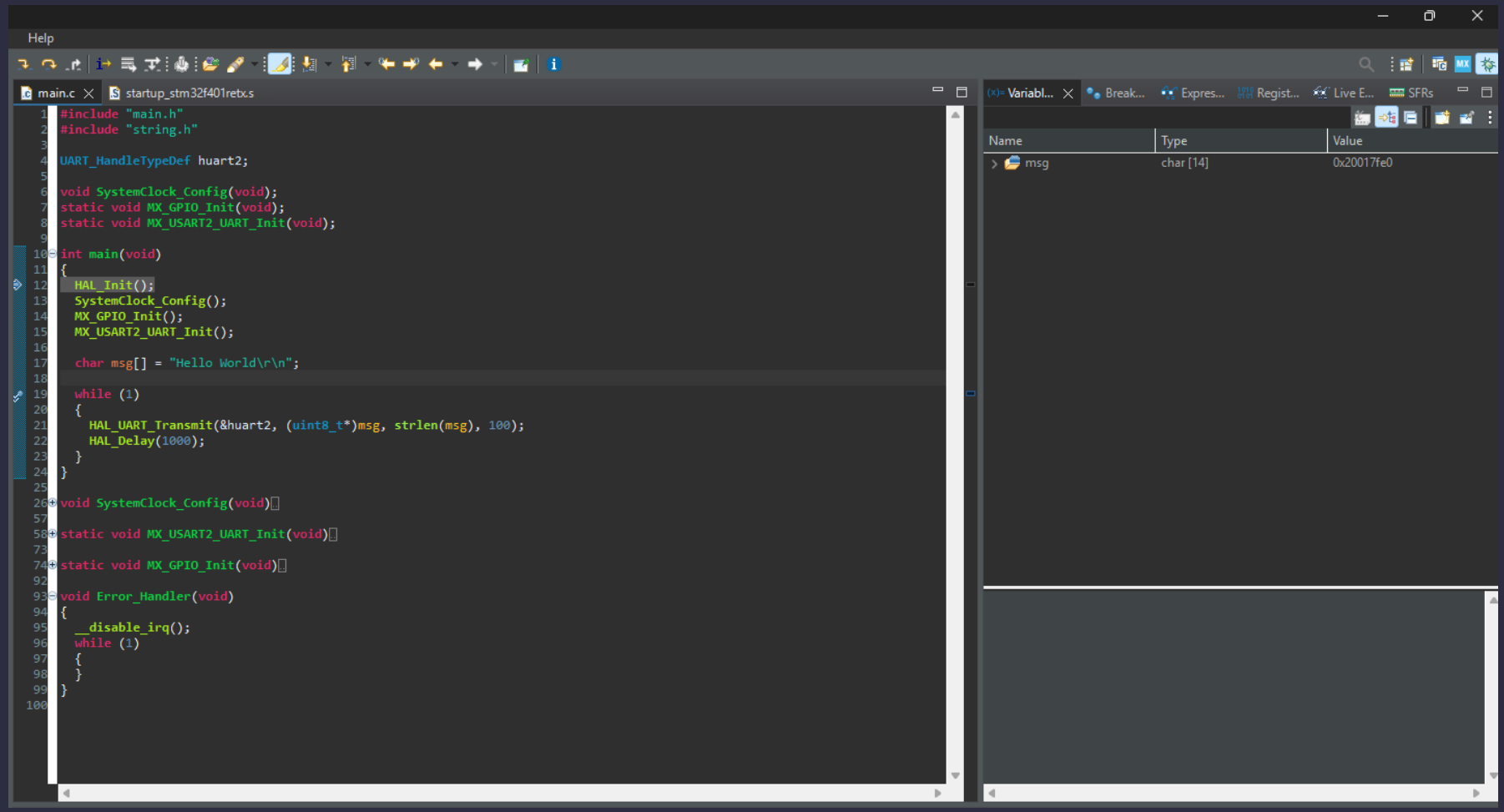
System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Debug



System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Variables



System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Step Down F6 Key

main.c | startup_stm32f401retx.s

```c
#include "main.h"
#include "string.h"

UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();

  char msg[] = "Hello World\r\n";

  while (1)
  {
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), 100);
    HAL_Delay(1000);
  }
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Hello World C Code

```c
#include "main.h"
#include "string.h"

UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();

  char msg[] = "Hello World\r\n";

  while (1)
  {
    HAL_UART_Transmit(&huart2, (uint8_t*)msg,
strlen(msg), 100);
    HAL_Delay(1000);
  }
}
```

Program STM32

Open Putty

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```c
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
}

void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {
  }
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Output Led Blink

```c
#include "stm32f4xx_hal.h"
#include "string.h"

UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  char msg[] = "Hello World\r\n";
  while (1)
  {
    HAL_UART_Transmit(&huart2, (uint8_t *)msg,
strlen(msg), 100);
    HAL_Delay(1000);
  }
}
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    while(1);
  }
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    while(1);
  }
}
```

```c
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength =
UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl =
UART_HWCONTROL_NONE;
  huart2.Init.OverSampling =
UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    while(1);
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5,
GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed =
GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Output Led Blink

Program STM32

Open Putty

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Input/Output Led Blink

```c
#include "stm32f4xx_hal.h"

void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  while (1)
  {
    if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1)==GPIO_PIN_RESET)
    {
      if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5)==GPIO_PIN_SET)
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
      else
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_Delay(200);
    }
  }
}
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|
                                RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2);
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOA_CLK_ENABLE();
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Input/Output Led Blink

Program STM32

Open Putty

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - UART

```c
#include "stm32f4xx_hal.h"
#include "string.h"
#include <stdbool.h>

UART_HandleTypeDef huart2;
bool ledState = false;
bool lastButtonState = true;
uint8_t msgClosed[] = "LED Closed\r\n";
uint8_t msgOpen[] = "LED Open\r\n";

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  while (1)
  {
    bool buttonState = (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1) == GPIO_PIN_SET);
    if (lastButtonState && (!buttonState))
    {
      ledState = !ledState;
      if (ledState)
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      else
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
      if (ledState)
        HAL_UART_Transmit(&huart2, msgClosed, strlen((char*)msgClosed), 100);
      else
        HAL_UART_Transmit(&huart2, msgOpen, strlen((char*)msgOpen), 100);
      HAL_Delay(2000);
    }
    lastButtonState = buttonState;
    HAL_Delay(50);
  }
}
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|
                                RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2);
}

static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  HAL_UART_Init(&huart2);
}
```

```c
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - UART

Program STM32

Open Putty

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Counter

```c
#include "stm32f4xx_hal.h"
#include <string.h>
#include <stdio.h>
#include <stdbool.h>

UART_HandleTypeDef huart2;
int pressCount = 0;
GPIO_PinState buttonState, lastButtonState;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  lastButtonState = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);
  char buffer[50];
  while (1)
  {
    buttonState = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);
    if(lastButtonState == GPIO_PIN_SET && buttonState == GPIO_PIN_RESET)
    {
      pressCount++;
      sprintf(buffer, "Butona basildi. Sayac: %d\r\n", pressCount);
      HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 100);
      HAL_Delay(2000);
    }
    lastButtonState = buttonState;
    HAL_Delay(50);
  }
}
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|
                                RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2);
}

static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  HAL_UART_Init(&huart2);
}
```

```c
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOA_CLK_ENABLE();
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

System Design with Sensors II - FreeRTOS

- STM32 Nucleo Development Board
  - Counter

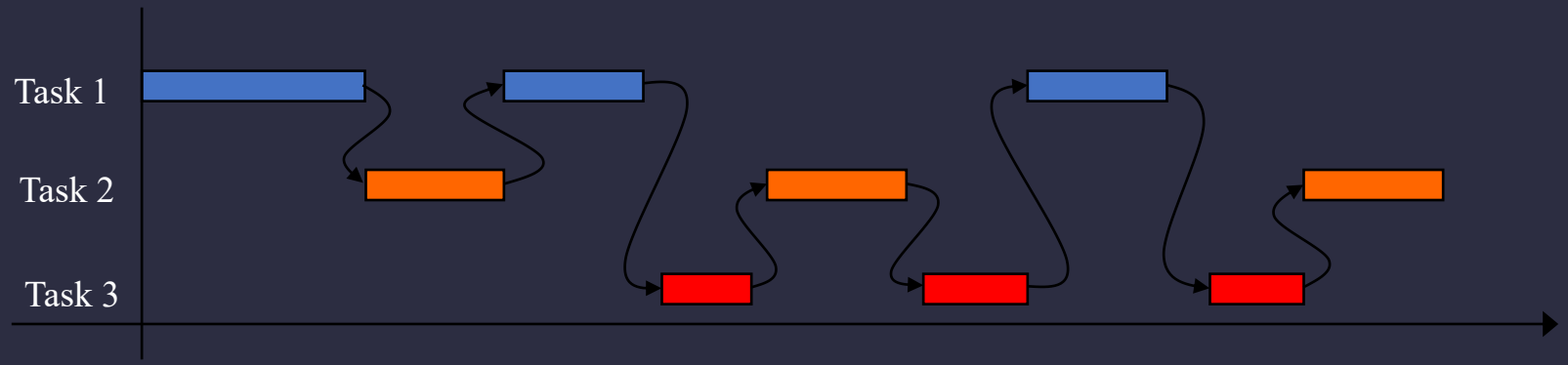
Program STM32

Open Putty

System Design with Sensors II - FreeRTOS

Main tasks of a OS (Operating System)

- Process Management
  - Process creation
  - Process loading
  - Process execution control
  - Interaction of the process with signal events
  - Process monitoring CPU allocation Process termination



System Design with Sensors II - FreeRTOS

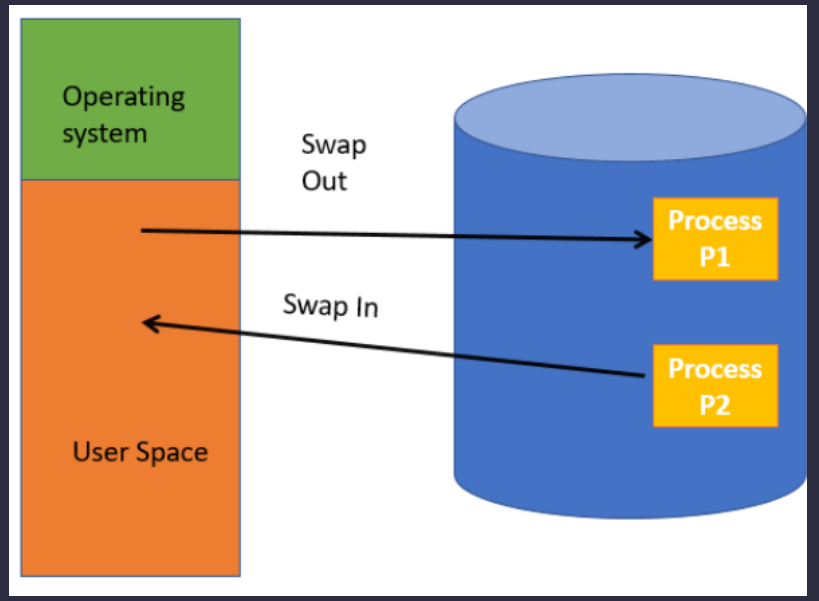
Main tasks of a OS

- Interprocess Communication
  - Synchronization and coordination
  - Deadlock detection and protection
  - Data Exchange Mechanisms

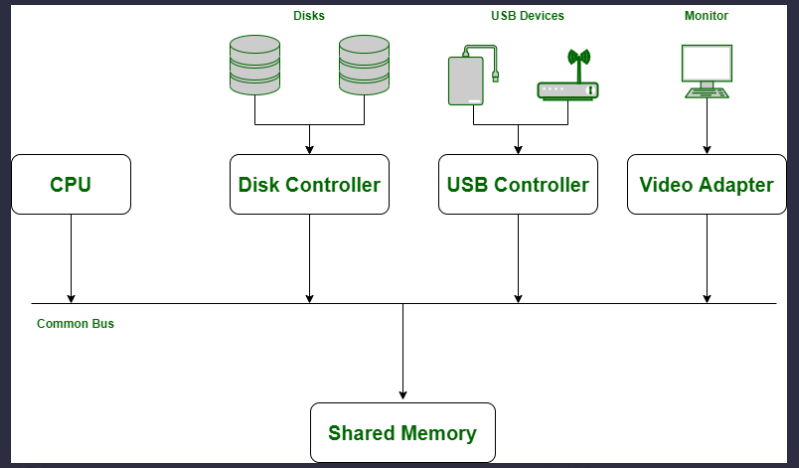
System Design with Sensors II - FreeRTOS

Main tasks of a OS

- Memory Management
  - Services for file creation, deletion
  - Access control and protection

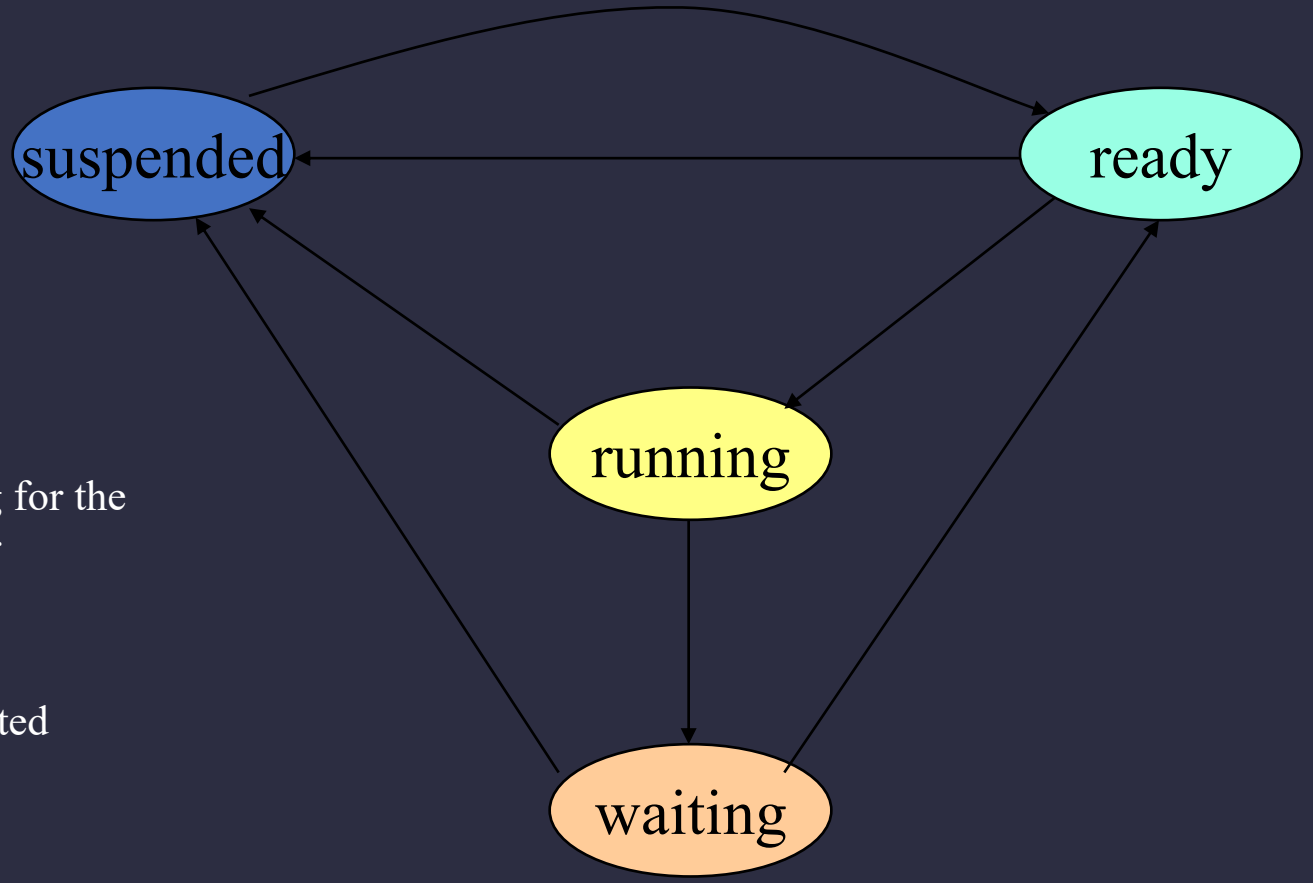


- Input/output Management
  - Handles requests and release subroutines for a variety of peripherals and read, write and reposition programs



- Handle Interrupts

System Design with Sensors II - FreeRTOS



suspended:
the task doesn‘t take part on what is goining on

ready:
the task is ready and waiting for the assignment of the processor

running (busy):
the code of the task is executed

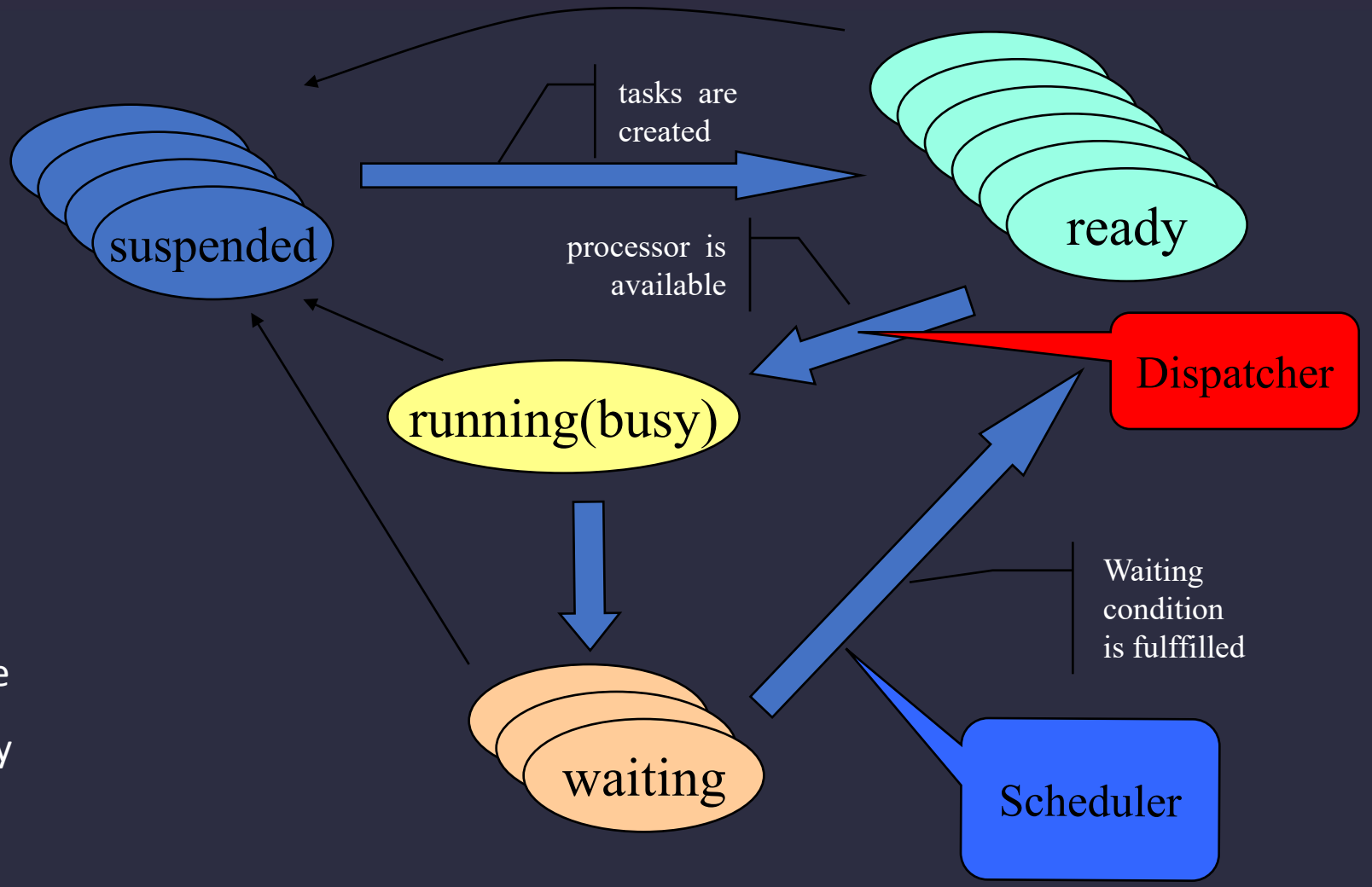
waiting:
the task is waiting for the occurrence of an event

System Design with Sensors II - FreeRTOS

- Threads are lightweight processes
  - Inherits only part of process
  - Belongs to the same environment as other threads making up the process
  - May be stopped, started and resumed

Multitasking

"Pseudo parallelism" - to handle multiple external events occurring simultaneously



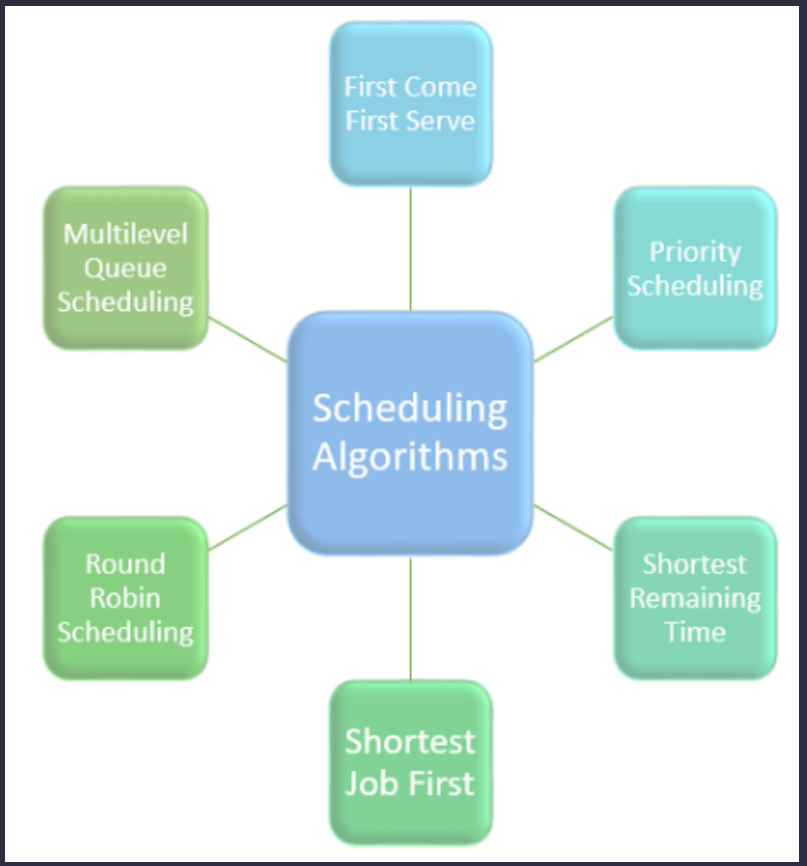
System Design with Sensors II - FreeRTOS

Types of Scheduling Policies

- Deadline driven
- Cooperative — relies on current process to give up the CPU
- Pre-emptive priority scheduling — higher priority tasks may interrupt lower priority tasks
    - Static — priorities are set before the system begins execution
    - Dynamic — priorities can be redefined at run time
- Round Robin — give each task an equal share of the processor

  Implemented when all tasks or threads have the same priority level

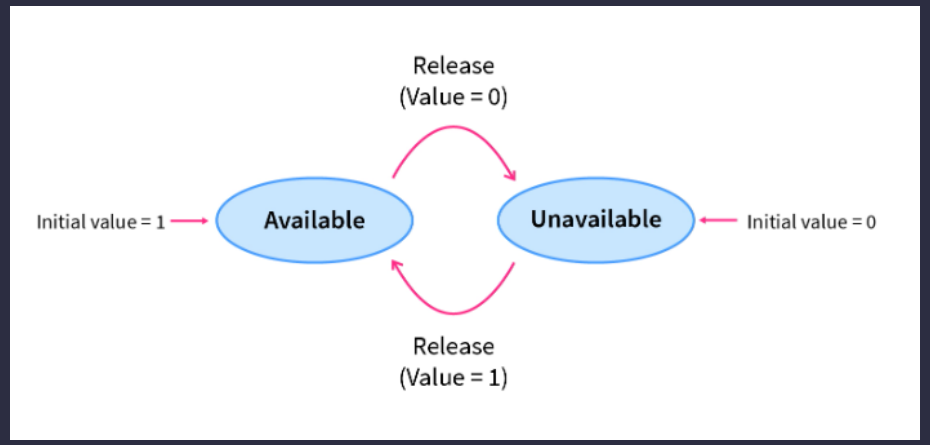
  May be implemented within a priority range



System Design with Sensors II - FreeRTOS

Synchronization and exclusion objects

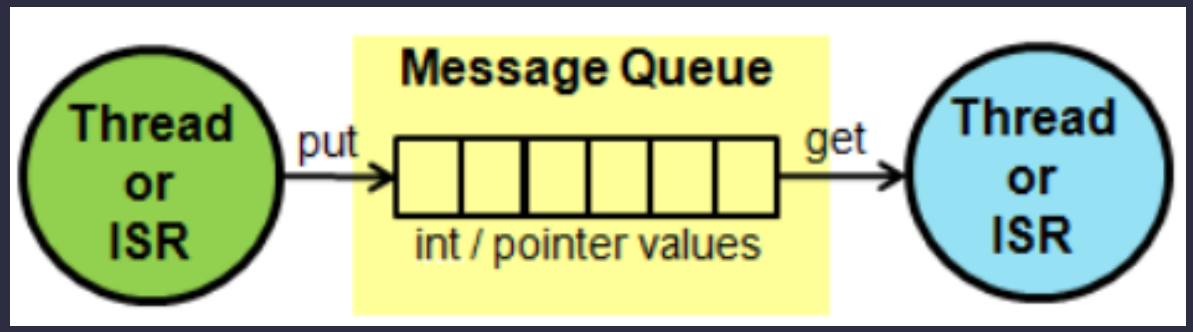
- Required so that threads and tasks can execute critical code and to guarantee access in a particular order
- Semaphores — synchronization and exclusion
  - Mutexes
    - Conditional variables — exclusion based on a condition Event flags
    - Signals —asynchronous event processing and exception handling



System Design with Sensors II - FreeRTOS

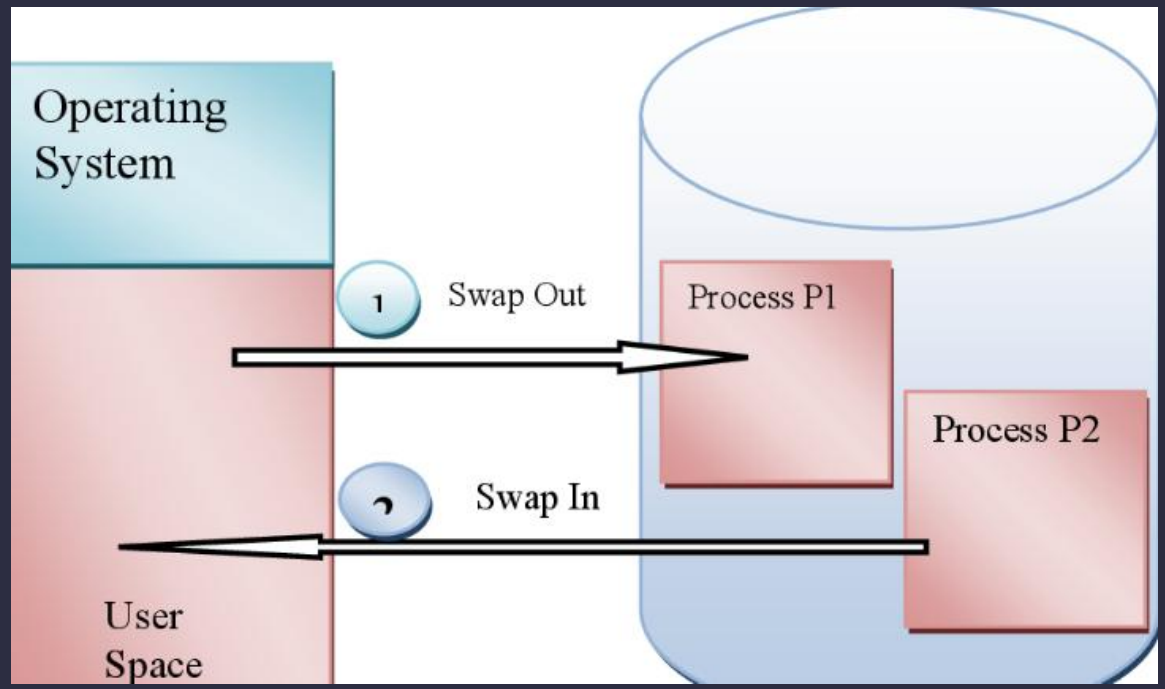
Communications

- Data may be exchanged between threads and tasks via
  - Queues — multiple messages



System Design with Sensors II - FreeRTOS

- Static memory allocation
  - All memory allocated to each process at system startup
- Dynamic memory allocation
  - Nemory requests are made at runtime
  - Should know what to do upon allocation failure
  - Some RTOSs support a timeout function



System Design with Sensors II - FreeRTOS

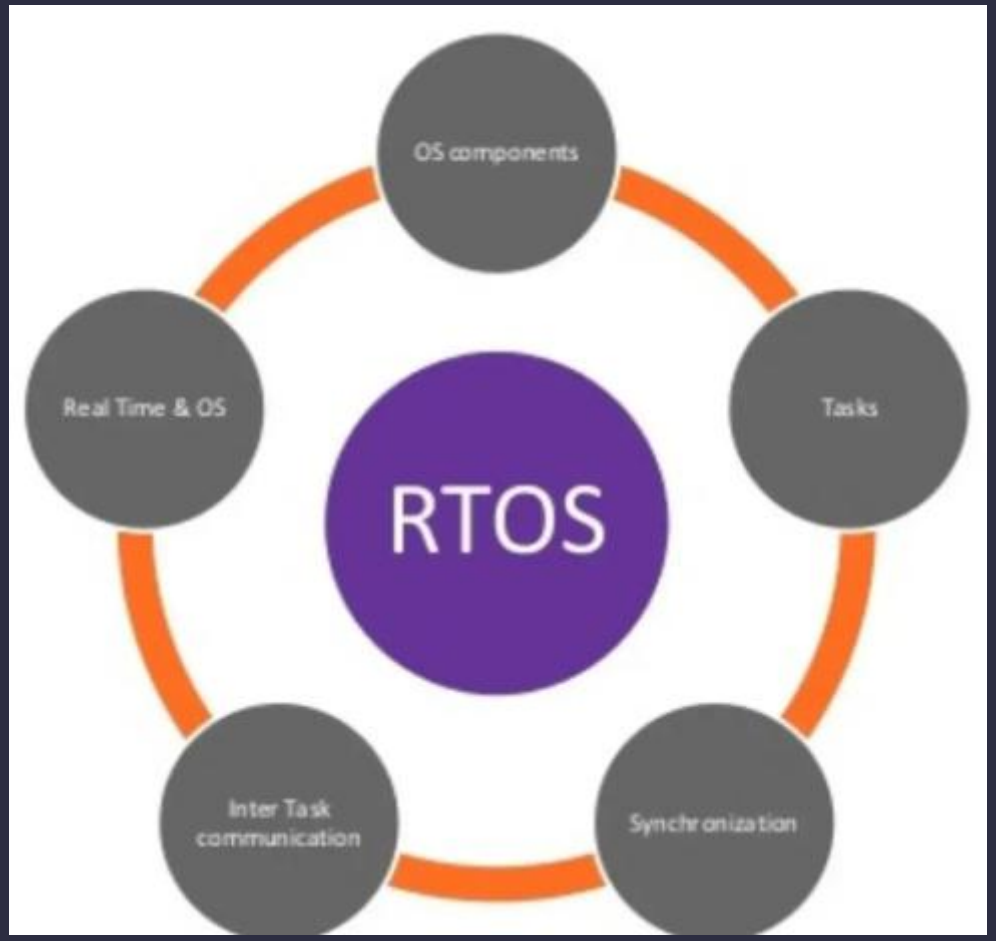
Real Time Operating System (RTOS):

Processes

- Data
- Events

that have critically defined time constraints.

A system where correctness depends not only on the correctness of the logical result of the computation, but also on the result delivery time



System Design with Sensors II - FreeRTOS

FreeRTOS

Developed in partnership with the world's leading chip companies over an 18-year period, and now downloaded every 170 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 IoT libraries suitable for use across all industry sectors.

FreeRTOS is built with an emphasis on reliability and ease of use.

System Design with Sensors II - FreeRTOS

FreeRTOS Highligts

- Provides a single and independent solution for many different architectures and development tools.
- Is known to be reliable. Confidence is assured by the activities undertaken by the SafeRTOS sister project.
- Has a minimal ROM, RAM and processing overhead. Typically an RTOS kernel binary image will be in the region of 6K to 12K bytes.
- Is very simple - the core of the RTOS kernel is contained in only 3 C files. The majority of the many files included in the .zip file download relate only to the numerous demonstration applications.
- Is truly free for use in commercial applications (see license conditions for details).

FreeRTOS vs OpenRTOS

	FreeRTOS Kernel MIT License	OpenRTOS Commercial License
Is it free?	Yes	No
Can I use it in a commercial application?	Yes	Yes
Is it royalty free?	Yes	Yes
Is a warranty provided?	No	**Yes**
Can I receive professional technical support on a commercial basis?	Yes	Yes
Is legal protection provided?	No	**Yes**, IP infringement protection is provided
Do I have to open source my application code that makes use of the FreeRTOS services?	No	No
Do I have to open source my changes to the RTOS kernel?	No	No
Do I have to document that my product uses FreeRTOS?	No	No
Do I have to offer to provide the FreeRTOS code to users of my application?	No	No

System Design with Sensors II - FreeRTOS

SAFERTOS is a derivative version of the FreeRTOS kernel that has been

- Analyzed
- Documented
- Tested

to meet the stringent requirements of industrial (IEC 61508 SIL 3), medical (IEC 62304 and FDA 510(K)) automotive (ISO 26262) and other international safety standards.

SafeRTOS includes independently audited safety lifecycle documentation artifacts. SafeRTOS is provided by WITTENSTEIN high integrity systems.

System Design with Sensors II - FreeRTOS

Windriver VxWorks

Proven in the most challenging safety-critical applications, VxWorks makes it easier and more cost-effective for technology suppliers to meet the stringent safety certification requirements of

- EN 50128
- IEC 61508
- ISO 26262
- DO-178C
- ED-12C

VxWorks® 653 Multi-core Edition is a safe, secure, and reliable real-time operating system (RTOS). It delivers an ARINC 653–conformant system by providing robust time and space partitioning on the latest hardware platforms to ensure fault containment and the ability to upgrade applications with minimal test and integration demands.

System Design with Sensors II - FreeRTOS

FreeRTOS Naming Conventions

- Function names use camel case, are unambiguously descriptive, and use full words (no abbreviations, except universally accepted acronyms).
- File scope static (private) functions are prefixed with prv.
- API functions are prefixed with their return type, as per the convention defined for variables with the addition of the prefix v for void.
- API function names start with the name of the file in which they are defined. For example vTaskDelete is defined in tasks.c, and has a void return type.

System Design with Sensors II - FreeRTOS

FreeRTOS Naming Conventions

- Download From:
https://www.freertos.org/

Contents:

- Task.c
- Queue.c
- List.c
- Heap.c
- Croutine.c
- Port.c

System Design with Sensors II - FreeRTOS

Frequently using functions on RTOS

```
#include “FreeRTOS.h”
#include “task.h”

BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
 const char * const pcName,
 unsigned short usStackDepth,
 void *pvParameters,
 UBaseType_t uxPriority,
 TaskHandle_t *pxCreatedTask );
```

System Design with Sensors II - FreeRTOS

Frequently using functions on RTOS

```
#include “FreeRTOS.h”
#include “task.h”
void vTaskDelay( TickType_t xTicksToDelay );
```

Places the task that calls vTaskDelay() into the Blocked state for a fixed number of tick interrupts.

System Design with Sensors II - FreeRTOS

References

- https://www.freertos.org/
- https://www.freertos.org/Documentation/RTOS_book.html
- https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf
- https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

System Design with Sensors II - FreeRTOS

Basic Arduino FreeRTOS Tasks

- Thread Start

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

#define mainDELAY_LOOP_COUNT 0xffff

void vTask1(void *pvParameters) {
  (void) pvParameters;
  const char *pcTaskName = "Task 1 is running\r\n";
  volatile unsigned long ul;
  for (;;) {
    Serial.print(pcTaskName);
    for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) {}
  }
}

void vTask2(void *pvParameters) {
  (void) pvParameters;
  const char *pcTaskName = "Task 2 is running\r\n";
  volatile unsigned long ul;
  for (;;) {
    Serial.print(pcTaskName);
    for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) {}
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTask1, "Task 1", 128, NULL, 1, NULL);
  xTaskCreate(vTask2, "Task 2", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II - FreeRTOS

Parameterized Dual Task Example

- Dual Tasks
- Parameter Passing

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

#define mainDELAY_LOOP_COUNT (0xffff)

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vTaskFunction(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  volatile unsigned long ul;
  for (;;) {
    Serial.print(pcTaskName);
    for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) { }
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 1, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Multiple Parameter Passing in FreeRTOS Tasks

- Parameter Struct
- Dual Task Instances
- String, Int, Float

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

#define mainDELAY_LOOP_COUNT (0xffff)

typedef struct {
  const char* taskName;
  int someValue;
  float anotherValue;
} TaskParameters;

TaskParameters taskParams1 = { "Task 1 is running\r\n", 123, 3.14 };
TaskParameters taskParams2 = { "Task 2 is running\t\n", 456, 6.28 };

void vTaskFunction(void *pvParameters) {
  TaskParameters* params = (TaskParameters*) pvParameters;
  volatile unsigned long ul;
  for (;;) {
    Serial.print(params->taskName);
    Serial.print(" Value: ");
    Serial.print(params->someValue);
    Serial.print(" Another: ");
    Serial.println(params->anotherValue);
    for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) {}
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void*)&taskParams1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void*)&taskParams2, 1, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Priority

- Stuck at Task 2
- Predefined Tick Time Context Switching
- Port Macro Header

```
/* Architecture specifics. */
#define portSTACK_GROWTH            ( -1 )
#define portSWITCH_INT_NUMBER       0x80
#define portYIELD()                         __asm{ int portSWITCH_INT_NUMBER }
#define portDOS_TICK_RATE           ( 18.20648 )
#define portTICK_RATE_MS      ( ( portTickType ) 1000 / configTICK_RATE_HZ )
#define portTICKS_PER_DOS_TICK      ( ( unsigned portSHORT ) ( ( ( portDOUBLE ) configTICK_RATE_HZ / portDOS_TICK_RATE ) + 0.5 ) )
#define portINITIAL_SW              ( ( portSTACK_TYPE ) 0x0202 )   /* Start the tasks with interrupts enabled. */
/*-----------------------------------------------------------*/
```

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

#define mainDELAY_LOOP_COUNT (0xffff)

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vTaskFunction(void *pvParameters) {
  char *pcTaskName = (char*) pvParameters;
  volatile unsigned long ul;
  for (;;) {
    Serial.print(pcTaskName);
    for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++) {}
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 2, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Priority fix with vTaskDelay

- vTaskDelay

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vTaskFunction(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  for (;;) {
    Serial.print(pcTaskName);
    vTaskDelay(250 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void *)pcTextForTask1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void *)pcTextForTask2, 2, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

vTaskDelayUntil

- Periodic Execution
- vTaskDelayUntil Usage
- Dual Tasks

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vTaskFunction(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  TickType_t xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    Serial.print(pcTaskName);
    vTaskDelayUntil(&xLastWakeTime, (250 / portTICK_PERIOD_MS));
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void *)pcTextForTask1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void *)pcTextForTask2, 2, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

3 Tasks

- Continuous Tasks
- Periodic Task
- DelayUntil
- Priority Levels

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

const char *pcTextForTask1 = "Continuous Task 1 is running\r\n";
const char *pcTextForTask2 = "Continuous Task 2 is running\r\n";
const char *pcTextForPeriodicTask = "Periodic task is running\r\n";

void vContinuousProcessingTask(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  for (;;) {
    Serial.print(pcTaskName);
  }
}

void vPeriodicTask(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    Serial.print(pcTextForPeriodicTask);
    vTaskDelayUntil(&xLastWakeTime, (1000 / portTICK_PERIOD_MS));
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vContinuousProcessingTask, "Task 1", 1000, (void *)pcTextForTask1, 1,
NULL);
  xTaskCreate(vContinuousProcessingTask, "Task 2", 1000, (void *)pcTextForTask2, 1,
NULL);
  xTaskCreate(vPeriodicTask, "Task 3", 1000, (void *)pcTextForPeriodicTask, 2, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Atomic Serial Printing with Critical Sections

- Atomic Print
- Critical Section
- No Interleaving
- Lightweight Synchronization
- Real-Time Consistency

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

const char *pcTextForTask1 = "Continuous Task 1 is running\r\n";
const char *pcTextForTask2 = "Continuous Task 2 is running\r\n";
const char *pcTextForPeriodicTask = "Periodic task is running\r\n";

void vContinuousProcessingTask(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  for (;;) {
    taskENTER_CRITICAL();
    Serial.print(pcTaskName);
    taskEXIT_CRITICAL();
  }
}

void vPeriodicTask(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  for (;;) {
    taskENTER_CRITICAL();
    Serial.print(pcTextForPeriodicTask);
    taskEXIT_CRITICAL();
    vTaskDelayUntil(&xLastWakeTime, (1000 / portTICK_PERIOD_MS));
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vContinuousProcessingTask, "Task 1", 1000, (void *)pcTextForTask1, 1,
NULL);
  xTaskCreate(vContinuousProcessingTask, "Task 2", 1000, (void *)pcTextForTask2, 1,
NULL);
  xTaskCreate(vPeriodicTask, "Task 3", 1000, (void *)pcTextForPeriodicTask, 2, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Idle Hook

- Idle Hook Counter
- Dual Tasks
- Periodic Execution
- Serial Debug Output

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

static unsigned long ulIdleCycleCount = 0UL;

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vPrintStringAndNumber(const char *pcString, unsigned long ulNumber) {
  Serial.print(pcString);
  Serial.println(ulNumber);
}

void vTaskFunction(void *pvParameters) {
  char *pcTaskName = (char *)pvParameters;
  for (;;) {
    vPrintStringAndNumber(pcTaskName, ulIdleCycleCount);
    vTaskDelay(250 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTaskFunction, "Task 1", 1000, (void *)pcTextForTask1, 1, NULL);
  xTaskCreate(vTaskFunction, "Task 2", 1000, (void *)pcTextForTask2, 2, NULL);
}

void loop() { }

void vApplicationIdleHook(void) {
  ulIdleCycleCount++;
}
```

System Design with Sensors II - FreeRTOS

Dynamic Task Priority Adjustment

- Dynamic Priority
- Task Coordination
- Preemption
- Real-Time Scheduling

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

TaskHandle_t xTask2Handle;

const char *pcTextForTask1 = "Task 1 is running\r\n";
const char *pcTextForTask2 = "Task 2 is running\t\n";

void vTask1(void *pvParameters) {
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet(NULL);
  for (;;) {
    Serial.print("Task1 is running\r\n");
    Serial.print("About to raise the Task2 priority\r\n");
    vTaskPrioritySet(xTask2Handle, (uxPriority + 1));
  }
}

void vTask2(void *pvParameters) {
  UBaseType_t uxPriority;
  uxPriority = uxTaskPriorityGet(NULL);
  for (;;) {
    Serial.print("Task2 is running\r\n");
    Serial.print("About to lower the Task2 priority\r\n");
    vTaskPrioritySet(NULL, 0);
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTask1, "Task 1", 1000, (void *)pcTextForTask1, 2, NULL);
  xTaskCreate(vTask2, "Task 2", 1000, (void *)pcTextForTask2, 1, &xTask2Handle);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Dynamic Task Creation & Self-Deletion

- Dynamic Creation
- Task Self-Deletion
- Priority Management
- Preemptive Scheduling

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>

TaskHandle_t xTask2Handle;

void vTask1(void *pvParameters) {
  const TickType_t xDelay100ms = 100 / portTICK_PERIOD_MS;
  for (;;) {
    Serial.print("Task1 is running\r\n");
    xTaskCreate(vTask2, "Task 2", 1000, NULL, 2, &xTask2Handle);
    vTaskDelay(xDelay100ms);
  }
}

void vTask2(void *pvParameters) {
  Serial.print("Task2 is running and about to delete itself\r\n");
  vTaskDelete(xTask2Handle);
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(vTask1, "Task 1", 1000, NULL, 1, NULL);
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Dynamic Queue Communication

- Queue-Based Messaging
- Multiple Sender Tasks
- Single Receiver Task
- Priority
- Task Yield: Context Switch

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#include <queue.h>

QueueHandle_t xQueue;

static void vSenderTask(void *pvParameters) {
  long lValueToSend = (long) pvParameters;
  BaseType_t xStatus;
  for (;;) {
    xStatus = xQueueSendToBack(xQueue, &lValueToSend, 0);
    if (xStatus != pdPASS) {
      Serial.print("Could not send to the queue.\r\n");
    }
    taskYIELD();
  }
}

static void vReceiverTask(void *pvParameters) {
  long lReceivedValue;
  BaseType_t xStatus;
  const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;
  for (;;) {
    if (uxQueueMessagesWaiting(xQueue) != 0) {
      Serial.print("Queue should have been empty!\r\n");
    }
    xStatus = xQueueReceive(xQueue, &lReceivedValue, xTicksToWait);
    if (xStatus == pdPASS) {
      Serial.print("Received = ");
      Serial.println(lReceivedValue);
    } else {
      Serial.print("Could not receive from the queue.\r\n");
    }
  }
}
```

```
void setup() {
  Serial.begin(9600);
  while(!Serial) {}

  xQueue = xQueueCreate(5, sizeof(long));
  if (xQueue != NULL) {
    xTaskCreate(vSenderTask, "Sender1",
1000, (void *)100, 1, NULL);
    xTaskCreate(vSenderTask, "Sender2",
1000, (void *)200, 1, NULL);
    xTaskCreate(vReceiverTask, "Receiver",
1000, NULL, 2, NULL);
  }
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

FreeRTOS Queue Communication

- Multiple Senders
- Single Receiver

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#include <queue.h>

#define mainSENDER_1 1
#define mainSENDER_2 2

typedef struct {
  unsigned char ucValue;
  unsigned char ucSource;
} xData;

static const xData xStructsToSend[2] = {
  { 100, mainSENDER_1 },
  { 200, mainSENDER_2 }
};

QueueHandle_t xQueue;

void vPrintString(const char *s) {
  Serial.print(s);
}

void vPrintStringAndNumber(const char *s, long n) {
  Serial.print(s);
  Serial.println(n);
}

static void vSenderTask(void *pvParameters) {
  BaseType_t xStatus;
  const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;
  for (;;) {
    xStatus = xQueueSendToBack(xQueue, pvParameters,
xTicksToWait);
    if (xStatus != pdPASS) {
      vPrintString("Could not send to the queue.\r\n");
    }
    taskYIELD();
  }
}
```

```
static void vReceiverTask(void *pvParameters) {
  xData xReceivedStructure;
  BaseType_t xStatus;
  for (;;) {
    if (uxQueueMessagesWaiting(xQueue) != 3) {
      vPrintString("Queue should have been
full!\r\n");
    }
    xStatus = xQueueReceive(xQueue,
&xReceivedStructure, 0);
    if (xStatus == pdPASS) {
      if (xReceivedStructure.ucSource ==
mainSENDER_1) {
        vPrintStringAndNumber("From Sender 1 = ",
xReceivedStructure.ucValue);
      } else {
        vPrintStringAndNumber("From Sender 2 = ",
xReceivedStructure.ucValue);
      }
    } else {
      vPrintString("Could not receive from the
queue.\r\n");
    }
  }
}

void setup() {
  Serial.begin(9600);
  while (!Serial) { }
  xQueue = xQueueCreate(3, sizeof(xData));
  if (xQueue != NULL) {
    xTaskCreate(vSenderTask, "Sender1", 1000,
(void *)&xStructsToSend[0], 2, NULL);
    xTaskCreate(vSenderTask, "Sender2", 1000,
(void *)&xStructsToSend[1], 2, NULL);
    xTaskCreate(vReceiverTask, "Receiver", 1000,
NULL, 1, NULL);
    vTaskStartScheduler();
  }
}

void loop() { }
```

System Design with Sensors II - FreeRTOS

Semaphore

- Binary Semaphore
- ISR Simulation
- Periodic Trigger

Program Arduino

Open Putty

```
#include <Arduino_FreeRTOS.h>
#include <task.h>
#include <semphr.h>

SemaphoreHandle_t xBinarySemaphore;

void vHandlerTask(void *pvParameters) {
  for (;;) {
    xSemaphoreTake(xBinarySemaphore, portMAX_DELAY);
    Serial.print("Handler task - Processing event.\r\n");
  }
}

static void vExampleInterruptHandler(void) {
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  xSemaphoreGiveFromISR(xBinarySemaphore,
&xHigherPriorityTaskWoken);
  if(xHigherPriorityTaskWoken == pdTRUE) {
    portYIELD_FROM_ISR();
  }
}


void loop() { }
```

```
void vPeriodicTask(void *pvParameters) {
  for (;;) {
    vTaskDelay(500 / portTICK_PERIOD_MS);
    Serial.print("Periodic task - About to
generate an interrupt.\r\n");
    vExampleInterruptHandler();
    Serial.print("Periodic task - Interrupt
generated.\r\n\r\n\r\n");
  }
}

void setup() {
  Serial.begin(9600);
  while (!Serial) { }
  xBinarySemaphore = xSemaphoreCreateBinary();
  if(xBinarySemaphore != NULL) {
    xTaskCreate(vHandlerTask, "Handler", 1000,
NULL, 3, NULL);
    xTaskCreate(vPeriodicTask, "Periodic", 1000,
NULL, 1, NULL);
    vTaskStartScheduler();
  }
}
```

System Design with Sensors II – FreeRTOS

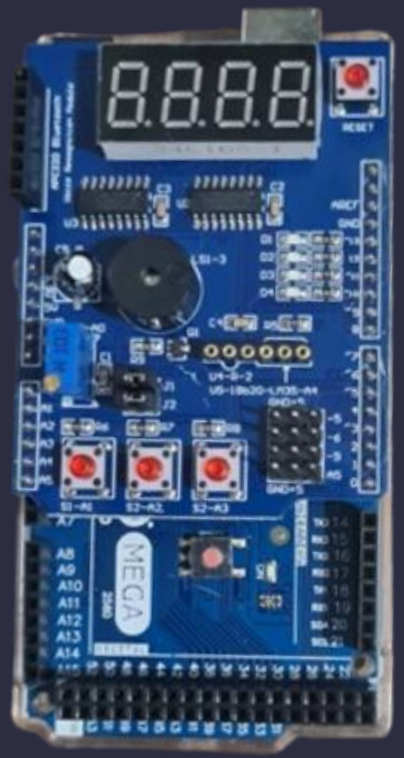
How to Build an Embedded System?

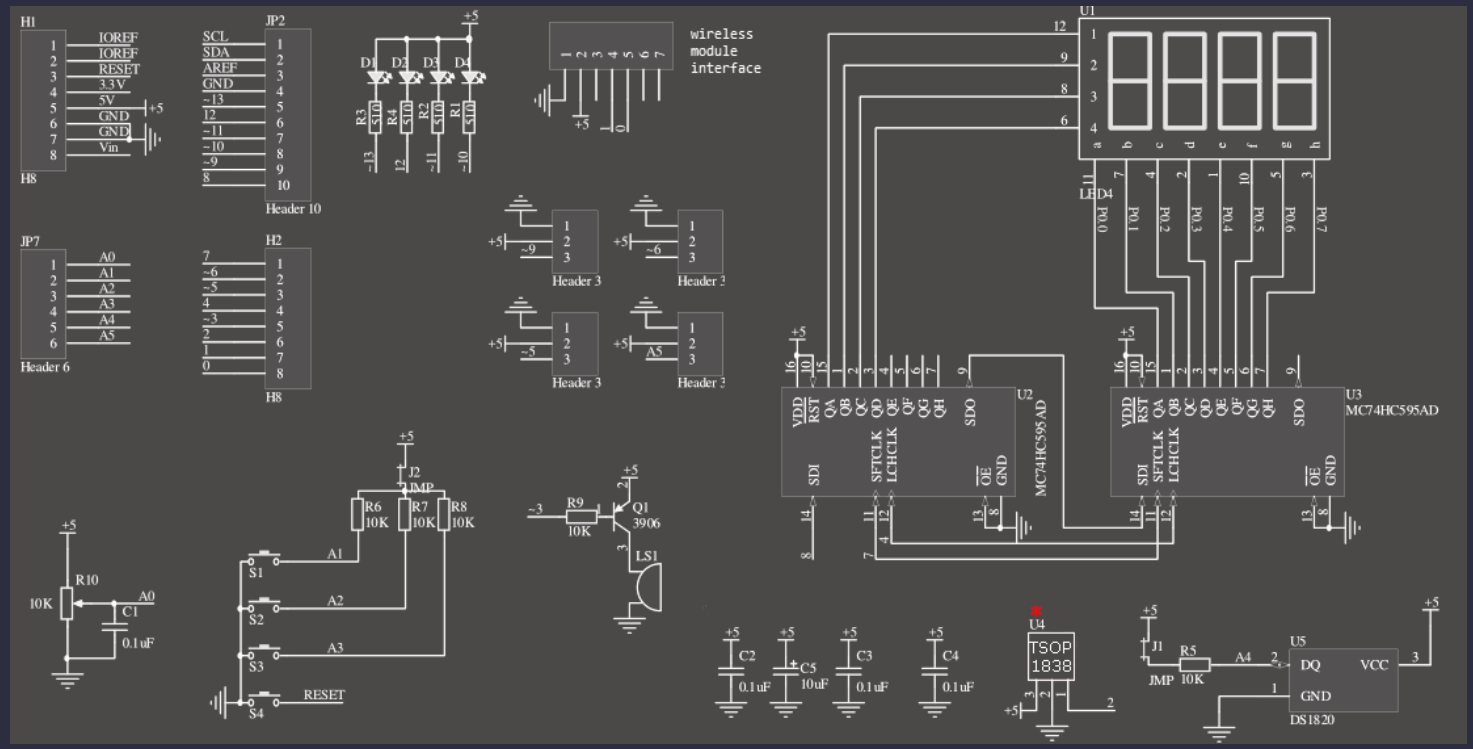
- Buttons
- Sensors
  - Temperature
  - IMU
  - GPS
  - …
- Communication Interfaces
  - UART
  - SPI
  - I2C
  - Ethernet
  - …
- RF Transceiver

- LEDs
- Motors
- Communication Interfaces
  - UART
  - SPI
  - I2C
  - Ethernet
  - …
- RF Transceiver

System Design with Sensors II – FreeRTOS

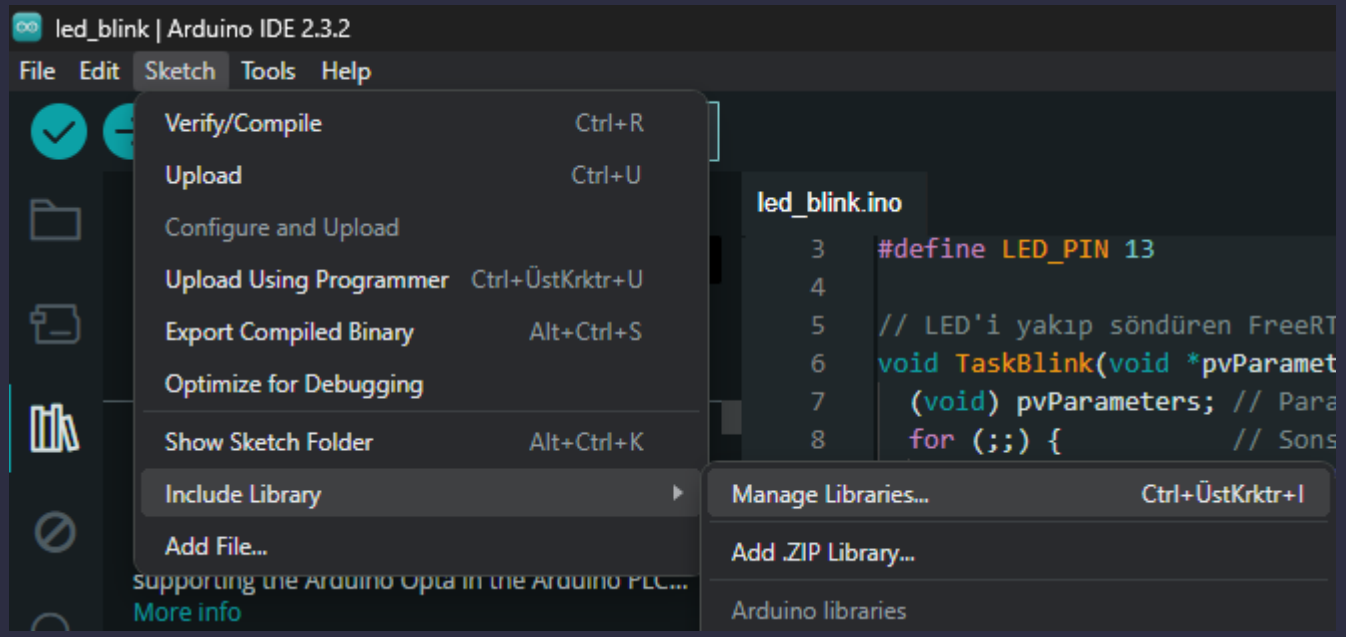
Arduino Mega and Shield Connection





System Design with Sensors II – FreeRTOS

FreeRTOS Setup on Arduino IDE



System Design with Sensors II – FreeRTOS

Simple Output

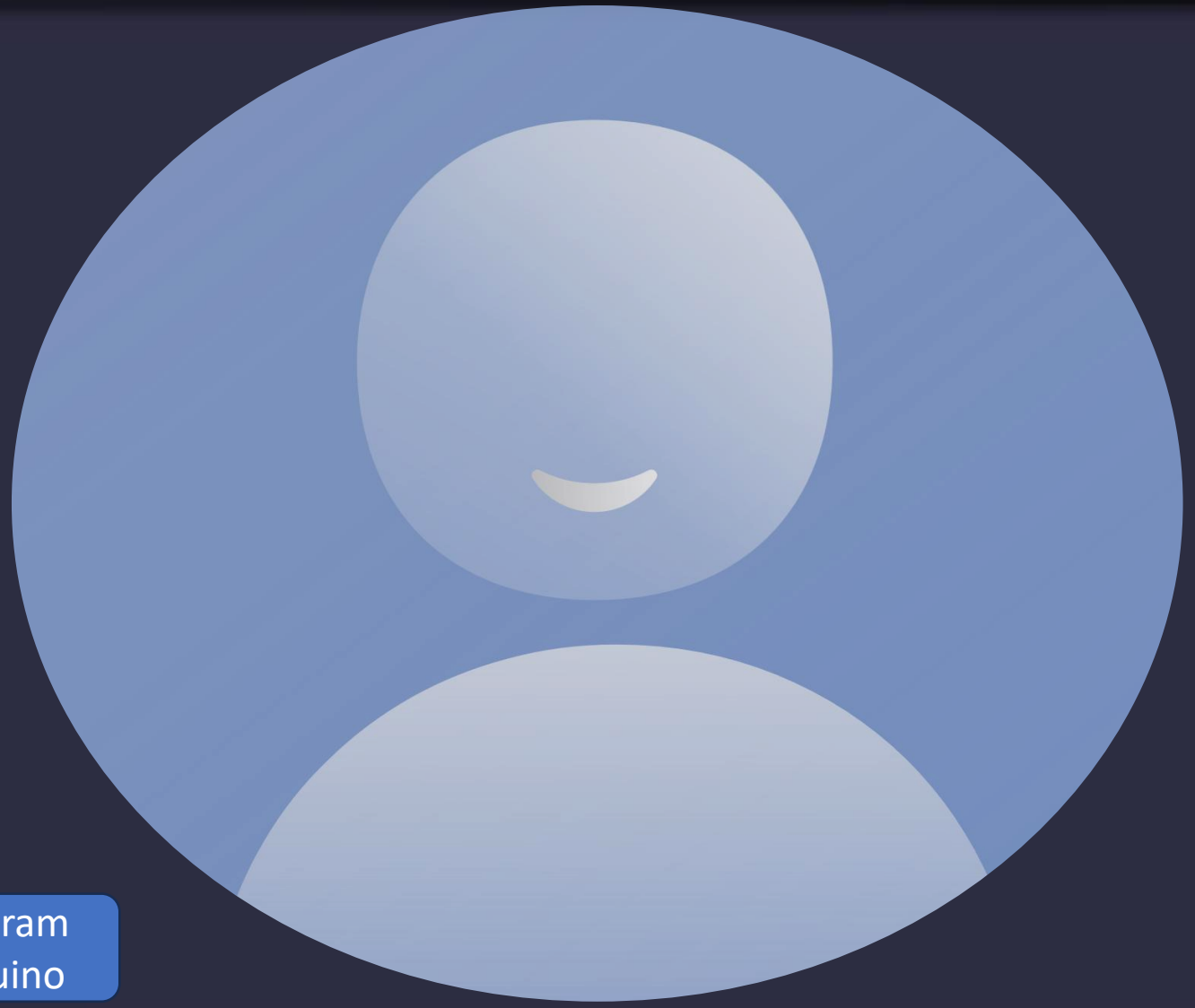
```
#define LED_PIN 13

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(250);
  digitalWrite(LED_PIN, LOW);
  delay(250);
}
```

```
#include <Arduino_FreeRTOS.h>

#define LED_PIN 13
void TaskBlink(void *pvParameters) {
  (void) pvParameters; // Parametre kullanılmıyor
  for (;;) {           // Sonsuz döngü, görevin temel yapısı
    digitalWrite(LED_PIN, HIGH);
    vTaskDelay(500 / portTICK_PERIOD_MS); // 500 ms bekle
    digitalWrite(LED_PIN, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS); // 500 ms bekle
  }
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  xTaskCreate(
    TaskBlink,       // Görev fonksiyonu
    "Blink",         // Görev adı
    128,             // Yığın boyutu (bytes)
    NULL,            // Parametre
    1,               // Öncelik (düşük öncelikli görev)
    NULL             // Görev tutamacı (kullanılmıyor)
  );
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Simple Output

```
#include <Arduino_FreeRTOS.h>
#define LED_PIN 13
void TaskBlink(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    digitalWrite(LED_PIN, HIGH);
    vTaskDelay(500 / portTICK_PERIOD_MS);
    digitalWrite(LED_PIN, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  xTaskCreate(
    TaskBlink,
    "Blink",
    128,
    NULL,
    1,
    NULL
  );
}

void loop() {
}
```



Program Arduino

System Design with Sensors II – FreeRTOS

Simple Output

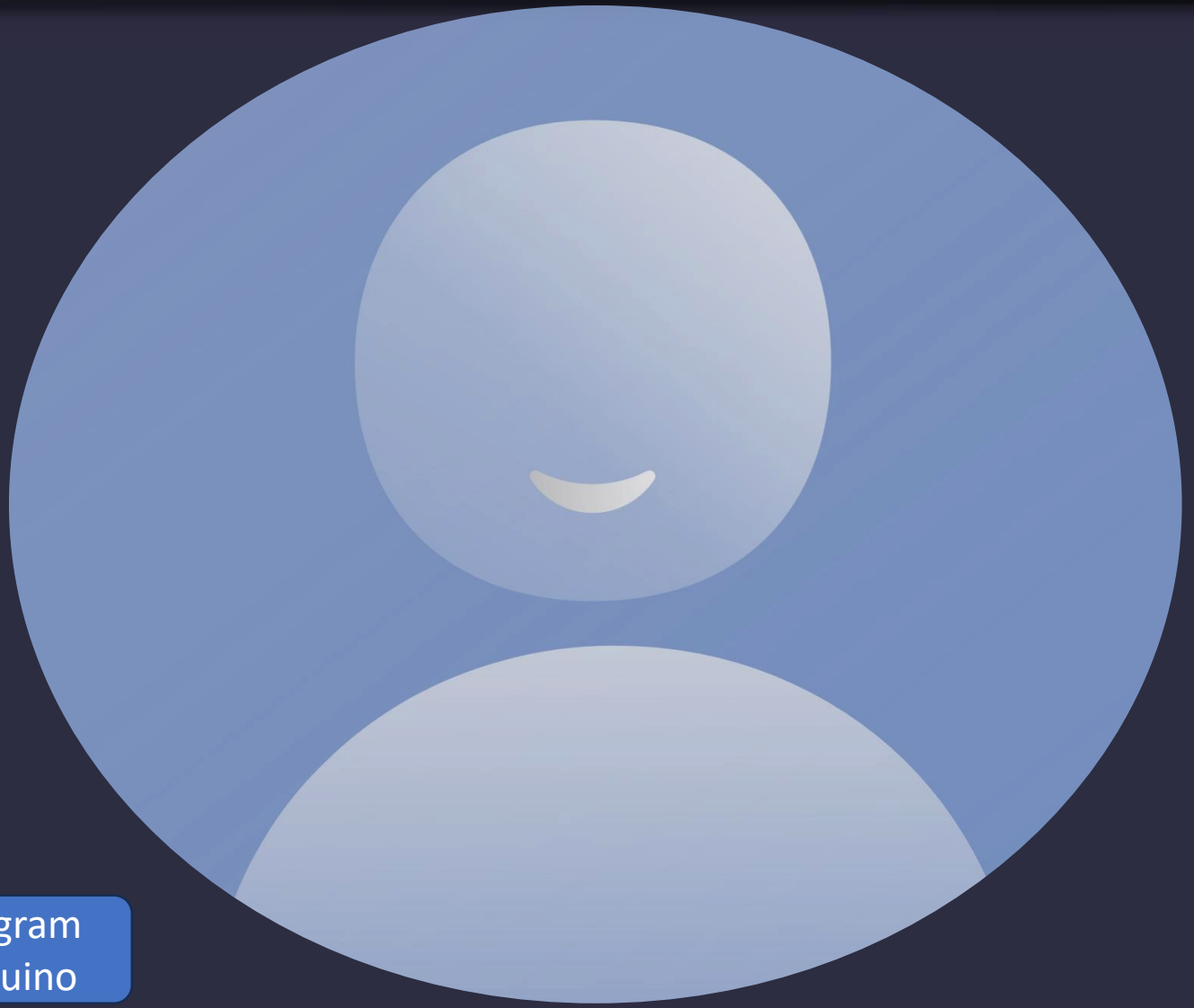
```
#define LED_PIN 13

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  digitalWrite(LED_PIN, LOW);
  delay(1000);
}
```

```
#include <Arduino_FreeRTOS.h>

#define LED_PIN 13
void TaskBlink(void *pvParameters) {
  (void) pvParameters; // Parametre kullanılmıyor
  for (;;) {           // Sonsuz döngü, görevin temel yapısı
    digitalWrite(LED_PIN, HIGH);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    digitalWrite(LED_PIN, LOW);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
  xTaskCreate(
    TaskBlink,       // Görev fonksiyonu
    "Blink",         // Görev adı
    128,             // Yığın boyutu (bytes)
    NULL,            // Parametre
    1,               // Öncelik (düşük öncelikli görev)
    NULL             // Görev tutamacı (kullanılmıyor)
  );
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Simple Output

```
#include <Arduino_FreeRTOS.h>

#define LED_PIN 13
void TaskBlink(void *pvParameters) {
  (void) pvParameters; // Parametre kullanılmıyor
  for (;;) {           // Sonsuz döngü, görevin temel yapısı
    digitalWrite(LED_PIN, HIGH);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    digitalWrite(LED_PIN, LOW);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
    xTaskCreate(
    TaskBlink,       // Görev fonksiyonu
    "Blink",         // Görev adı
    128,             // Yığın boyutu (bytes)
    NULL,            // Parametre
    1,               // Öncelik (düşük öncelikli görev)
    NULL             // Görev tutamacı (kullanılmıyor)
  );
}

void loop() {
}
```



Program Arduino

System Design with Sensors II – FreeRTOS

Simple Input & Output

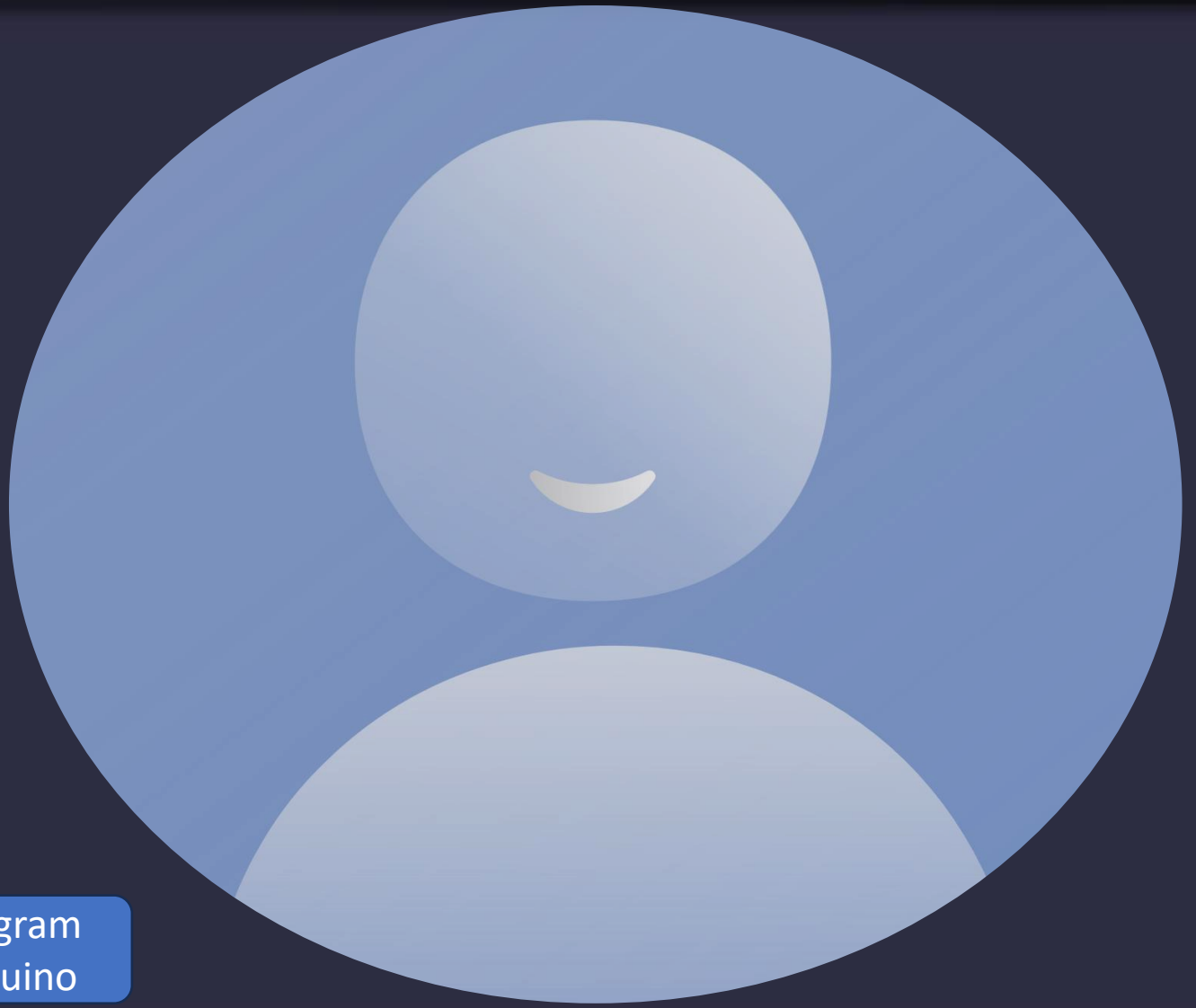
```
int ledPin = 13;
int buttonPin = A1;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  static bool ledDurum = LOW;
  if (digitalRead(buttonPin) == LOW) {
    ledDurum = !digitalRead(ledPin);
    digitalWrite(ledPin, ledDurum);
    delay(200);
  }
}
```

```
#include <Arduino_FreeRTOS.h>

int ledPin = 13;
int buttonPin = A1;

void TaskButtonLED(void *pvParameters) {
  (void) pvParameters;

  static bool ledDurum = LOW;

  for (;;) {
    if (digitalRead(buttonPin) == LOW) {
      ledDurum = !digitalRead(ledPin);
      digitalWrite(ledPin, ledDurum);
      vTaskDelay(200 / portTICK_PERIOD_MS);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);

  // FreeRTOS görev oluşturma
  xTaskCreate(
    TaskButtonLED,       // Görev fonksiyonu
    "ButtonLED",         // Görev adı
    128,                 // Stack (yığın) boyutu
    NULL,                // Parametre (kullanılmıyor)
    1,                   // Görev önceliği
    NULL                 // Görev tutamacı (kullanılmıyor)
  );

}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Simple Input & Output

```
#include <Arduino_FreeRTOS.h>

int ledPin = 13;
int buttonPin = A1;

void TaskButtonLED(void *pvParameters) {
  (void) pvParameters;

  static bool ledDurum = LOW;

  for (;;) {
    if (digitalRead(buttonPin) == LOW) {
      ledDurum = !digitalRead(ledPin);
      digitalWrite(ledPin, ledDurum);
      vTaskDelay(200 / portTICK_PERIOD_MS);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);

  // FreeRTOS görev oluşturma
  xTaskCreate(
    TaskButtonLED,       // Görev fonksiyonu
    "ButtonLED",         // Görev adı
    128,                 // Stack (yığın) boyutu
    NULL,                // Parametre (kullanılmıyor)
    1,                   // Görev önceliği
    NULL
  );

}

void loop() {
}
```



Program Arduino

System Design with Sensors II – FreeRTOS

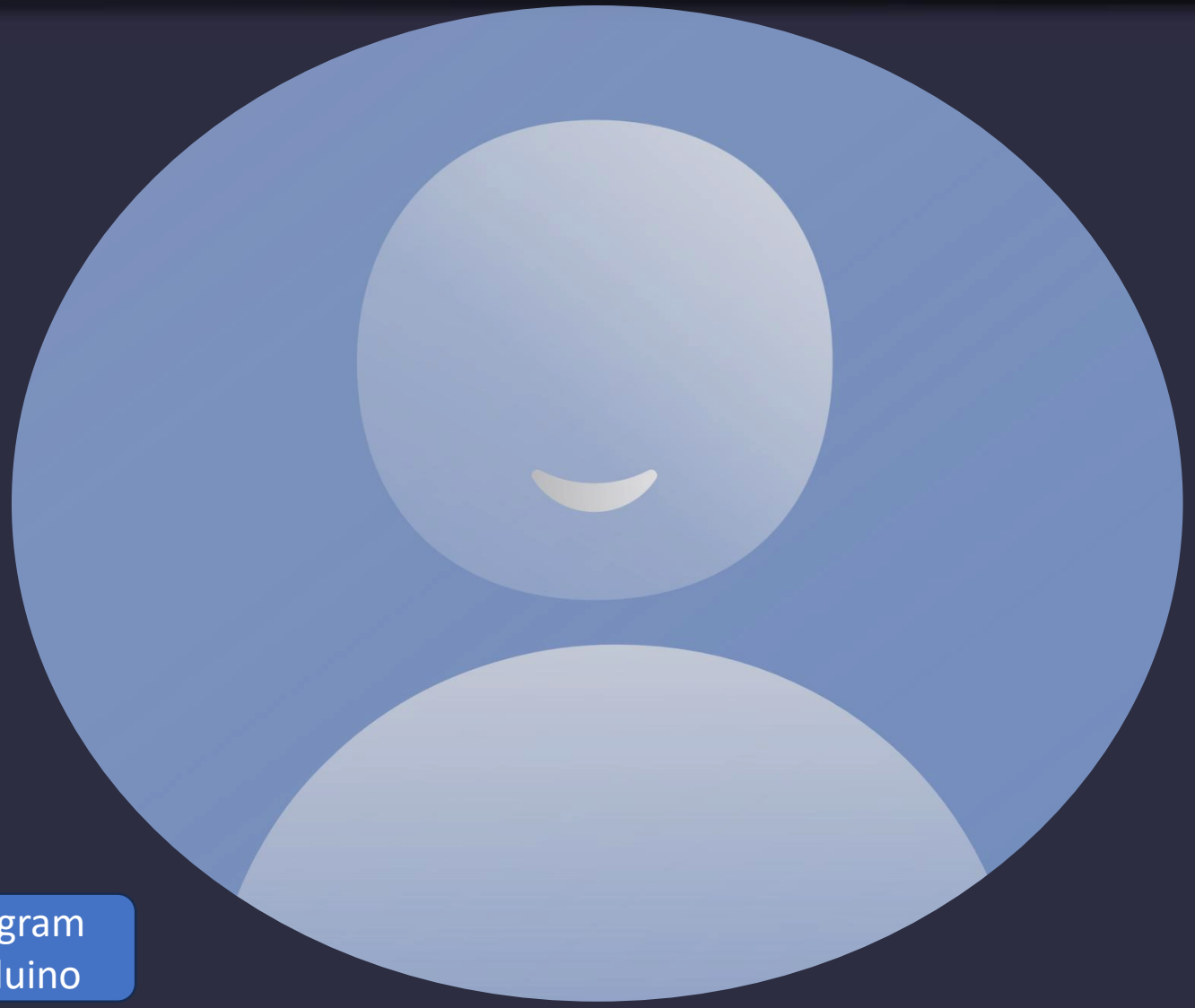
Simple Input & Output

```
#include <Arduino_FreeRTOS.h>

// Donanım pin tanımları
const int ledPinBlink = 13;
const int ledPinButton = 12;
const int buttonPin    = A1;

void TaskBlink(void *pvParameters) {
  (void) pvParameters; // Parametre kullanılmıyor
  pinMode(ledPinBlink, OUTPUT);

  for (;;) {
    digitalWrite(ledPinBlink, HIGH);
    vTaskDelay(500 / portTICK_PERIOD_MS);  // 500ms bekle
    digitalWrite(ledPinBlink, LOW);
    vTaskDelay(500 / portTICK_PERIOD_MS);  // 500ms bekle
  }
}
```

```
void TaskButton(void *pvParameters) {
  (void) pvParameters;
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(ledPinButton, OUTPUT);


  static bool ledDurum = LOW;

  for (;;) {
    if (digitalRead(buttonPin) == LOW) {
      ledDurum = !ledDurum;
      digitalWrite(ledPinButton, ledDurum);

      vTaskDelay(200 / portTICK_PERIOD_MS);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
void setup() {
  // 1. Blink görevi oluştur
  xTaskCreate(
    TaskBlink,          // Görev fonksiyonu
    "BlinkLED",         // Görev adı (debug için)
    128,                // Stack boyutu (bayt cinsinden)
    NULL,               // Parametre (kullanılmıyor)
    1,                  // Görev önceliği
    NULL                // Görev tutamacı (kullanılmıyor)
  );

  xTaskCreate(
    TaskButton,
    "ButtonTask",
    128,
    NULL,
    1,
    NULL
  );

}
void loop() {
}
```

System Design with Sensors II – FreeRTOS

Simple Input & Output



Program Arduino

System Design with Sensors II – FreeRTOS

UART

```
int ledPin = 13;
int buttonPin = A1;
bool ledState = LOW;
bool lastButtonState = HIGH;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  bool buttonState = digitalRead(buttonPin);

  if (lastButtonState == HIGH && buttonState == LOW) {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
    Serial.println(ledState ? "LED Closed" : "LED Open");
    delay(2000);
  }

  lastButtonState = buttonPin;
  delay(50);
}
```

System Design with Sensors II – FreeRTOS

UART

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

int ledPin = 13;
int buttonPin = A1;
bool ledState = LOW;
bool lastButtonState = HIGH;

QueueHandle_t toggleQueue;

void TaskReadButton(void *pvParameters) {
  (void) pvParameters;
  pinMode(buttonPin, INPUT);
  for (;;) {
    bool buttonState = digitalRead(buttonPin);
    if (lastButtonState == HIGH && buttonState == LOW) {
      bool toggleCommand = true;
      xQueueSend(toggleQueue, &toggleCommand, portMAX_DELAY);
      vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    lastButtonState = buttonState;
    vTaskDelay(50 / portTICK_PERIOD_MS);
  }
}
```

```
void TaskToggleLED(void *pvParameters) {
  (void) pvParameters;
  pinMode(ledPin, OUTPUT);
  for (;;) {
    bool cmd;
    if (xQueueReceive(toggleQueue, &cmd, portMAX_DELAY) == pdTRUE) {
      ledState = !ledState;
      digitalWrite(ledPin, ledState);
      if (ledState) {
        Serial.println("LED Closed");
      } else {
        Serial.println("LED Open");
      }
    }
  }
}

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
  toggleQueue = xQueueCreate(5, sizeof(bool));
  xTaskCreate(TaskReadButton, "ButtonTask", 128, NULL, 1, NULL);
  xTaskCreate(TaskToggleLED, "LEDTask", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

UART

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

Counter

```
int buttonPin = A1;
int buttonState = HIGH;
int lastButtonState = HIGH;
int pressCount = 0;

void setup() {
    pinMode(buttonPin, INPUT_PULLUP);
    Serial.begin(9600);
}

void loop() {
    buttonState = digitalRead(buttonPin);

    if (lastButtonState == HIGH && buttonState == LOW) {
        pressCount++;
        Serial.print("Butona basildi. Sayac: ");
        Serial.println(pressCount);
        delay(200);
    }

    lastButtonState = buttonState;
}
```

System Design with Sensors II – FreeRTOS

Counter

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

int buttonPin = A1;
int buttonState = HIGH;
int lastButtonState = HIGH;
int pressCount = 0;

QueueHandle_t pressCountQueue;

void TaskReadButton(void *pvParameters) {
  (void) pvParameters;
  pinMode(buttonPin, INPUT_PULLUP);
  for (;;) {
    buttonState = digitalRead(buttonPin);
    if (lastButtonState == HIGH && buttonState == LOW) {
      pressCount++;
      xQueueSend(pressCountQueue, &pressCount, portMAX_DELAY);
      vTaskDelay(200 / portTICK_PERIOD_MS);
    }
    lastButtonState = buttonState;
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

```
void TaskPrintSerial(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    int count;
    if (xQueueReceive(pressCountQueue, &count, portMAX_DELAY) == pdTRUE) {
      Serial.print("Butona basildi. Sayac: ");
      Serial.println(count);
    }
  }
}

void setup() {
  Serial.begin(9600);
  pressCountQueue = xQueueCreate(5, sizeof(int));
  xTaskCreate(TaskReadButton, "ReadButton", 128, NULL, 1, NULL);
  xTaskCreate(TaskPrintSerial, "PrintSerial", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Counter

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

UART Command

```
int ledPin = 13;
char receivedChar;

void setup() {
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
    Serial.println("Komut : 'A' = LED Ac, 'K' = LED Kapat");
}

void loop() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();

        if (receivedChar == 'A') {
            digitalWrite(ledPin, LOW);
            Serial.println("LED Acildi");
        }
        else if (receivedChar == 'K') {
            digitalWrite(ledPin, HIGH);
            Serial.println("LED Kapandi");
        }
        else {
            Serial.println("Gecersiz Komut! 'A' veya 'K' girin.");
        }
    }
}
```

System Design with Sensors II – FreeRTOS

UART Command

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

int ledPin = 13;
QueueHandle_t cmdQueue;

void TaskReadSerial(void *pvParameters) {
  for (;;) {
    if (Serial.available() > 0) {
      char c = Serial.read();
      xQueueSend(cmdQueue, &c, portMAX_DELAY);
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

```
void TaskProcessCommand(void *pvParameters) {
  pinMode(ledPin, OUTPUT);
  for (;;) {
    char c;
    if (xQueueReceive(cmdQueue, &c, portMAX_DELAY) == pdTRUE) {
      if (c == 'A') {
        digitalWrite(ledPin, LOW);
        Serial.println("LED Acildi");
      } else if (c == 'K') {
        digitalWrite(ledPin, HIGH);
        Serial.println("LED Kapandi");
      } else {
        Serial.println("Gecersiz Komut! 'A' veya 'K' girin.");
      }
    }
  }
}

void setup() {
  Serial.begin(9600);
  Serial.println("Komut : 'A' = LED Ac, 'K' = LED Kapat");
  cmdQueue = xQueueCreate(10, sizeof(char));
  xTaskCreate(TaskReadSerial, "ReadSerial", 128, NULL, 1, NULL);
  xTaskCreate(TaskProcessCommand, "ProcessCommand", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

UART Command

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

UART Command

```
int ledPins[] = {10, 11, 12, 13};
char receivedChar;
bool ledStates[] = {LOW, LOW, LOW, LOW};

void setup() {
    for (int i = 0; i < 4; i++) {
        pinMode(ledPins[i], OUTPUT);
        digitalWrite(ledPins[i], ledStates[i]);
    }
    Serial.begin(9600);
    Serial.println("1-4 arasinda bir sayi girin.");
}

void loop() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();

        if (receivedChar >= '1' && receivedChar <= '4') {
            int ledIndex = receivedChar - '1';
            ledStates[ledIndex] = !ledStates[ledIndex];
            digitalWrite(ledPins[ledIndex], ledStates[ledIndex]);

            Serial.print("LED ");
            Serial.print(ledIndex + 1);
            Serial.print(" durumu: ");
            Serial.println(ledStates[ledIndex] ? "Acik" : "Kapali");
        }
        else {
            Serial.println("Gecersiz giris! 1-4 arasinda bir sayi girin.");
        }
    }
}
```

System Design with Sensors II – FreeRTOS

UART Command

```
#include <Arduino_FreeRTOS.h>
#include <queue.h>

int ledPins[] = {10, 11, 12, 13};
bool ledStates[] = {LOW, LOW, LOW, LOW};
QueueHandle_t ledIndexQueue;

void TaskReadSerial(void *pvParameters) {
  for (;;) {
    if (Serial.available() > 0) {
      char c = Serial.read();
      if (c >= '1' && c <= '4') {
        int ledIndex = c - '1';
        xQueueSend(ledIndexQueue, &ledIndex, portMAX_DELAY);
      } else {
        Serial.println("Gecersiz giris! 1-4 arasinda bir sayi girin.");
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

```
void TaskProcessLED(void *pvParameters) {
  for (;;) {
    int index;
    if (xQueueReceive(ledIndexQueue, &index, portMAX_DELAY) == pdTRUE) {
      ledStates[index] = !ledStates[index];
      digitalWrite(ledPins[index], ledStates[index]);
      Serial.print("LED ");
      Serial.print(index + 1);
      Serial.print(" durumu: ");
      Serial.println(ledStates[index] ? "Acik" : "Kapali");
    }
  }
}

void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(ledPins[i], OUTPUT);
    digitalWrite(ledPins[i], ledStates[i]);
  }
  Serial.begin(9600);
  Serial.println("1-4 arasinda bir sayi girin.");
  ledIndexQueue = xQueueCreate(10, sizeof(int));
  xTaskCreate(TaskReadSerial, "ReadSerial", 128, NULL, 1, NULL);
  xTaskCreate(TaskProcessLED, "ProcessLED", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

UART Command

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

Buzzer

```
int buttonPins[] = {A1, A2, A3};
int buzzerPin = 3;
int delays[] = {2, 3, 4};

void setup() {
    for (int i = 0; i < 4; i++) {
        pinMode(buttonPins[i], INPUT_PULLUP);
    }
    pinMode(buzzerPin, OUTPUT);
    digitalWrite(buzzerPin, HIGH);
}

void loop() {
    for (int i = 0; i < 3; i++) {
        if (digitalRead(buttonPins[i]) == LOW) {
            for (int j = 0; j < 500 / delays[i]; j++) {
                digitalWrite(buzzerPin, LOW);
                delay(delays[i]);
                digitalWrite(buzzerPin, HIGH);
                delay(delays[i]);
            }
        }
    }
}
```

System Design with Sensors II – FreeRTOS

Buzzer

```
#include <Arduino_FreeRTOS.h>

int buttonPins[] = {A1, A2, A3};
int buzzerPin = 3;
int delaysArr[] = {2, 3, 4};
```

```
void TaskBuzzer(void *pvParameters) {
  (void) pvParameters;
  for (int i = 0; i < 3; i++) {
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, HIGH);

  for (;;) {
    for (int i = 0; i < 3; i++) {
      if (digitalRead(buttonPins[i]) == LOW) {
        //taskENTER_CRITICAL();
         for (int j = 0; j < 500 / delaysArr[i]; j++) {
                digitalWrite(buzzerPin, LOW);
                vTaskDelay(delaysArr[i]);
                digitalWrite(buzzerPin, HIGH);
                vTaskDelay(delaysArr[i]);
            }
       //taskEXIT_CRITICAL();
      }
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

void setup() {
  xTaskCreate(TaskBuzzer, "Buzzer", 128, NULL, 3, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Buzzer

Program Arduino

System Design with Sensors II – FreeRTOS

Buzzer

```
#include <Arduino_FreeRTOS.h>

int buttonPins[] = {A1, A2, A3};
int buzzerPin = 3;
int delaysArr[] = {2, 3, 4};

void beepTone(int delayVal) {
  int cycles = 500 / delayVal;
  for (int j = 0; j < cycles; j++) {
    digitalWrite(buzzerPin, LOW);
    unsigned long start = micros();
    while (micros() - start < delayVal * 1000);
    digitalWrite(buzzerPin, HIGH);
    start = micros();
    while (micros() - start < delayVal * 1000);
  }
}
```

```
void TaskBuzzer(void *pvParameters) {
  (void) pvParameters;
  for (int i = 0; i < 3; i++) {
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
  pinMode(buzzerPin, OUTPUT);
  digitalWrite(buzzerPin, HIGH);
  for (;;) {
    for (int i = 0; i < 3; i++) {
      if (digitalRead(buttonPins[i]) == LOW) {
        beepTone(delaysArr[i]);
      }
    }
    vTaskDelay(pdMS_TO_TICKS(10));
  }
}

void setup() {
  xTaskCreate(TaskBuzzer, "Buzzer", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Buzzer

Program Arduino

System Design with Sensors II – FreeRTOS

Seven Segment

```
#define LATCH_PIN 4
#define CLK_PIN   7
#define DATA_PIN  8

void SendDataToSegment(byte Segment_no, byte hexValue);

const byte DIGIT_MAP[] = {
  0xC0, // 0
  0xF9, // 1
  0xA4, // 2
  0xB0, // 3
  0x99, // 4
  0x92, // 5
  0x82, // 6
  0xF8, // 7
  0x80, // 8
  0x90  // 9
};

const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};

int counter = 0;

void SendDataToSegment(byte Segment_no, byte hexValue) {
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, hexValue);
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, Segment_no);
  digitalWrite(LATCH_PIN, HIGH);
}
```

```
void setup() {
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLK_PIN, OUTPUT);
  pinMode(DATA_PIN, OUTPUT);
}

void loop() {
  unsigned long startTime = millis();
  while (millis() - startTime < 1000) {
    displayNumber(counter);
  }
  counter = (counter + 1) % 10000;
}

void displayNumber(int num) {
  int digits[4];

  digits[0] = num / 1000;         // Binler basamağı
  digits[1] = (num / 100) % 10;   // Yüzler basamağı
  digits[2] = (num / 10) % 10;    // Onlar basamağı
  digits[3] = num % 10;           // Birler basamağı

  for (int i = 0; i < 4; i++) {
    SendDataToSegment(SEGMENT_SELECT[i], DIGIT_MAP[digits[i]]);
    delay(2);
  }
}
```

System Design with Sensors II – FreeRTOS

Seven Segment

```
#include <Arduino_FreeRTOS.h>

#define LATCH_PIN 4
#define CLK_PIN   7
#define DATA_PIN  8

const byte DIGIT_MAP[] = {
  0xC0, 0xF9, 0xA4, 0xB0,
  0x99, 0x92, 0x82, 0xF8,
  0x80, 0x90
};

const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};

volatile int counter = 0;

void SendDataToSegment(byte seg, byte val) {
  digitalWrite(LATCH_PIN, LOW);
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, val);
  shiftOut(DATA_PIN, CLK_PIN, MSBFIRST, seg);
  digitalWrite(LATCH_PIN, HIGH);
}

void displayNumber(int num) {
  int d[4];
  d[0] = num / 1000;
  d[1] = (num / 100) % 10;
  d[2] = (num / 10)  % 10;
  d[3] = num % 10;
  for (int i = 0; i < 4; i++) {
    SendDataToSegment(SEGMENT_SELECT[i], DIGIT_MAP[d[i]]);
    vTaskDelay(2 / portTICK_PERIOD_MS);
  }
}

void TaskDisplay(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    displayNumber(counter);
  }
}

void TaskCounter(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    counter = (counter + 1) % 10000;
  }
}

void setup() {
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLK_PIN, OUTPUT);
  pinMode(DATA_PIN, OUTPUT);
  xTaskCreate(TaskDisplay, "Disp", 128, NULL, 1, NULL);
  xTaskCreate(TaskCounter, "Count", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Seven Segment

Program Arduino

System Design with Sensors II – FreeRTOS

Potentiometer

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    int potValue = analogRead(A0);
    Serial.println(potValue);
    delay(500);
}
```

```
#include <Arduino_FreeRTOS.h>

void TaskReadPot(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    int potValue = analogRead(A0);
    Serial.println(potValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(TaskReadPot, "ReadPot", 128, NULL, 1, NULL);
}

void loop() {
}
```

System Design with Sensors II – FreeRTOS

Potentiometer

```
#include <Arduino_FreeRTOS.h>

void TaskReadPot(void *pvParameters) {
  (void) pvParameters;
  for (;;) {
    int potValue = analogRead(A0);
    Serial.println(potValue);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(9600);
  xTaskCreate(TaskReadPot, "ReadPot", 128, NULL, 1,
NULL);
}

void loop() {
}
```

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

MPU6050 IMU

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

Adafruit_MPU6050 mpu;

void setup() {
    Serial.begin(9600);
    while (!Serial);  // Seri port açılmasını bekle (opsiyonel)

    Wire.begin();  // Arduino Mega için SDA=20, SCL=21
    if (!mpu.begin()) {
        Serial.println("MPU6050 bağlanamadı!");
        while (1);
    }

    Serial.println("MPU6050 Başlatıldı!");
}
```

```
void loop() {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    // İvmeölçer ile açı hesaplama (derece cinsinden)
    float accelAngleX = atan2(a.acceleration.y, a.acceleration.z) * 180 / PI;
    float accelAngleY = atan2(a.acceleration.x, a.acceleration.z) * 180 / PI;

    // Jiroskop verilerini dereceye çevirme
    float gyroX = g.gyro.x * 180 / PI;
    float gyroY = g.gyro.y * 180 / PI;
    float gyroZ = g.gyro.z * 180 / PI;

    // Yaw, Pitch, Roll hesaplama (Temel Filtre)
    static float yaw = 0, pitch = 0, roll = 0;
    float dt = 0.01; // 10ms döngü süresi

    roll = 0.96 * (roll + gyroX * dt) + 0.04 * accelAngleX;
    pitch = 0.96 * (pitch + gyroY * dt) + 0.04 * accelAngleY;
    yaw += gyroZ * dt;

    // Seri porttan gönder
    Serial.print("Yaw: ");
    Serial.print(yaw);
    Serial.print(" | Pitch: ");
    Serial.print(pitch);
    Serial.print(" | Roll: ");
    Serial.println(roll);

    delay(10);
}
```

System Design with Sensors II – FreeRTOS

MPU6050 IMU

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>

Adafruit_MPU6050 mpu;

void setup() {
    Serial.begin(9600);
    while (!Serial);  // Seri port açılmasını bekle (opsiyonel)

    Wire.begin();  // Arduino Mega için SDA=20, SCL=21
    if (!mpu.begin()) {
        Serial.println("MPU6050 bağlanamadı!");
        while (1);
    }

    Serial.println("MPU6050 Başlatıldı!");
}
```

```
void loop() {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    // İvmeölçer ile açı hesaplama (derece cinsinden)
    float accelAngleX = atan2(a.acceleration.y, a.acceleration.z) * 180 / PI;
    float accelAngleY = atan2(a.acceleration.x, a.acceleration.z) * 180 / PI;

    // Jiroskop verilerini dereceye çevirme
    float gyroX = g.gyro.x * 180 / PI;
    float gyroY = g.gyro.y * 180 / PI;
    float gyroZ = g.gyro.z * 180 / PI;

    // Yaw, Pitch, Roll hesaplama (Temel Filtre)
    static float yaw = 0, pitch = 0, roll = 0;
    float dt = 0.01; // 10ms döngü süresi

    roll = 0.96 * (roll + gyroX * dt) + 0.04 * accelAngleX;
    pitch = 0.96 * (pitch + gyroY * dt) + 0.04 * accelAngleY;
    yaw += gyroZ * dt;

    // Seri porttan gönder
    Serial.print("Yaw: ");
    Serial.print(yaw);
    Serial.print(" | Pitch: ");
    Serial.print(pitch);
    Serial.print(" | Roll: ");
    Serial.println(roll);

    delay(10);
}
```

System Design with Sensors II – FreeRTOS

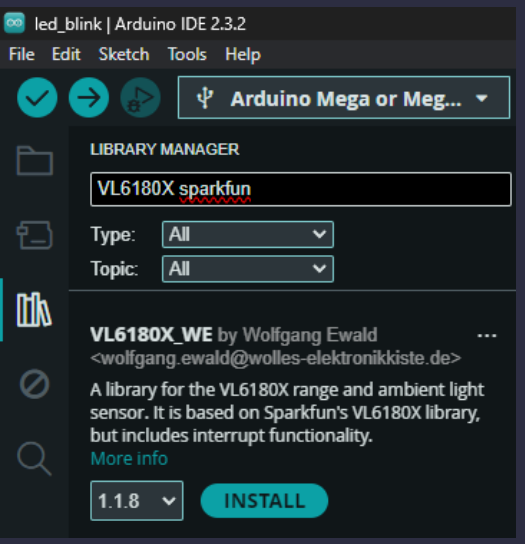
MPU6050 IMU

Program Arduino

Open Putty

System Design with Sensors II – FreeRTOS

TOF050C Laser Distance Measurement

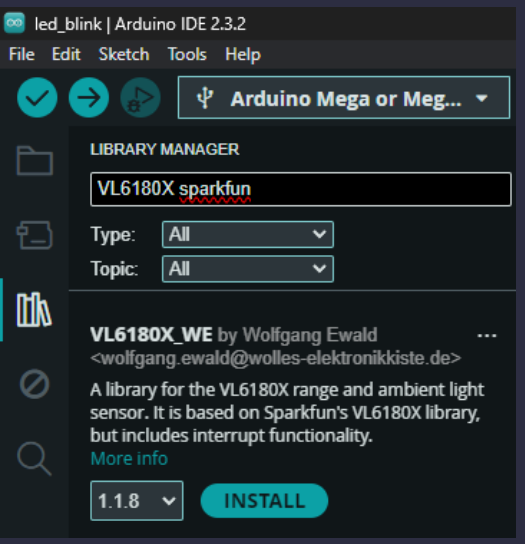



```
#include <Wire.h>
#include <VL6180X_WE.h>
#define VL6180X_ADDRESS 0x29

VL6180xIdentification identification;
VL6180x sensor(VL6180X_ADDRESS);

void setup() {

  Serial.begin(9600); //Start Serial at 9600bps
  Wire.begin(); //Start I2C library

  sensor.getIdentification(&identification);
  printIdentification(&identification);

    if(sensor.VL6180xInit() != 0){
    Serial.println("FAILED TO INITALIZE");
  };

  sensor.VL6180xDefaultSettings();
  delay(100); // delay 0.1 s


}
```

```
void loop() {

  Serial.print("Distance measured (mm) = ");
  Serial.println( sensor.getDistance() );

  delay(500);
};

void printIdentification(struct VL6180xIdentification *temp){
  Serial.print("Model ID = ");
  Serial.println(temp->idModel);

  Serial.print("Model Rev = ");
  Serial.print(temp->idModelRevMajor);
  Serial.print(".");
  Serial.println(temp->idModelRevMinor);

  Serial.print("Module Rev = ");
  Serial.print(temp->idModuleRevMajor);
  Serial.print(".");
  Serial.println(temp->idModuleRevMinor);

  Serial.print("Manufacture Date = ");
  Serial.print((temp->idDate >> 3) & 0x001F);
  Serial.print("/");
  Serial.print((temp->idDate >> 8) & 0x000F);
  Serial.print("/1");
  Serial.print((temp->idDate >> 12) & 0x000F);
 }
```

System Design with Sensors II – FreeRTOS

TOF050C Laser Distance Measurement




```
#include <Wire.h>
#include <VL6180X_WE.h>
#define VL6180X_ADDRESS 0x29

VL6180xIdentification identification;
VL6180x sensor(VL6180X_ADDRESS);

void setup() {

  Serial.begin(9600); //Start Serial at 9600bps
  Wire.begin(); //Start I2C library

  sensor.getIdentification(&identification);
  printIdentification(&identification);

    if(sensor.VL6180xInit() != 0){
    Serial.println("FAILED TO INITALIZE");
  };

  sensor.VL6180xDefaultSettings();
  delay(100); // delay 0.1 s


}
```

```
void loop() {

  Serial.print("Distance measured (mm) = ");
  Serial.println( sensor.getDistance() );

  delay(500);
};

void printIdentification(struct VL6180xIdentification *temp){
  Serial.print("Model ID = ");
  Serial.println(temp->idModel);

  Serial.print("Model Rev = ");
  Serial.print(temp->idModelRevMajor);
  Serial.print(".");
  Serial.println(temp->idModelRevMinor);

  Serial.print("Module Rev = ");
  Serial.print(temp->idModuleRevMajor);
  Serial.print(".");
  Serial.println(temp->idModuleRevMinor);

  Serial.print("Manufacture Date = ");
  Serial.print((temp->idDate >> 3) & 0x001F);
  Serial.print("/");
  Serial.print((temp->idDate >> 8) & 0x000F);
  Serial.print("/1");
  Serial.print((temp->idDate >> 12) & 0x000F);
 }
```

System Design with Sensors II – FreeRTOS

TOF050C Laser Distance Measurement

Program Arduino

Open Putty