MICROCONTROLADORES

Unidade II - Introdução a ESP8266/ESP32
Aula 2

Prof. Ewerton Salvador

Instalação do ESP SDK

- Em geral, os seguintes passos são necessários para preparação dos ambientes de programação da Espressif:
  - Instalação dos pré-requisitos para a SDK, como interpretador Python, Flex, Cmake, etc.;
  - Instalação da toolchain para microcontroladores ESP
    - *Toolchain* é um conjunto de ferramentas criado para possibilitar a execução de tarefas de desenvolvimento e implantação de software
    - Esse passo é automático na instalação da ESP-IDF SDK
  - Instalação do SDK propriamente dito (script de configuração/compilação/flashing, bibliotecas, etc.)
  - Configuração de variáveis de ambiente, como PATH, IDF_PATH, etc.

Fluxo de desenvolvimento ESP SDK

- O fluxo de trabalho típico para o desenvolvimento de programas com os SDKs da ESP pode ser resumido da seguinte forma:
  1. Desenvolvimento
  2. Configuração do projeto
  3. *Building*
  4. *Flashing*
  5. *Monitoring*

Configuração do projeto

- ESP-IDF usa a biblioteca **kconfiglib** do Python para proporcionar um mecanismo de configuração do projeto. Exemplos de opções de configuração:
  - A forma como o projeto é gravado na ESP (*flashing*)
  - Configurações do FreeRTOS (bootloader, partições, etc.)
  - Opções do compilador, etc.

- Configuração é salva em um arquivo chamado sdkconfig
  - Sistema de build constrói sdkconfig.h a partir do arquivo sdkconfig, fazendo com que as opções sejam acessíveis no programa

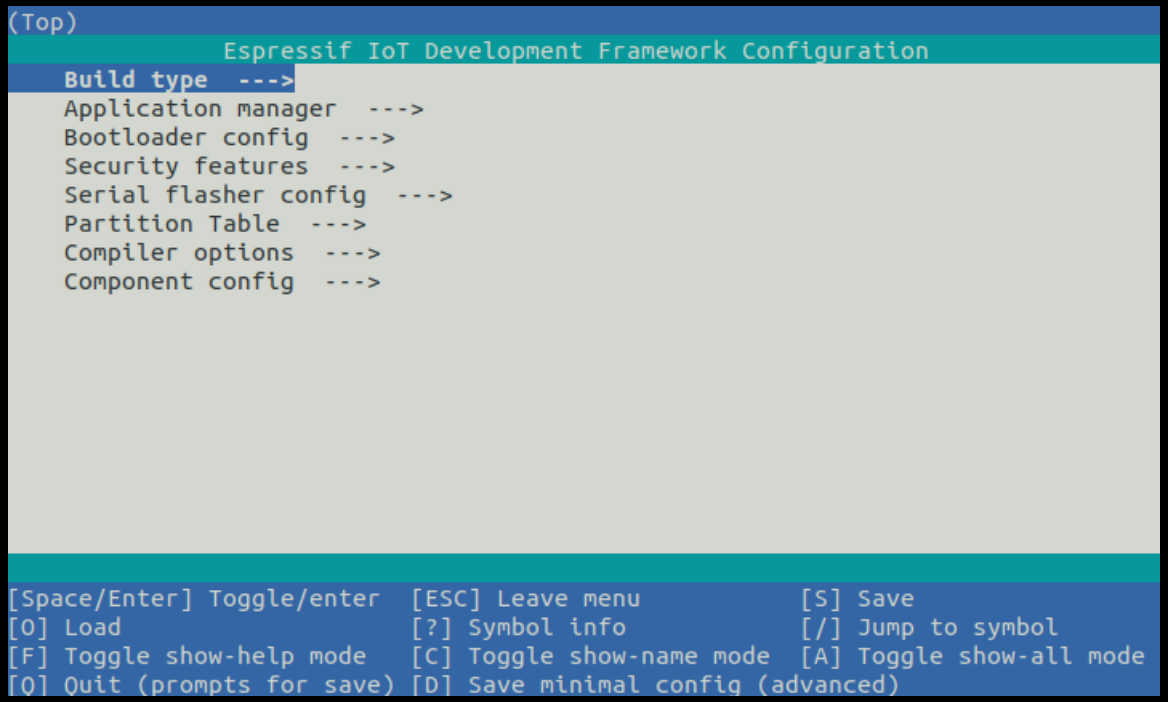
Configuração do projeto

- Trecho de arquivo sdkconfig

```
#
# Automatically generated file. DO NOT EDIT.
# Espressif IoT Development Framework (ESP-IDF) Project Configuration
#
CONFIG_SOC_BROWNOUT_RESET_SUPPORTED="Not determined"
CONFIG_SOC_TWAI_BRP_DIV_SUPPORTED="Not determined"
CONFIG_SOC_DPORT_WORKAROUND="Not determined"
CONFIG_SOC_CAPS_ECO_VER_MAX=3
CONFIG_SOC_ADC_SUPPORTED=y
CONFIG_SOC_DAC_SUPPORTED=y
CONFIG_SOC_MCPWM_SUPPORTED=y
CONFIG_SOC_SDMMC_HOST_SUPPORTED=y
CONFIG_SOC_BT_SUPPORTED=y
CONFIG_SOC_CLASSIC_BT_SUPPORTED=y
CONFIG_SOC_PCNT_SUPPORTED=y
CONFIG_SOC_WIFI_SUPPORTED=y
CONFIG_SOC_SDIO_SLAVE_SUPPORTED=y
CONFIG_SOC_TWAI_SUPPORTED=y
CONFIG_SOC_EMAC_SUPPORTED=y
CONFIG_SOC_ULP_SUPPORTED=y
```

Configuração do projeto

- ESP-IDF também fornece a opção de utilização de um menu baseado em terminal para configuração do projeto
  - ESP8266: make menuconfig
  - ESP32: idf.py menuconfig



Building

- A construção do programa envolve a compilação de uma série de componentes. Exemplos de componentes:
  - Bibliotecas básicas da ESP (libc, ROM bindings, etc.)
  - Drivers Wi-Fi
  - Pilha TCP/IP
  - S.O. FreeRTOS
  - Webserver, etc.
- Tipicamente um projeto constrói duas aplicações:
  - Project App
  - Bootloader App

Building

- Duas variáveis de ambiente precisam estar configuradas para que a construção funcione:
  - IDF_PATH, informando o caminho do framework
  - PATH, que deverá incluir os diretórios da toolchain do MCU

- Como fazer o build do projeto:
  - ESP8266: make build
  - ESP32: idf.py build

Flashing

- Processo de gravação das aplicações na ESP
- Realizado através dos comandos abaixo:
  - ESP8266: make flash
  - ESP32: idf.py –p *[PORTA]* flash
- Se necessário, os comandos acima também compilam as aplicações
- Atenção: no caso da ESP32, ao ser ligada/resetada, o pino GPIO0 precisa ser mantido no nível lógico baixo para que possa entrar no modo de *flashing*
  - Um botão FLASH ou BOOT disponível em algumas placas coloca o pino GPIO0 no nível lógico baixo

Flashing

- Tamanho a ser utilizado na ESP pode ser conhecido através da opção size:
  - ESP8266: make size
  - ESP32: idf.py size

```
Total sizes:
 DRAM .data size:    1348 bytes
 DRAM .bss  size:    6472 bytes
Used static DRAM:    7820 bytes (  90484 available, 8.0% used)
Used static IRAM:   22591 bytes (  26561 available, 46.0% used)
      Flash code:  117882 bytes
    Flash rodata:   29060 bytes
Total image size:~ 170881 bytes (.bin may be padded larger)
```

Monitoring

- O SDK provê uma ferramenta para monitorar o funcionamento da ESP
  - Essencialmente é um terminal serial

- Ferramenta é acionada através dos comandos abaixo:
  - ESP8266: make monitor
  - ESP32: idf.py –p *[PORTA]* monitor

O exemplo “hello_world”

- Vamos experimentar o fluxo de desenvolvimento ESP com o código exemplo “hello_world”, disponibilizado no próprio SDK

- Copie o exemplo para o seu diretório “home” ou algum outro de sua preferência:
  - cp –R $IDF_PATH/examples/get-started/hello_world .

- Utilizando uma IDE (como VSCode), abra o diretório hello_world e em seguida abra o arquivo ./main/hello_world_main.c

O exemplo “hello_world”

- Teste os processos de configuração do projeto (não precisa alterar o valor de nenhuma opção), building, flashing e monitoring

- Em seguida, verifique que o código “mínimo” abaixo também funciona

```
1   #include <stdio.h>
2
3
4   void app_main(void)
5   {
6       printf("Hello world!\n");
7   }
```

O exemplo “hello_world”

- Alguns pontos de aprendizado do exemplo “hello_world”:
  - A função de entrada para a aplicação a ser executada na ESP é “void app_main(void)”;
  - A função printf pode ser utilizada para transmitir strings da ESP para o PC através da conexão serial da USB;
  - A função típica para provocar um delay é a vTaskDelay (do FreeRTOS);
  - A função vTaskDelay recebe como parâmetro a duração do delay **em número de ciclos**
  - Para transformar uma determinada quantidade de milissegundos em número de ciclos, tipicamente utiliza-se a divisão do número de milissegundos pela constante **portTICK_PERIOD_MS**, definida em freeRTOS.h

Introdução a programação super loop

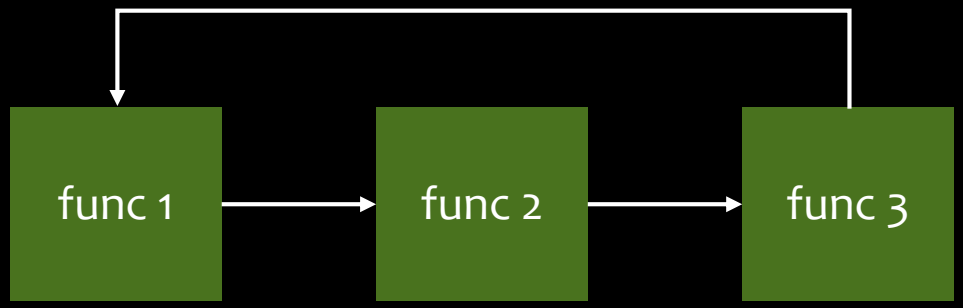
- Sistemas embarcados, em praticamente todos os casos, não possuem um ponto de parada do programa

- Por natureza, é esperado que um programa esteja sempre funcionando, como se em segundo plano, realizando tarefa administrativas e estando pronto para processar entradas

- Uma forma de se implementar esse comportamento é utilizando um **super loop**

Introdução a programação super loop

```
void main (void) {
    while(1){
        func1();
        func2();
        func3();
        //...
    }
}
```

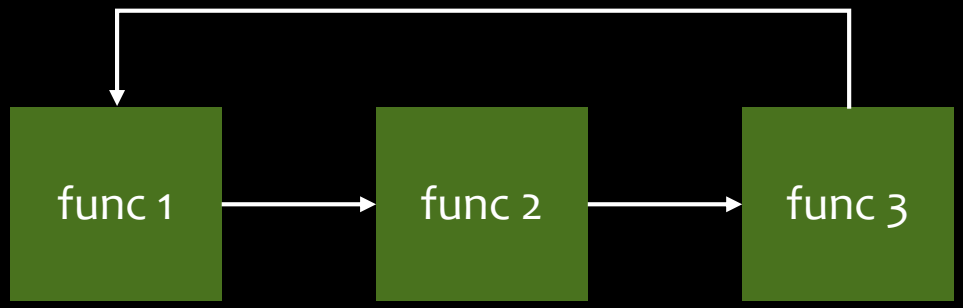


Super Loop

Introdução a programação super loop

- Funções do programa são realizadas de forma estritamente sequencial

- Qualquer atraso introduzido em uma função termina sendo propagado para as funções seguintes



Super Loop

ESP-IDF RTOS: Tasks

- RTOS introduz o conceito de tarefa (*task*)
- Tarefas podem ser encaradas como super loops adicionais
- Cada tarefa possui a sua própria pilha de execução
  - Importante para tornar as chamadas de funções independentes em cada tarefa
- Tarefas possuem níveis de prioridades associadas a elas
- O escalonador de tarefas sempre priorizará a execução da(s) task(s) com maior nível de prioridade em um dado momento
  - Escalonamento preemptivo

ESP-IDF RTOS: Tasks

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"


//Tasks handles
static TaskHandle_t task1_handle = NULL;
static TaskHandle_t task2_handle = NULL;

//Tasks
void task1(void *parameter){

  while(1){
    printf("Task 1\n");
    vTaskDelay(500/portTICK_PERIOD_MS);
  }

}

void task2(void *parameter){

 while(1){
    printf("Task 2\n");
    vTaskDelay(900/portTICK_PERIOD_MS);
  }

}


void app_main(void){

    xTaskCreate(task1,"Task 1",1024,NULL,1,&task1_handle);
    xTaskCreate(task2,"Task 2",1024,NULL,1,&task2_handle);

}
```

Exemplo de criação de 2 tasks no FreeRTOS

ESP-IDF RTOS: Tasks

- *Tasks Handles* podem ser associadas a cada task, de modo a permitir que essas tasks sejam controladas em outras tasks
  - Exemplos: inicar, interromper, deletar, etc.

- Declaração como variável global:
  - static TaskHandle_t nome_var = NULL;

ESP-IDF RTOS: Tasks

- Função: xTaskCreate
- Parâmetros:
  1. Nome da função que implementa a task. A função não deve retornar nunca, ou deve ser encerrada com a função vTaskDelete
  2. Um nome descritivo da task (para debugging)
  3. Tamanho da pilha, especificada em número de **bytes**
  4. Um ponteiro para um parâmetro que se queira passar para a task
  5. A prioridade na qual a task deverá rodar. No ESP, por padrão, as prioridades vão de 0 a 24
  6. Endereço para gravação do handle da task
- Retorno:
  - A constante pdPASS se a task for criada com sucesso. Do contrário, retorna erro definido em projdefs.h

ESP-IDF RTOS: Tasks

- Função: xTaskCreatePinnedToCore
- Quase o mesmo que xTaskCreate, porém com a adição de um sétimo parâmetro para definir afinidade por núcleo
- Parâmetro adicional:
  7. 0 ou 1 para indicar em qual dos dois núcleos a task deve rodar, ou a constante tskNO_AFFINITY para não atrelar a task a um núcleo específico

ESP-IDF RTOS: Tasks

- Função: vTaskDelete
- Finalidade: remover uma task
- Parâmetros:
  1. Handle da task a ser deletada

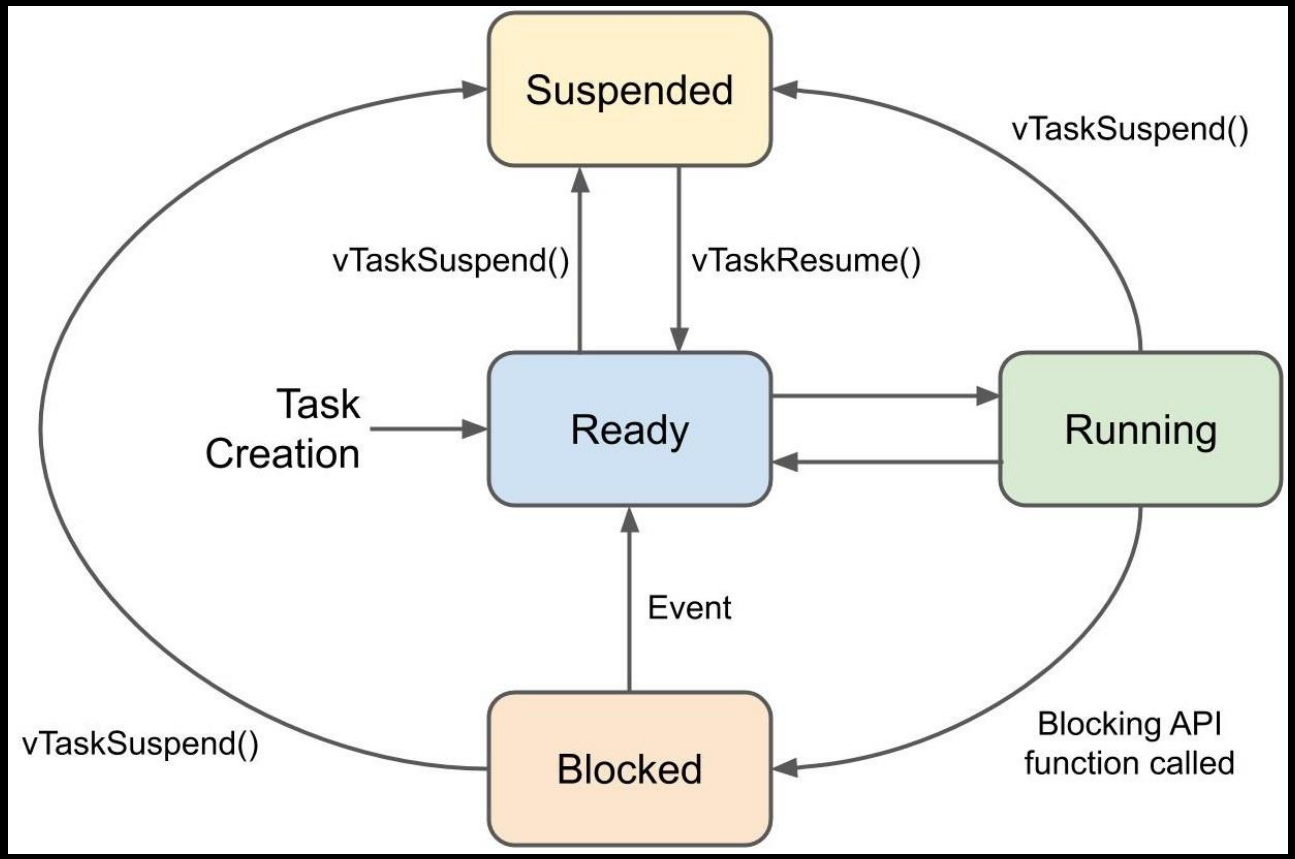
- Função: vTaskSuspend
- Finalidade: suspender uma task
- Parâmetros:
  1. Handle da task a ser suspensa

ESP-IDF RTOS: Tasks

- Função: vTaskResume
- Finalidade: retirar uma task do estado de suspensão
- Parâmetros:
  1. Handle da task a ser retomada

ESP-IDF RTOS: Tasks

- Diagrama de estados de uma task



Fonte: https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-3-task-scheduling/8fbb9e0b0eed4279a2dd698f02ce125f