Hệ nhúng
(Embedded Systems)

IT4210

Đỗ Công Thuần

Khoa Kỹ thuật máy tính, Trường CNTT&TT

Đại học Bách khoa Hà Nội

Email: thuandc@soict.hust.edu.vn

ONE LOVE. ONE FUTURE.

Giới thiệu môn học

- Tên học phần: **Hệ nhúng**
- Mã học phần: **IT4210 (3-0-1-6)**
- Thời lượng:
  - 16.5 buổi lý thuyết (3 tiết/buổi)
  - 3 buổi thực hành (5 tiết/buổi)
- Yêu cầu kiến thức nền tảng:
  - Kiến trúc máy tính
  - Vi xử lý
  - Lập trình C

Mục tiêu môn học

- Nắm được kiến trúc tổng quan, đặc điểm và hoạt động của một hệ nhúng
- Biết thiết kế hệ nhúng cơ bản (nguyên lý thiết kế mạch, …)
- Nắm được kiến trúc vi điều khiển (Intel, ARM)
- Lập trình vi điều khiển từ cơ bản đến nâng cao với các dòng vi điều khiển phổ biến
- Lập trình với hệ điều hành nhúng

Đánh giá học phần

1. Đánh giá quá trình: **40%**
   - Bài tập về nhà
   - Chuyên cần
   - Các bài thực hành, nhóm **4 SV/nhóm**
2. Đánh giá cuối kỳ: **60%**
   - Làm project cuối kỳ, nhóm **4 SV/nhóm**
   - Yêu cầu sinh viên tự chọn nhóm và đăng kí đề tài.
     Chú ý: *danh sách đề tài sẽ được cập nhật sau!*

Tài liệu tham khảo

- Textbook/Lecture notes:
  - Peter Marwedel, ***Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things***, Spinger, 4th edition, 2021.
  - Edward A. Lee and Sanjit A. Seshia, ***Introduction to Embedded Systems: A Cyber-Physical Systems Approach***, MIT Press, 2nd edition, 2017.
  - Tammy Noergaard, ***Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers***, Elsevier, 2nd edition, 2013.
  - Han-Way Huang, Leo Chartrand, ***Microcontroller: An Introduction to Software & Hardware Interfacing***, Cengage Learning, 2004.
  - Lectures in Embedded Systems from *Univ. of Cincinnati (EECE 6017C)*, *Univ. of California, Berkeley (EECS 149)*, *Univ. of Pennsylvania (ESE 350)*, *Univ. of Kansas (EECS388).*
  - *…*
- Manuals/Handbooks/Internet
  - Atmel, Microchip, Texas Instruments, Keil…
  - Keil ASM51
  - Arduino IDE
  - …

Nội dung học phần

- Chương 1: Giới thiệu về Hệ nhúng
- Chương 2: Thiết kế phần cứng Hệ nhúng
- Chương 3: Lập trình với 8051
- Chương 4: Ghép nối ngoại vi với 8051
- Chương 5: Arduino
- Chương 6: Ghép nối nối tiếp
- Chương 7: Ghép nối với thế giới thực
- Chương 8: Kiến trúc ARM
- Chương 9: RTOS và FreeRTOS

Nội dung học phần

- Chương 1: Giới thiệu về Hệ nhúng
- Chương 2: Thiết kế phần cứng Hệ nhúng
- Chương 3: Lập trình với 8051
- Chương 4: Ghép nối ngoại vi với 8051
- Chương 5: Arduino
- Chương 6: Ghép nối nối tiếp
- Chương 7: Ghép nối với thế giới thực
- Chương 8: Kiến trúc ARM
- Chương 9: RTOS và FreeRTOS

Chương 9

RTOS và FreeRTOS

Tại sao cần RTOS?

- CMSIS và HAL lib:
  - Giao tiếp với phần cứng/ngoại vi được thực hiện trên các API, driver đã chuẩn hóa.
  - Tương thích tốt giữa các dòng chip hoặc các thiết kế mạch khác nhau.
- Vấn đề:
  - Vẫn chỉ xây dựng được các ứng dụng đơn lẻ, giống như trên hệ 8 bit
  - Main loop + interrupt + DMA
  - Khó tận dụng hết tài nguyên tính toán của CPU

→Cần OS để hỗ trợ mô hình thực thi đa luồng

RTOS

- Mỗi RTOS gồm 1 ***real-time OS kernel***.
- Kernel quản lý các tài nguyên trong bất kỳ hệ thống thời gian thực nào, ví dụ: *processor*, *memory*, *system timer*.
- **Chức năng chính:**
  - *Thread management*
  - *Interprocess synchronization & communication*
  - *Time management*
  - *Memory management*

Yêu cầu cơ bản của RTOS

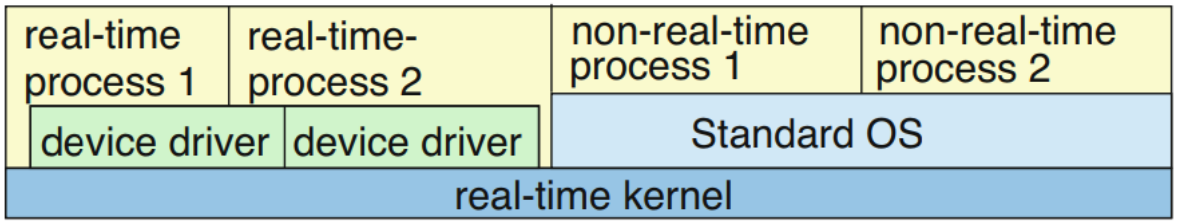
1. Predictable OS timing behavior
   - Đảm bảo *upper bound* về thời gian thực thi các *OS services*.
   - Thời gian vô hiệu hóa các ngắt (nếu có) phải ngắn để tránh *interference* giữa các bộ phận của *OS*.
2. OS must manage the timing and scheduling
   - OS cần phải "*aware of task deadlines*" để có thể triển khai các phương pháp lập lịch.
3. OS must be fast
   - Tất cả các yêu cầu được đề cập đều vô nghĩa nếu OS rất chậm!

Phân loại RTOS

- Small, fast, proprietary kernels
  - Để giảm *run-time overheads*:
    - Fast context switch; small size; responds to external interrupts quickly; provides fixed or variable sized partitions for memory management, …
  - Để đối phó với *timing-constraints*:
    - Provide bounded execution time for most primitives; maintain a real-time clock; provides primitives to delay processing by a fixed amount of time and to suspend/resume execution, …
  - Ví dụ: QNX, PDOS, VCOS, VxWORKS

Phân loại RTOS

- Standard OS with real-time extensions
  - *Hybrid OS* được phát triển bằng cách tận dụng lợi thế của các *mainstream OS*.
  - Một *RT-kernel* sẽ thực thi tất cả các *RT-process*.
  - *Standard OS* sẽ được thực thi như 1 trong những *RT-process*.
  - Ví dụ: RT-Linux



FreeRTOS

- Real-time OS (*real-time kernel*) cho hệ nhúng kích thước nhỏ, vừa
- Mã nguồn mở, phát triển bởi Richard Barry (2003)
- Maintain bởi Real Time Engineers Ltd.
- Mua lại bởi Amazon (2017)
- Hỗ trợ nhiều dòng chip:
  - ARM (ARM7, ARM9, Cortex-M3, Cortex-M4, Cortex-A), Atmel AVR, AVR32, HCS12, MicroBlaze, Cortus (APS1, APS3, APS3R, APS5, FPF3, FPS6, FPS8), MSP430, PIC, Renesas H8/S, SuperH, RX, x86, 8052, Coldfire, V850, 78K0R, Fujitsu MB91460 series, Fujitsu MB96340 series, Nios II, Cortex-R4, TMS570, RM4x, Espressif ESP32, RISC-V

Tại sao dùng real-time kernel?

- Đảm bảo mỗi ứng dụng đáp ứng *processing deadlines*
- Loại bỏ thông tin thời gian khỏi hệ thống bằng *time-related APIs*
- Dễ bào trì/mở rộng
- Tính module hóa
- Dễ phát triển, kiểm thử, tái sử dụng code
- Xử lý linh hoạt các ngắt
- …

PreeRTOS Features

- Pre-exemptive or co-operative operation
- Very flexible task priority assignement
- Flexible, fast and light weight task notification mechanism
- Queues
- Binary semaphores
- Counting semaphores
- Mutexes
- Recursive Mutexes
- Software timers
- Event groups
- Tick hook functions
- Idle hook functions
- Stack overflow checking
- Trace recording
- Task run-time statistics gathering
- Optional commercial licensing and support
- Full interrupt nesting model (for some architectures)
- A tick-less capability for extreme low power applications
- Software managed interrupt stack when appropriate (this can help save RAM)

Các API chính của FreeRTOS

- **Tạo task**: phức tạp nhất, bắt gặp đầu tiên

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

  - *pvTaskCode*: task routine
  - *pcName*: task name
  - *usStackDepth*: stack size (in words)
  - *pvParameters*: task parameter
  - *uxPriority*: handle

Ví dụ

```
/* Task to be created. */
void vTaskCode( void * pvParameters )
{
    /* The parameter value is expected to be 1 as 1 is passed in the
    pvParameters value in the call to xTaskCreate() below.
    configASSERT( ( ( uint32_t ) pvParameters ) == 1 );

    for( ;; )
    {
        /* Task code goes here. */
    }
}

/* Function that creates a task. */
void vOtherFunction( void )
{
BaseType_t xReturned;
TaskHandle_t xHandle = NULL;

    /* Create the task, storing the handle. */
    xReturned = xTaskCreate(
                    vTaskCode,       /* Function that implements the task. */
                    "NAME",          /* Text name for the task. */
                    STACK_SIZE,      /* Stack size in words, not bytes. */
                    ( void * ) 1,    /* Parameter passed into the task. */
                    tskIDLE_PRIORITY,/* Priority at which the task is created. */
                    &xHandle );      /* Used to pass out the created task's handle. */

    if( xReturned == pdPASS )
    {
        /* The task was created.  Use the task's handle to delete the task. */
        vTaskDelete( xHandle );
    }
}
```

Các API chính của FreeRTOS

- **Xóa task**

```
void vTaskDelete( TaskHandle_t xTask );
```

  - Xóa task khỏi hệ thống, kể cả khi nó đang chạy
  - Tài nguyên được thu hồi trong *system idle task*

Các API chính của FreeRTOS

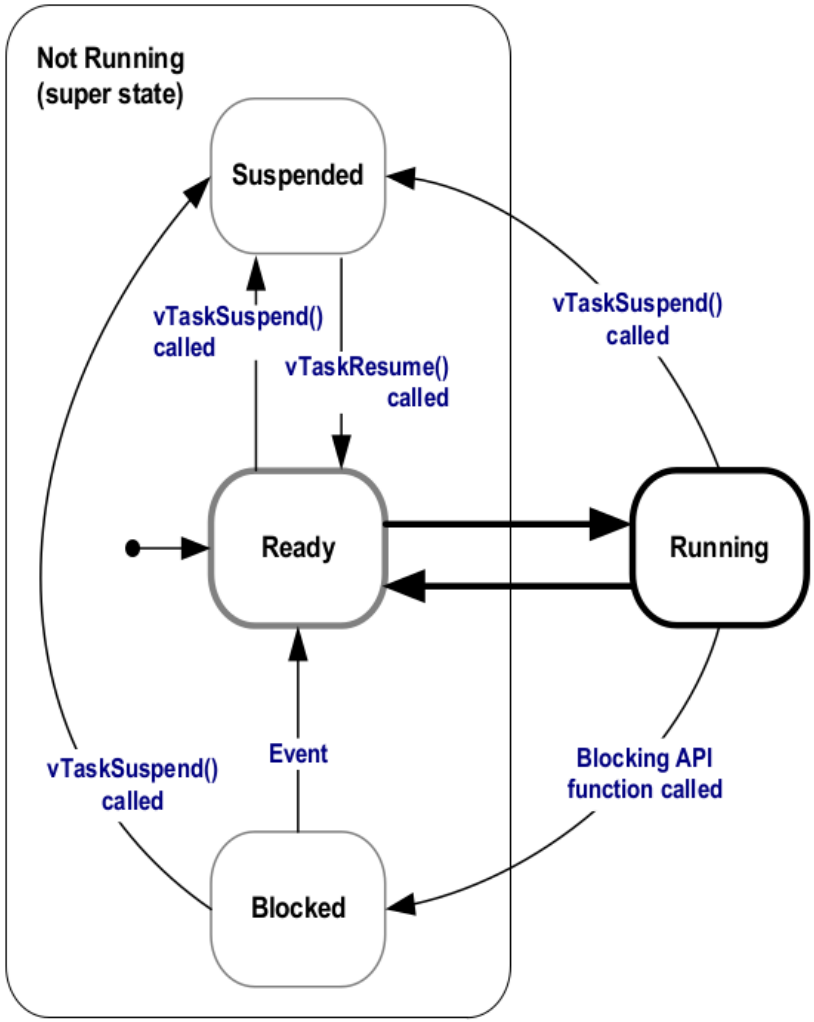
- **Các hàm điều khiển task:**
  - vTaskDelay: đặt *task* đang được gọi trong *blocked state*
  - vTaskDelayUntil: tương tự *vTaskDelay()*, khác biệt là trả về thời gian tuyệt đối
  - uxTaskPriorityGet: truy vấn *prirotiy* của *task*
  - vTaskPrioritySet: thay đổi *prirotiy* của *task*
  - vTaskSuspend: đưa *task* vào *suspended state*, và *task* sẽ không ở trong scheduler's list
  - vTaskResume: đưa task từ *suspended state* trờ lại *ready state*
  - xTaskResumeFromISR: tương tự *vTaskResume()*, khác biệt là có thể được gọi từ một ISR
  - xTaskAbortDelay: đưa một *task* ra khỏi *blocked state*

Các API chính của FreeRTOS

- **Các hàm hệ thống:**
  - taskYIELD
  - taskENTER_CRITICAL
  - taskEXIT_CRITICAL
  - taskENTER_CRITICAL_FROM_ISR
  - taskEXIT_CRITICAL_FROM_ISR
  - taskDISABLE_INTERRUPTS
  - taskENABLE_INTERRUPTS
  - vTaskStartScheduler
  - vTaskEndScheduler
  - vTaskSuspendAll
  - xTaskResumeAll
  - vTaskStepTick

Task State Transitions

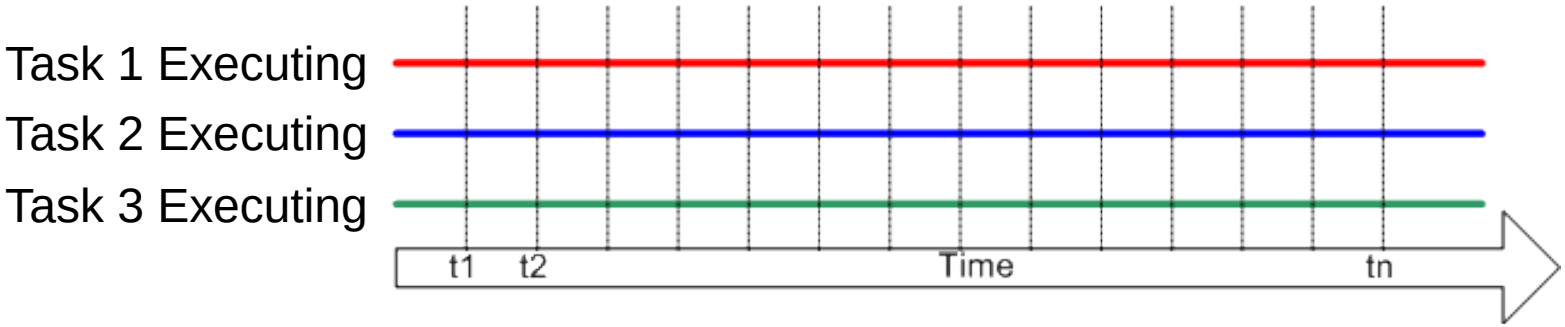
- Tại một thời điểm, mỗi *task* tồn tại ở 1 trong các 4 *states*:
  - ***Running***: đang thực thi
  - ***Ready***: sẵn sáng để thực thi
  - ***Blocked***: chưa sẵn sàng, ví dụ phải chờ data
  - ***Suspended***: bị ngưng



Multitasking vs. Concurrency

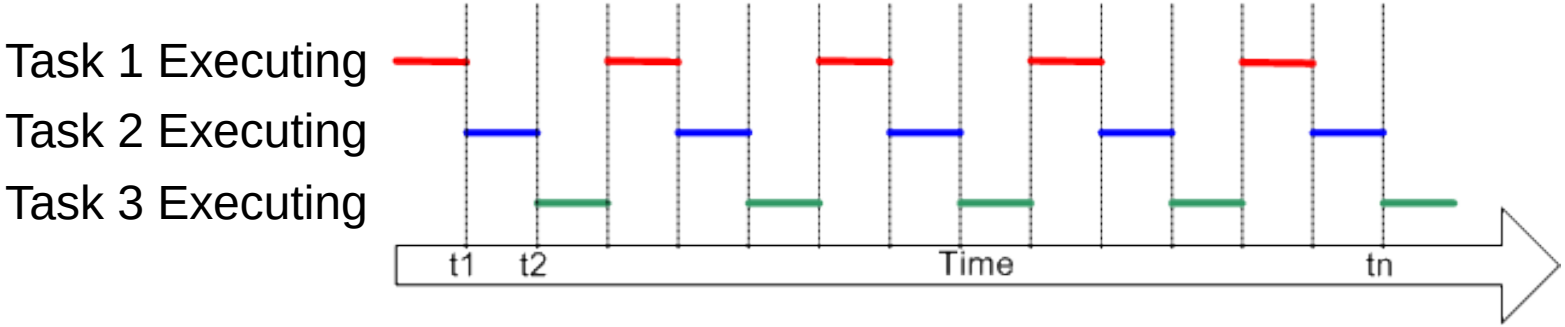
Concurrency

All available task appear to be executing …



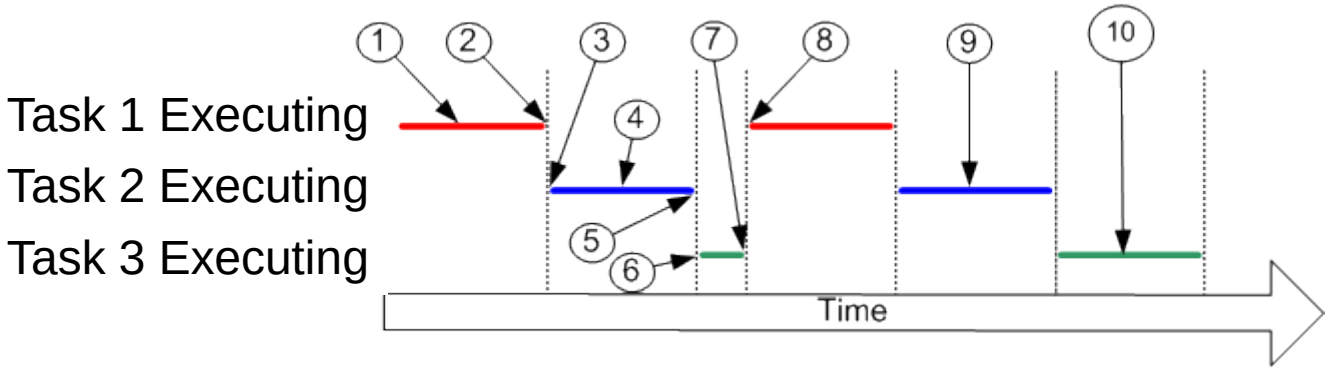
Multitasking

… but only one task is ever executing at any time.



FreeRTOS Scheduling

- Most likely allows each task a "fair" proportion of processor time.



- At **(1)** ***Task 1*** is executing.
- At **(2)** the kernel suspends (swaps out) ***Task 1*** ...
- ... and at **(3)** resumes ***Task 2***.
- While ***Task 2*** is executing **(4)**, it locks a processor peripheral for its own exclusive access.
- At **(5)** the kernel suspends ***Task 2*** ...
- ... and at **(6)** resumes ***task 3***.
- ***Task 3*** tries to access the same processor peripheral, finding it locked ***Task 3*** cannot continue so suspends itself at **(7)**.
- At **(8)** the kernel resumes ***Task 1***.
- Etc.
- The next time ***Task 2*** is executing **(9)** it finishes with the processor peripheral and unlocks it.
- The next time ***Task 3*** is executing **(10)** it finds it can now access the processor peripheral and this time executes until suspended by the kernel.

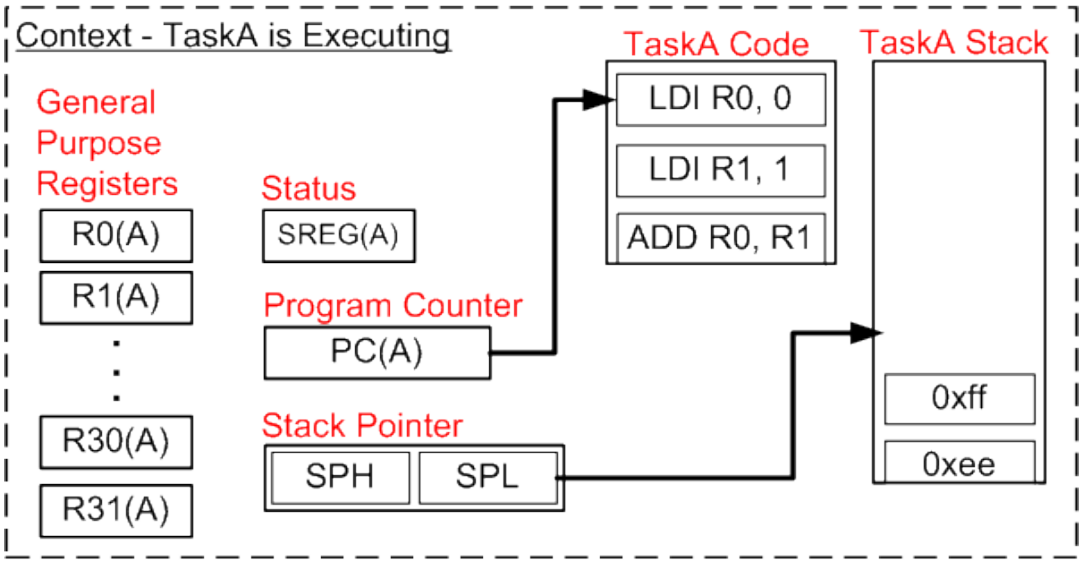
FreeRTOS Context Switching

- Resources (processor registers, stack pointer, etc.) comprise the task execution context.
- A context switch is the process of storing the execution context of a task, so that the task can be restored and resume its execution at a later time.

FreeRTOS Context Switching

- **Example:** context switch from TaskA to TaskB

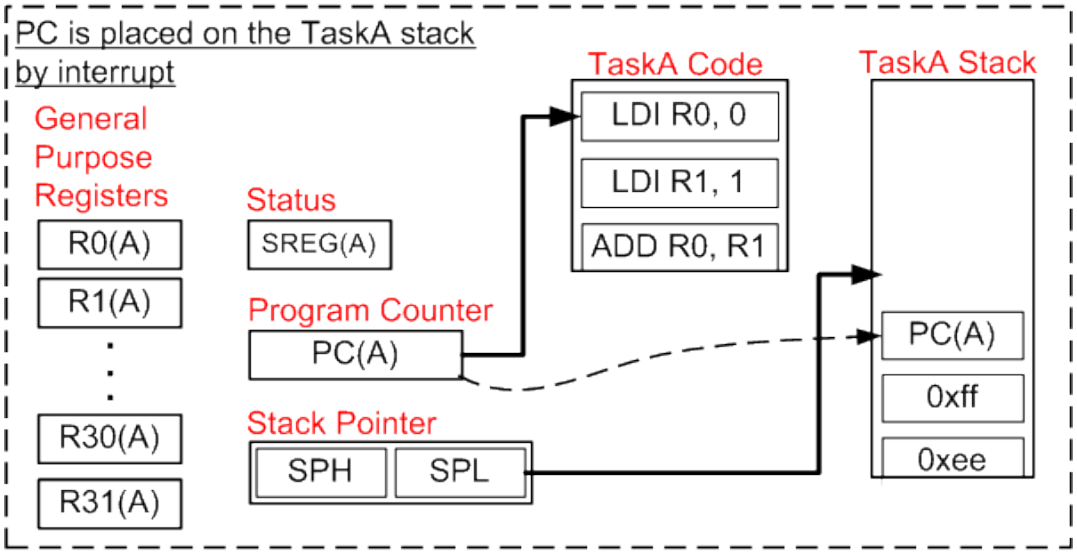
Step 1:



TaskA is executing, TaskB has previously been suspended.

FreeRTOS Context Switching

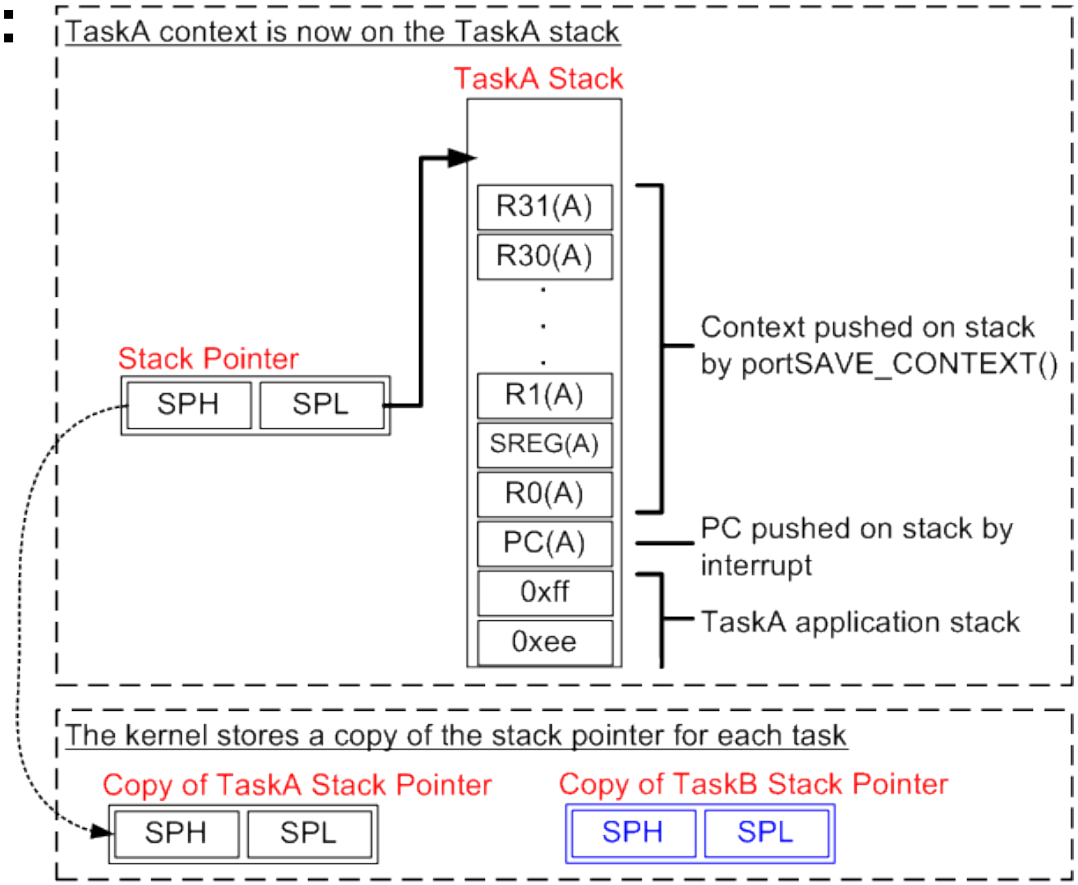
Step 2:



TaskA is about to execute an LDI instruction. When the interrupt occurs Task A context is automatically saved.

FreeRTOS Context Switching

Step 3:



The interrupt is executed.

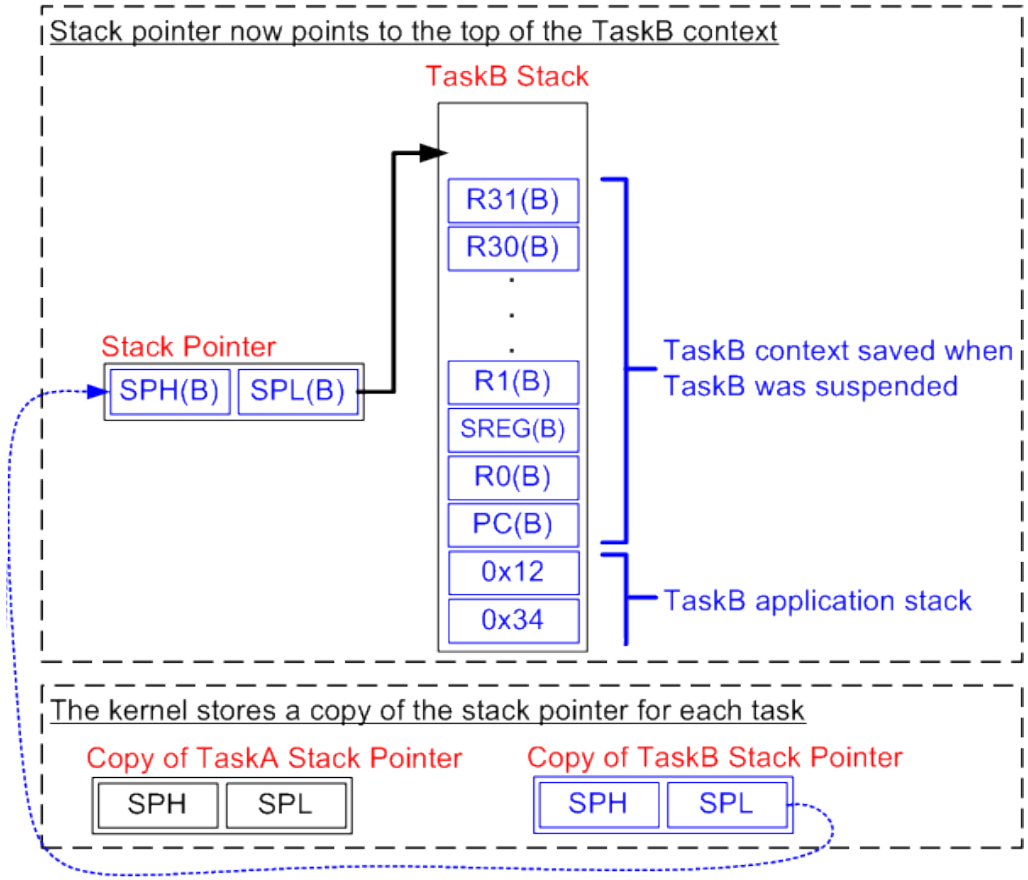
FreeRTOS Context Switching

Step 4:

- The RTOS function *vTaskIncrementTick()* executes after the TaskA context has been saved.
- Assume that incrementing the tick count has caused TaskB to become ready to run.
- TaskB has a higher priority than TaskA so *vTaskSwitchContext()* selects TaskB as the task to be given processing time when the ISR completes.

FreeRTOS Context Switching

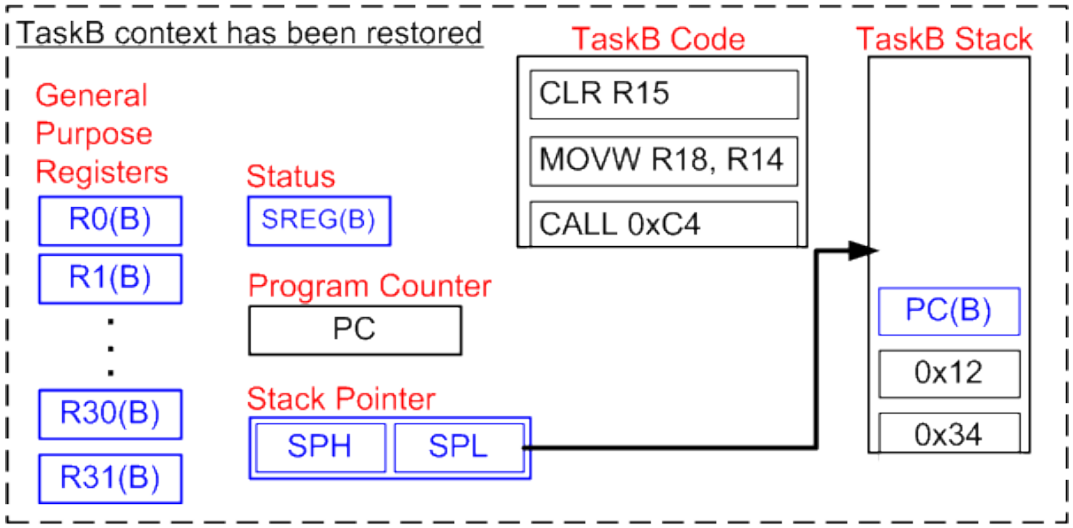
Step 5:



The TaskB context must be restored.

FreeRTOS Context Switching

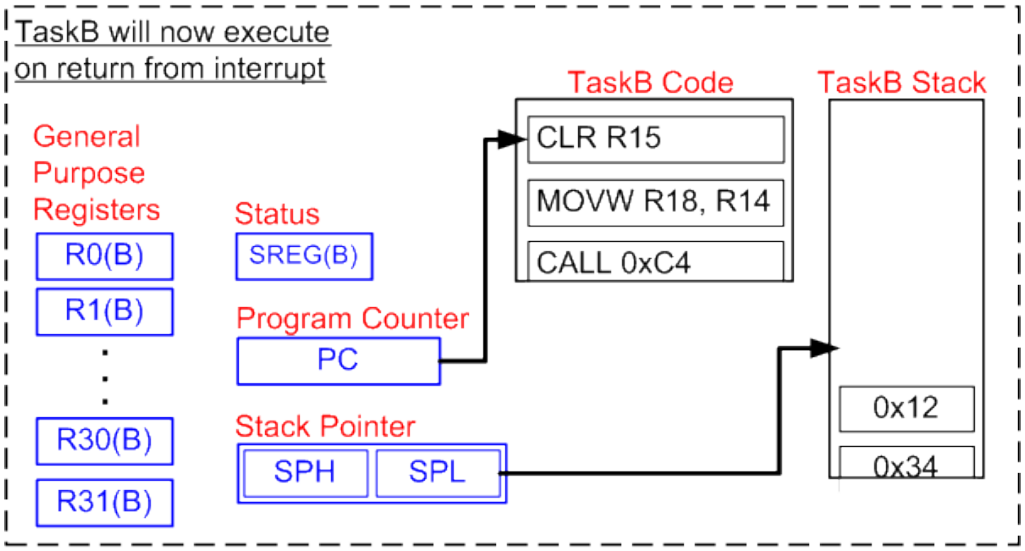
Step 6:



The TaskB context is completely restored.

FreeRTOS Context Switching

Step 7:



The TaskB is now executed.

Tích hợp CMSIS với FreeRTOS

- CMSIS cung cấp *wrapper* cho FreeRTOS
- Truy cập API của FreeRTOS qua CMSIS-RTOS interface.

CMSIS_OS API v1.x	CMSIS_OS API v2.x	FreeRTOS API
osThreadCreate()	osThreadNew()	xTaskCreate()
osThreadGetId()	osThreadGetId()	xTaskGetCurrentTaskHandle()
osThreadTerminate()	osThreadTerminate() osThreadExit()	vTaskDelete()
osThreadYield()	osThreadYield()	taskYIELD()
osThreadSetPriority()	osThreadSetPriority()	vTaskPrioritySet()
osThreadGetPriority()	osThreadGetPriority()	uxTaskPriorityGet() uxTaskPriorityGetFromISR()
osDelay()	osDelay()	vTaskDelay()

Tích hợp CMSIS với FreeRTOS

- Ví dụ:
  - `osThreadId_t Task1Handle;`
  - `osThreadId_t osThreadNew(osThreadFunc_t func, void *argument, const osThreadAttr_t* attr)`
  - `osStatus_t osThreadTerminate(osThreadId_t thread_id)`
  - `osStatus_t osThreadYield(void)`
  - `osStatus_t osThreadResume(osThreadId_t thread_id)`
  - `osThreadState_t osThreadGetState(osThreadId_t thread_id)`
  - `osStatus_t osThreadSuspend(osThreadId_t thread_id)`

Kernel primitives

- Queues
- Queue Sets
- Stream Buffers
- Message Buffers
- Semaphore / Mutexes

FreeRTOS Kernel Services

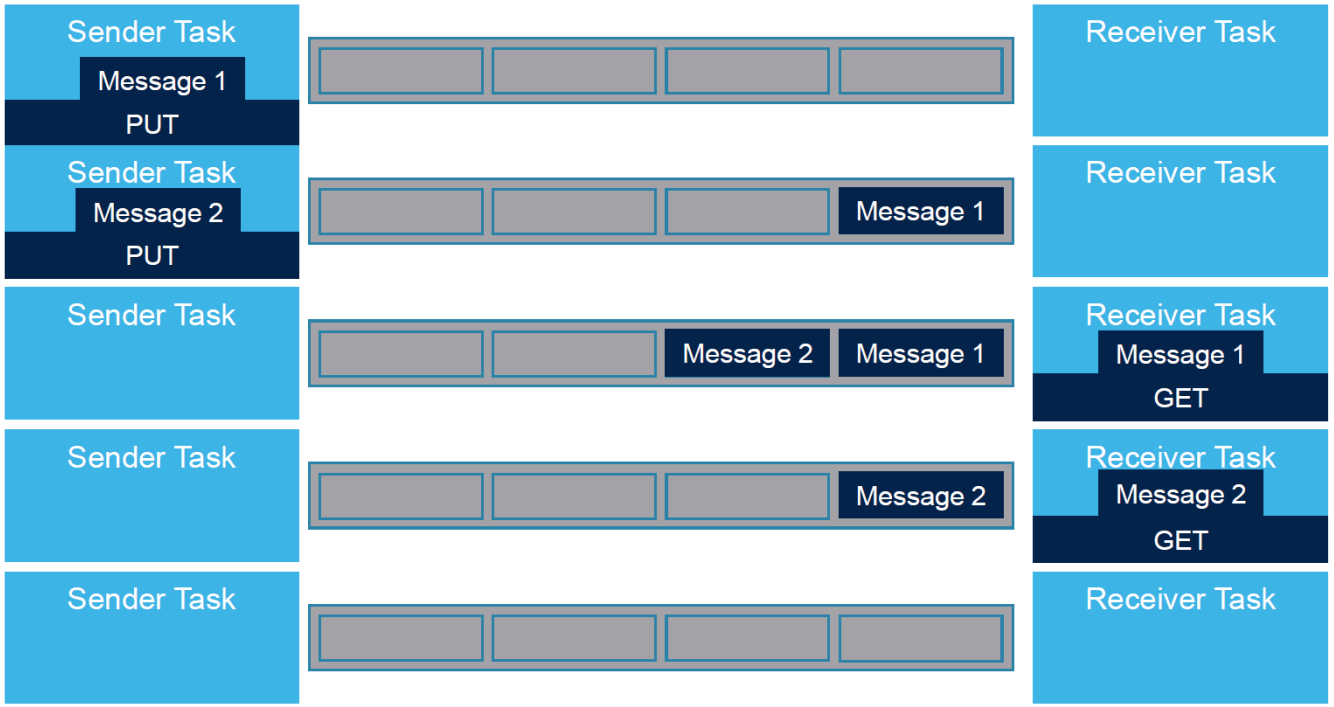
- Heap Memory Management
- Task Management
- Queue Management
- Software Timer Management
- Interrupt Management
- Resource Management
- Event Groups
- Task Notifications

➔ Xem thêm ***https://www.freertos.org***

➔ Xem thêm ***FreeRTOS Kernel Quick Start Guide***

Ví dụ: Thread và IPC

- IPC: *inter-process communication*, cho phép truyền dữ liệu giữa các tiến trình
- Queue: truyền số nguyên/con trỏ giữa các task



Các API liên quan

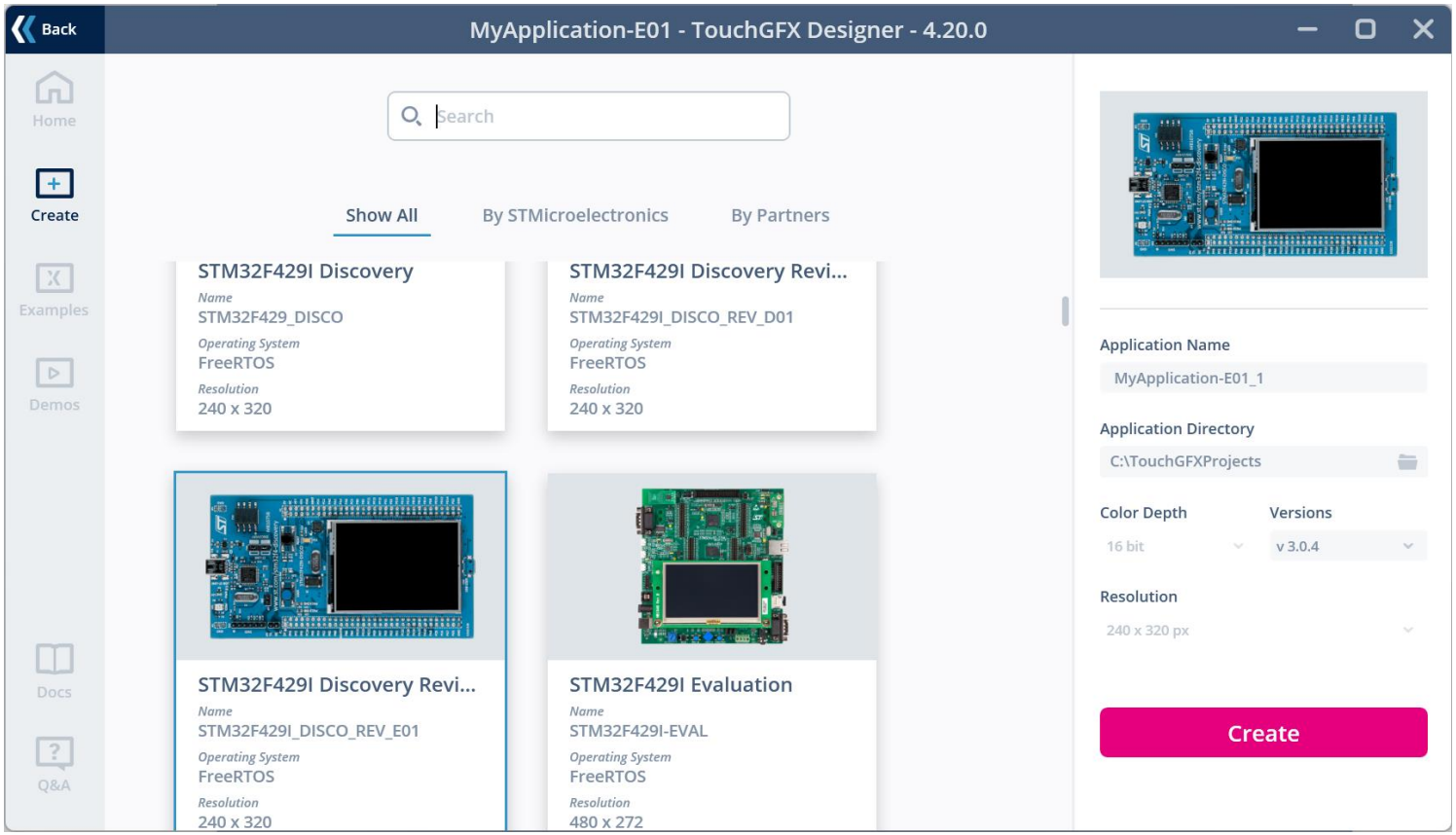
- `const osMessageQueueAttr_t attr= { .name= "Queue1" };`
- `osMessageQueueId_t osMessageQueueNew(uint32_t msg_cnt,`
  `uint32_t msg_size,`
  `const osMessageQueueAttr_t *attr);`
- `osStatus_t osMessageQueueDelete(osMessageQueueId_t queue_id);`
- `osStatus_t osMessageQueuePut(osMessageQueueId_t queue_id,`
  `const void *msg_ptr,`
  `uint8_t msg_prio,`
  `uint32_t timeout);`
- `osStatus_t osMessageQueueGet(osMessageQueueId_t queue_id,`
  `const void *msg_ptr,`
  `uint8_t *msg_prio,`
  `uint32_t timeout);`
- `uint32_t osMessageQueueGetCount(osMessageQueueId_t queue_id);`

Giới thiệu TouchGFX

- Kết hợp khả năng tính toán của CPU và tính đa nhiệm của OS → có thể xây dựng các hệ thống phức tạp hơn, hỗ trợ giao diện đồ họa.
- TouchGFX là framework do ST cung cấp, cho phép xây dựng giao diện đồ họa trên các hệ thống nhúng dựa trên STM32.
- Tích hợp chặt chẽ với bộ công cụ của ST.
  - Tạo project, cấu hình giao diện, sinh mã, chạy mô phỏng trên TouchGFX Designer.
  - Phát triển phần mềm trên STM32CubeIDE.
- Phiên bản mới nhất: 4.20.0

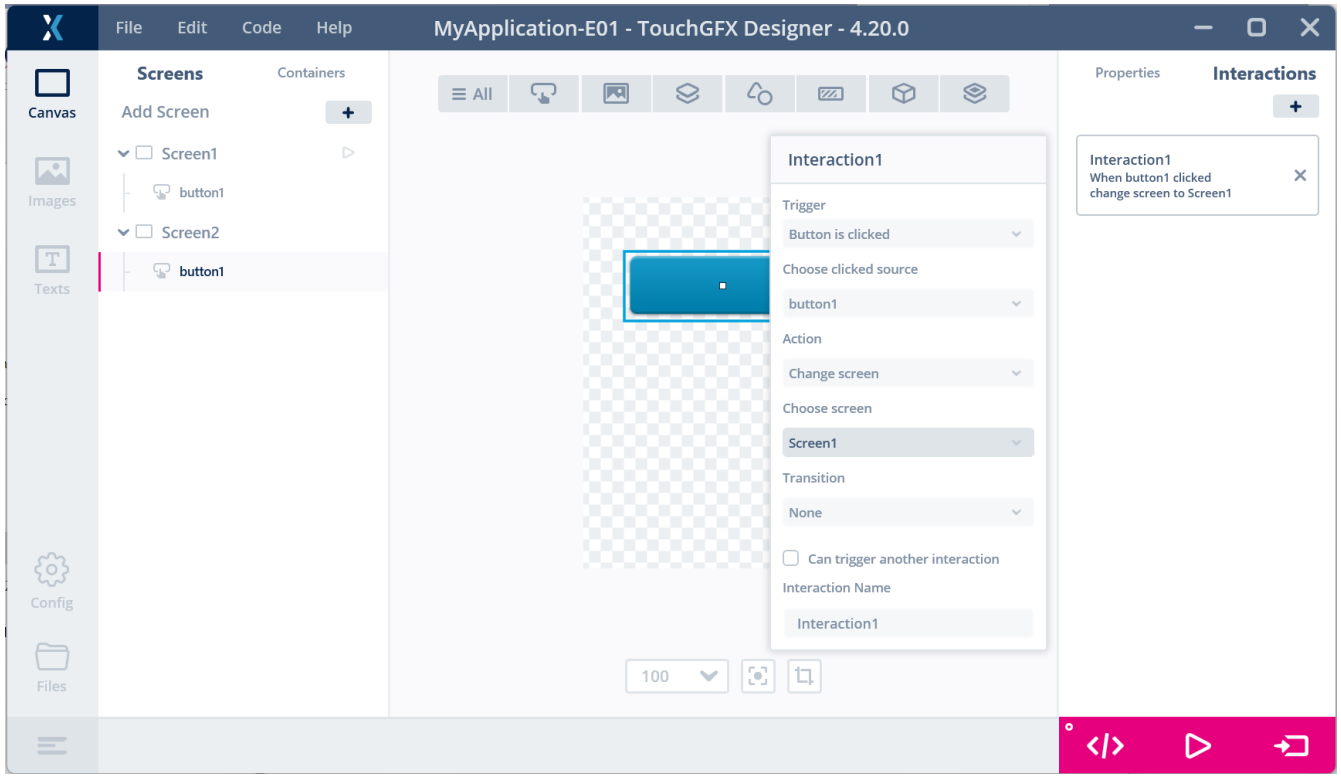
Tạo project với TouchGFX Designer

- Tạo project cho STM32F429I-DISCO
  - Note: chọn REV_E01 (xanh dương)



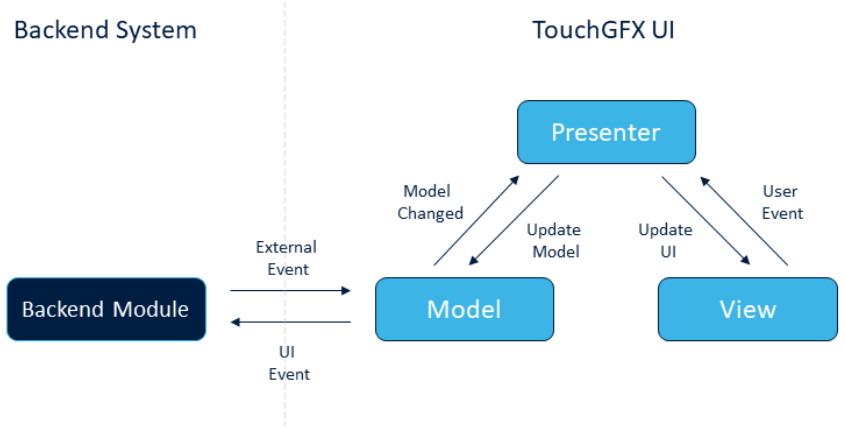
Tạo project với TouchGFX Designer

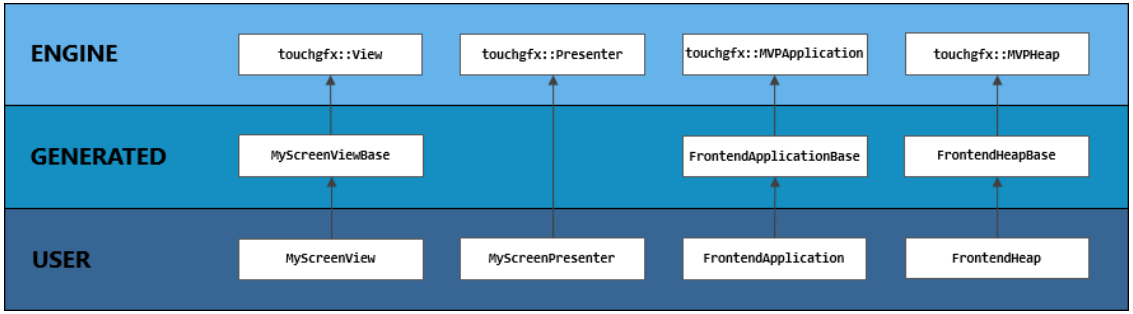
- Tạo giao diện, interaction
- Generate code, chạy thử trên board



Cấu trúc project

- STM32F429I-DISCO
  - Binaries
  - Includes
  - Application
    - Startup
    - User
      - generated
      - gui
        - FrontendApplication.cpp
        - Model.cpp
        - Screen1Presenter.cpp
        - Screen1View.cpp
        - Screen2Presenter.cpp
        - Screen2View.cpp
      - TouchGFX
      - freertos.c
      - main.c
      - stm32f4xx_hal_msp.c
      - stm32f4xx_hal_timebase_tim.c
      - stm32f4xx_it.c
      - syscalls.c
      - sysmem.c
  - Debug
  - Drivers
  - Middlewares
  - STM32F429I-DISCO.launch
  - STM32F429ZITX_FLASH.ld
  - STM32F429ZITX_RAM.ld





Animation với TouchGFX

- Timebased update (giống game loop)

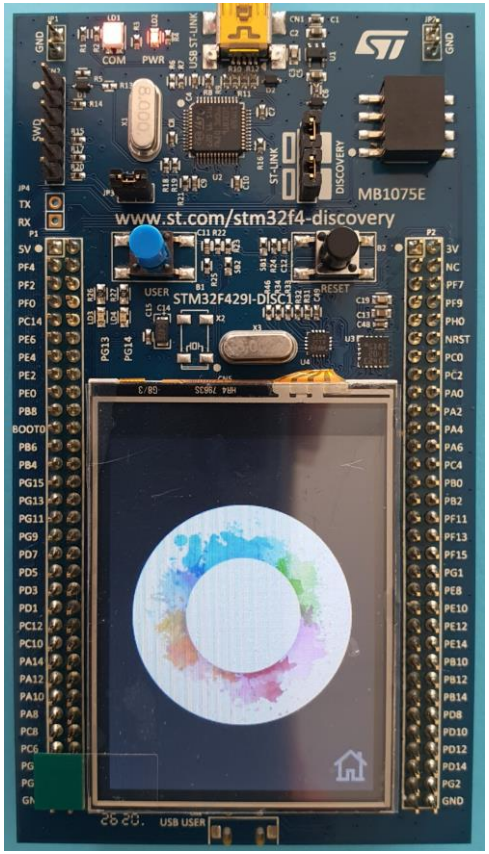
```
void handleTickEvent()
{
    Screen1ViewBase::handleTickEvent();     // Call superclass eventhandler
    tickCounter += 1;
    if (tickCounter == 600)
    {
        myWidget.startFadeAnimation(0, 20); // Fade to 0 = invisible in 20 frames
    }
}

void handleTickEvent()
{
    Screen1ViewBase::handleTickEvent();     // Call superclass eventhandler
    tickCounter += 1;
    if (tickCounter == 10)
    {
        box1.setColor(Color::getColorFromRGB(0xFF, 0x00, 0x00)); // Set color to red
        box1.invalidate();                                       // Request redraw
    }
}
```

Backend communication

- Giao tiếp giữa các đối tượng TouchGFX với hardware.
- 2 phương án:
  - Polling trực tiếp từ hàm tick của model, hoặc tick handler của view.
  - Tạo riêng 1 task giao tiếp với phần cứng, đẩy dữ liệu sang TouchGFX qua queue của FreeRTOS.

Ví dụ

- Lập trình theo mô hình MVP
- Vẽ và tạo fade animation cho hình tròn
- Di chuyển hình tròn từ trên xuống dưới
- Tô màu Box



Screen1View.hpp

```
#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP
#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>

class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    void handleTickEvent();
    uint32_t tickCount;
protected:
};
#endif // SCREEN1VIEW_HPP
```

Screen1View.cpp

```
#include <gui/screen1_screen/Screen1View.hpp>
#include <touchgfx/Color.hpp>

Screen1View::Screen1View() {
        tickCount = 0;
}
void Screen1View::setupScreen() {
    Screen1ViewBase::setupScreen();
}
void Screen1View::tearDownScreen() {
    Screen1ViewBase::tearDownScreen();
}
void Screen1View::handleTickEvent() {
    Screen1ViewBase::handleTickEvent();
    tickCount += 1;
    circle1.setY(tickCount % 320);
    invalidate();
}
```