Chương 6: FreeRTOS và TouchGFX

6.1 Giới thiệu CMSIS

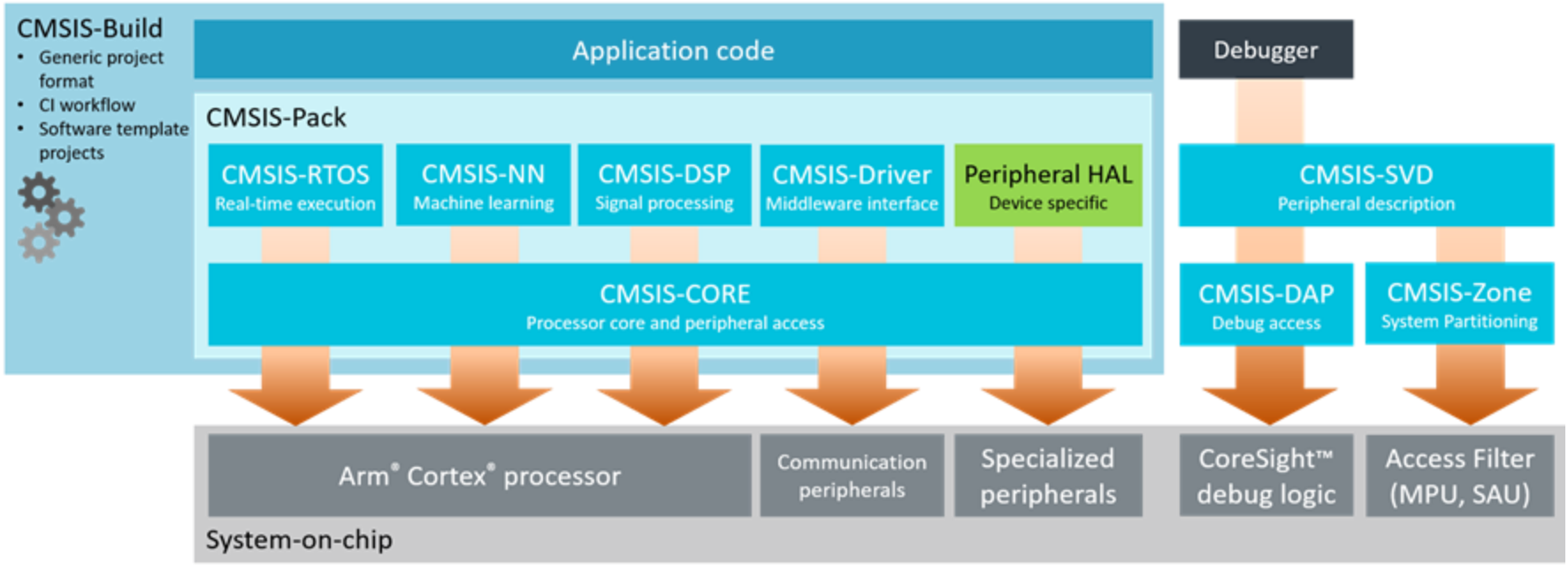
6.2 Hệ điều hành FreeRTOS và CMSIS-OS2

6.3 TouchGFX framework

Giới thiệu ARM CMSIS v5

- STM32F4 = ARM Cortex-M4 + ARM peripherals + ST (vendor) peripherals
  - ARM Cortex-M4: CPU core thiết kế bởi ARM.
  - ARM peripherals: thiết kế bởi ARM, đi cùng CPU core (system timer, NVIC, Memory Protection Unit, FPU,...)
  - ST peripherals: các thành phần do ST thiết kế.
- HAL: hardware abstraction layer. Ví dụ: thư viện HAL của ST cho điều khiển ghép nối các ST peripherals
  - Tương thích giữa các chip của ST.
  - Có thể sử dụng interface/API của CMSIS.
- CMSIS: HAL của ARM cho các chip ARM Cortex
  - Tương thích giữa các chip sử dụng ARM Cortex.
  - Phạm vi ảnh hưởng lớn hơn HAL.

Mô hình lập trình với ARM Cortex



https://arm-software.github.io/CMSIS_5/General/html/index.html

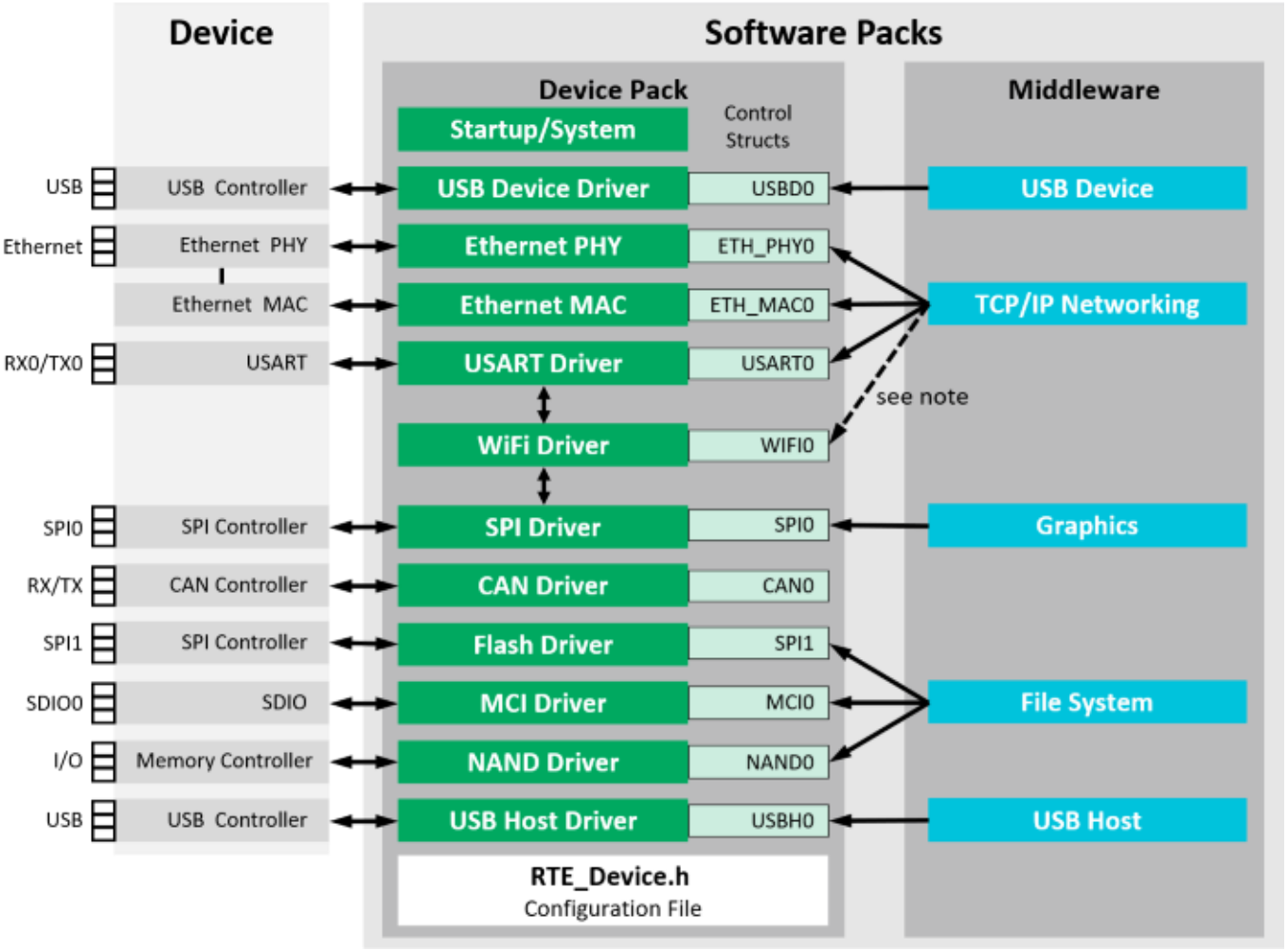
- CMSIS: vendor independent components provided by ARM.
- Peripheral HAL: provided by CPU vendor.
- Middleware: from 3rd party software vendors.

Mô hình lập trình với ARM Cortex

- CMSIS-CORE: cung cấp môi trường run-time cơ bản nhất, cho phép truy cập tới CPU và ngoại vi.
- CMSIS-Driver: generic driver cho các ngoại vi thông dụng (SPI, I2C, CAN, Ethernet,...).
- CMSIS-DSP: tập các hàm xử lý tín hiệu được tối ưu cho CPU Cortex-M và Cortex-A.
- CMSIS-NN: tập hàm hỗ trợ neural network tối ưu cho Cortex-M và Cortex-A.
- CMSIS-RTOS: cung cấp RTOS interfaces ở mức generic, cho phép ghép nối với các hệ điều hành cụ thể.

CMSIS driver

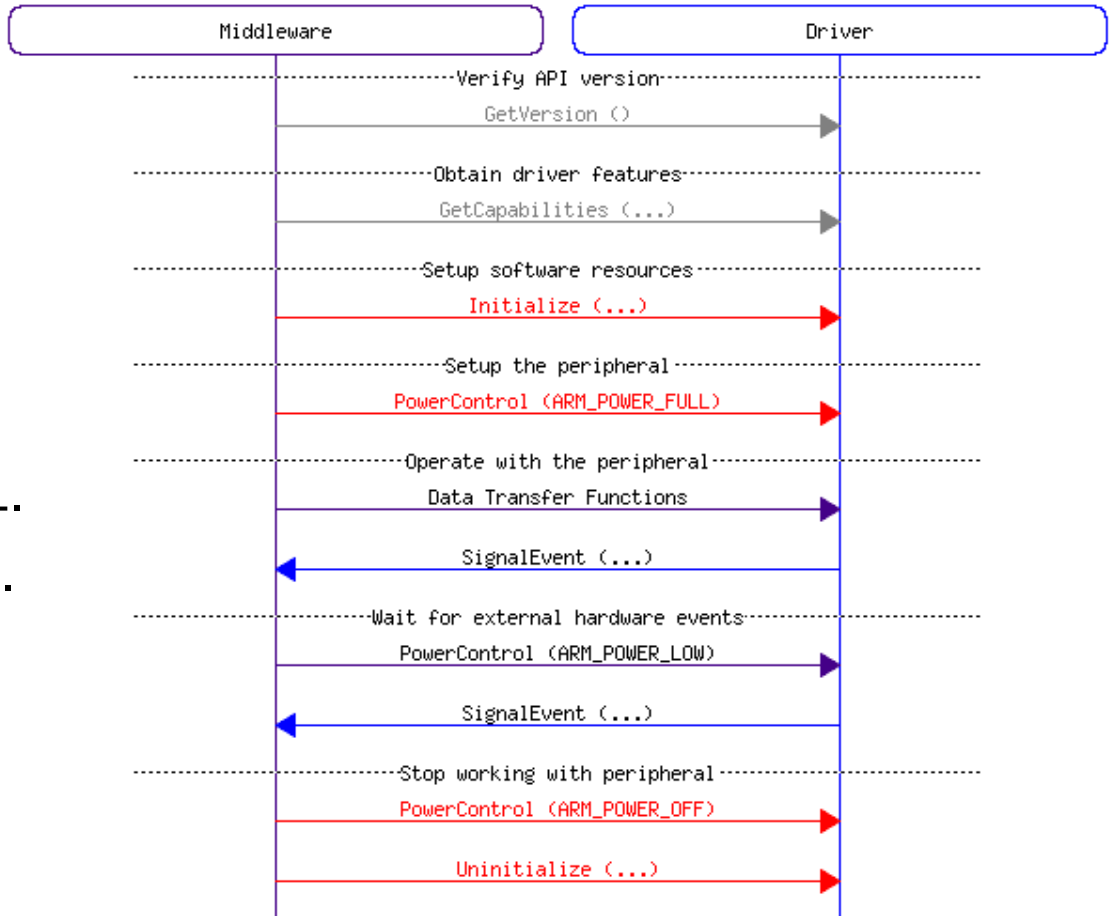
- Có thể sử dụng thay cho HAL



https://www.keil.com/pack/doc/CMSIS/Driver/html/index.html

Các hàm mặc định

- GetVersion().
- GetCapabilities().
- Initialize().
- SignalEvent().
- PowerControl()
  - ARM_POWER_FULL.
  - ARM_POWER_LOW.
  - ARM_POWER_OFF.
- Uninitialize().
- Control().



VD driver cho SPI

```
typedef struct _ARM_DRIVER_SPI {
  ARM_DRIVER_VERSION (*GetVersion) (void);
  ARM_SPI_CAPABILITIES (*GetCapabilities) (void);
  int32_t (*Initialize) (ARM_SPI_SignalEvent_t cb_event);
  int32_t (*Uninitialize) (void);
  int32_t (*PowerControl) (ARM_POWER_STATE state);
  int32_t (*Send) (const void *data, uint32_t num);
  int32_t (*Receive) ( void *data, uint32_t num);
  int32_t (*Transfer) (const void *data_out,
                       void *data_in, uint32_t num);
  uint32_t (*GetDataCount) (void);
  int32_t (*Control) (uint32_t control, uint32_t arg);
  ARM_SPI_STATUS (*GetStatus) (void);
} const ARM_DRIVER_SPI;

ARM_DRIVER_SPI Driver_SPI1; // access functions for SPI1
interface
```

CMSIS DSP lib

- Basic math functions
- Fast math functions
- Complex math functions
- Filtering functions
- Matrix functions
- Transform functions
- Motor control functions
- Statistical functions
- Support functions
- Interpolation functions
- Support Vector Machine functions (SVM)
- Bayes classifier functions
- Distance functions
- Quaternion functions

FreeRTOS

- Ưu điểm của CMSIS và HAL lib:
  - Cho phép giao tiếp với phần cứng/ngoại vi bằng các API hoặc driver đã chuẩn hóa.
  - Tương thích tốt giữa các dòng chip hoặc các thiết kế mạch khác nhau.
- Vấn đề:
  - Sử dụng main loop + interrupt + DMA để tối ưu tài nguyên CPU.
  - Nhưng vẫn khó tận dụng hết tài nguyên tính toán của CPU.

➔ cần OS để triển khai mô hình thực thi đa luồng.

FreeRTOS

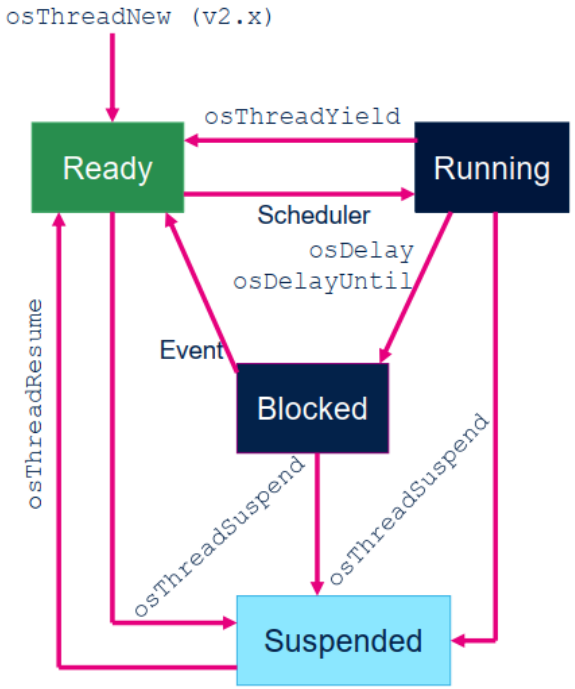
- Real-time OS cho hệ nhúng kích thước nhỏ và vừa.
- Mã nguồn mở, phát triển bởi Richard Barry (2003).
- Maintain bởi Amazon Web Services (2017).
- Hỗ trợ nhiều dòng chip:
  - 40+ architectures, 15+ toolchains.
  - Supported Devices - FreeRTOS™

FreeRTOS

- Đặc điểm:
  - Nhỏ gọn và tin cậy.
  - Cộng đồng sử dụng lớn.
  - Hỗ trợ cả phiên bản thương mại.
  - FreeRTOS trên STM32F4: 10 KB ROM
- Hỗ trợ đầy đủ các tính năng của hệ điều hành
  - Quản lý task: tạo task, lập lịch thực thi, quản lý độ ưu tiên task.
  - Quản lý bộ nhớ: cấp phát và thu hồi bộ nhớ.
  - Hỗ trợ đồng bộ, truyền thông giữa các task: semaphore, queue,..

FreeRTOS task

- Task là luồng thực thi nằm trong 1 hàm, trong đó có chứa vòng lặp vô hạn.
- Task có vùng stack và độ ưu tiên riêng.
- Có thể được tạo và xóa động.
- Vòng đời gồm 4 trạng thái



Các API chính của FreeRTOS

- Tạo task

```
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 param
  - *uxPriority* : handle

Ví dụ: tạo task với API của FreeRTOS

```
/* 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
  - vTaskDelayUntil
  - uxTaskPriorityGet
  - vTaskPrioritySet
  - vTaskSuspend
  - vTaskResume
  - xTaskResumeFromISR
  - xTaskAbortDelay

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**

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

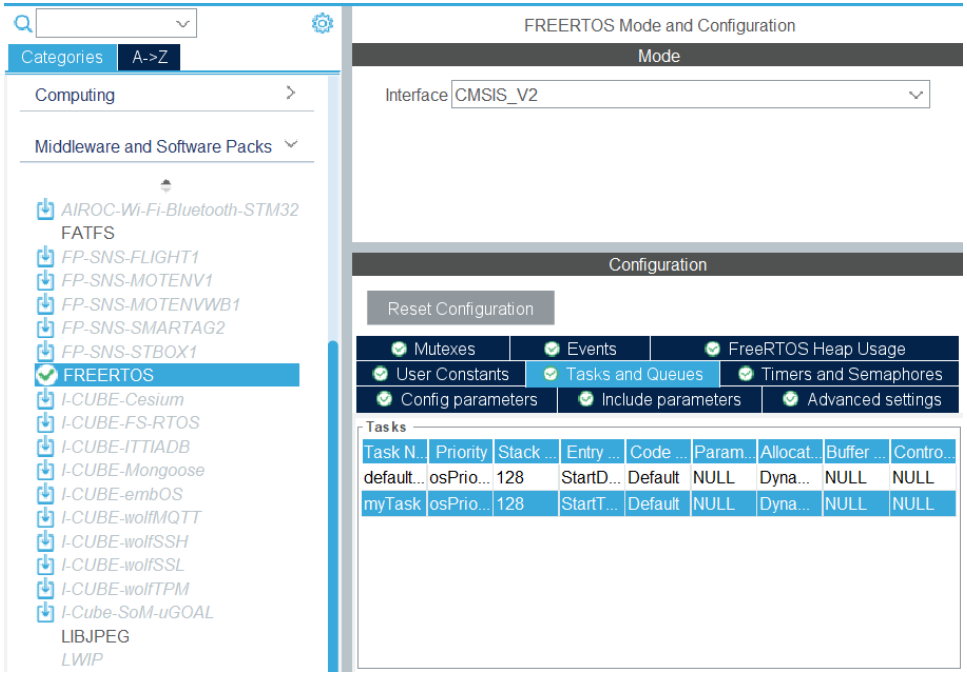
- Ví dụ
  - `osThreadId_t Task1Handle;`
  - `osThreadId_t osThreadNew(osThreadFunc_t func, void *argument, const osThreadAttr_t* attr)`
  - `osStatus_t osThreadTerminate(osThreadId_t thread_id)`
  - `osStatus_t osThreadYield(void)`
  - `osStatus_t osThreadResume(osThreadId_t thread_id)`
  - `osThreadState_t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

Ví dụ 1

- Xây dựng application với 2 task
  - Task 1: nháy đèn LED3 tần số 1 Hz
  - Task 2: gửi dữ liệu về PC qua ST-LINK virtual COM port

Ví dụ 1

- Tạo project mới cho STM32F429ZIT6
  - Cấu hình PG13 và PG14 là Output.
  - Cấu hình RCC và System clock để CPU chạy ở tần số 90 MHz. Chú ý Input frequency = 8 MHz ở Clock configuration.
  - Cấu hình USART1 ở chế độ Asynchronous 115200 bps.
  - Timebase source trong SYS: TIM6 (vì Systick đã dùng cho FreeRTOS).
- Cấu hình FreeRTOS trong tab Middleware



Sử dụng CMSIS_V2 interface

Khai báo thêm task myTask, đặt độ ưu tiên 2 task bằng nhau

Ví dụ 1

- Lập trình thân hàm tương ứng của 2 task

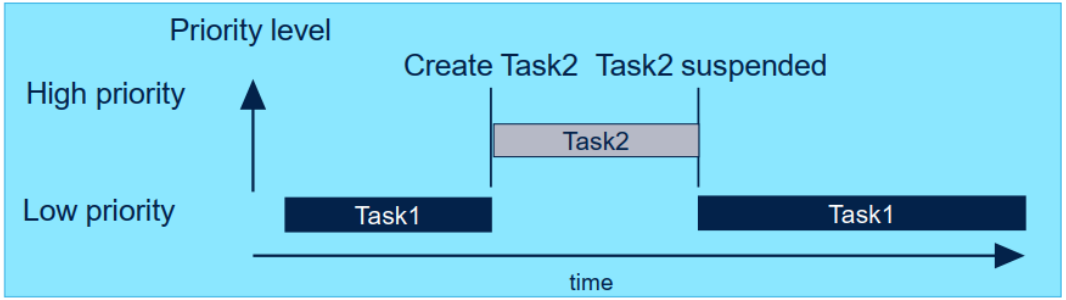
```
289 /* USER CODE BEGIN Header_StartDefaultTask */
290 /**
291   * @brief  Function implementing the defaultTask thread.
292   * @param  argument: Not used
293   * @retval None
294   */
295 /* USER CODE END Header_StartDefaultTask */
296 void StartDefaultTask(void *argument)
297 {
298   /* USER CODE BEGIN 5 */
299   /* Infinite loop */
300   for(;;)
301   {
302     osDelay(500);
303     HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_14);
304   }
305   /* USER CODE END 5 */
306 }
307
308 /* USER CODE BEGIN Header_StartTask02 */
309 /**
310 * @brief Function implementing the myTask thread.
311 * @param argument: Not used
312 * @retval None
313 */
314 /* USER CODE END Header_StartTask02 */
315 void StartTask02(void *argument)
316 {
317   /* USER CODE BEGIN StartTask02 */
318   /* Infinite loop */
319   for(;;)
320   {
321     HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
322     HAL_UART_Transmit(&huart1, "Hello from user task\n", 21, 10);
323
324     osDelay(500);
325   }
326   /* USER CODE END StartTask02 */
327 }
```

Ví dụ 1:

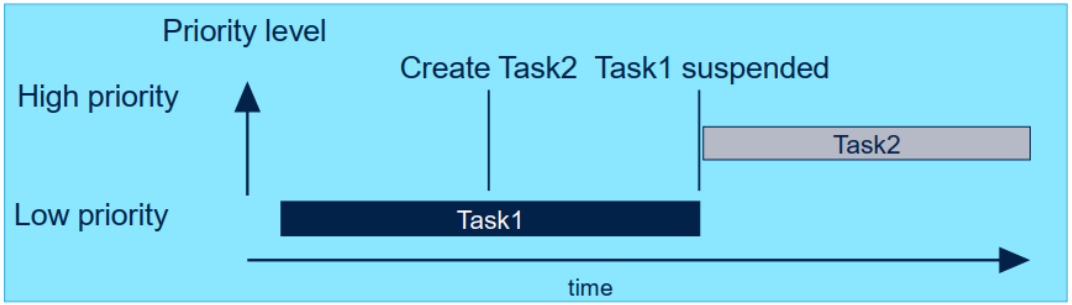
- Chạy và quan sát hoạt động của mạch:
  - Đèn LED nháy với tần số 1 Hz.
  - Dòng chữ “Hello from user task” được gửi liên tục về Hercules.
- 2 task này chạy song song, giống như có 2 main loop cùng chạy.
  - ➔ có thể tận dụng tài nguyên CPU tốt hơn để nhiều tác vụ có thể thực hiện đồng thời.

Task scheduling và context switching

- Scheduling: lập lịch thực thi cho các task
  - Preemtive: task có độ ưu tiên cao hơn được phép ngắt task có độ ưu tiên thấp hơn. Các task cùng độ ưu tiên được lập lịch theo round robin (xem học phần Hệ điều hành).
  - Non preemtive: task đến trước được thực thi trước.



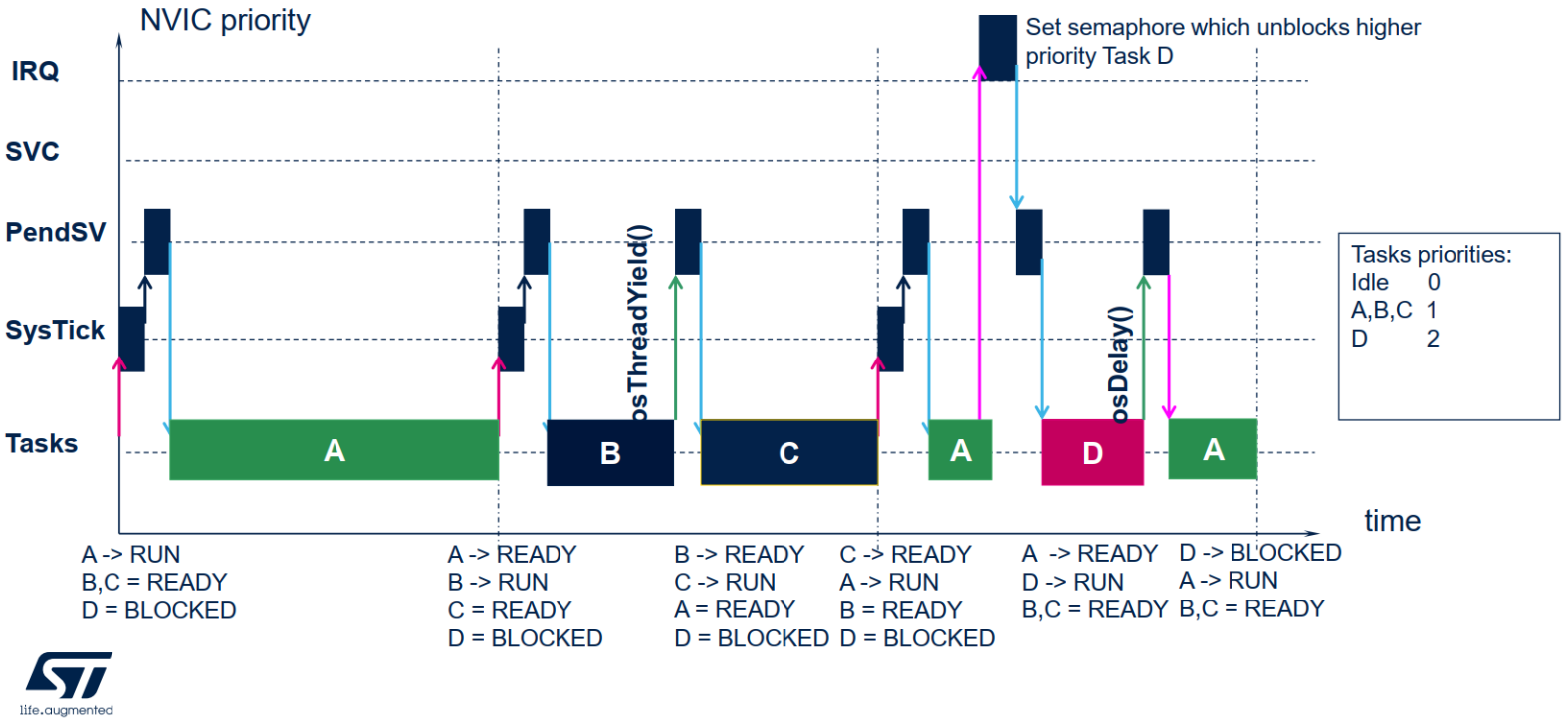
Pre-emtive scheduling



Non Pre-emtive scheduling

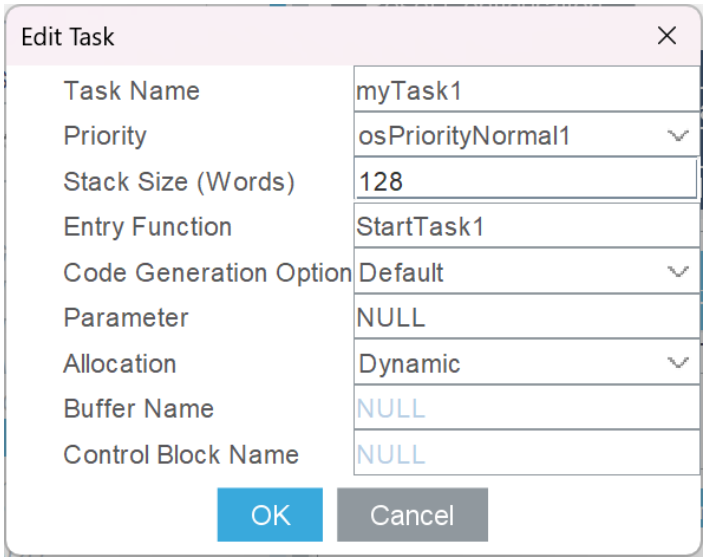
Task scheduling và context switching

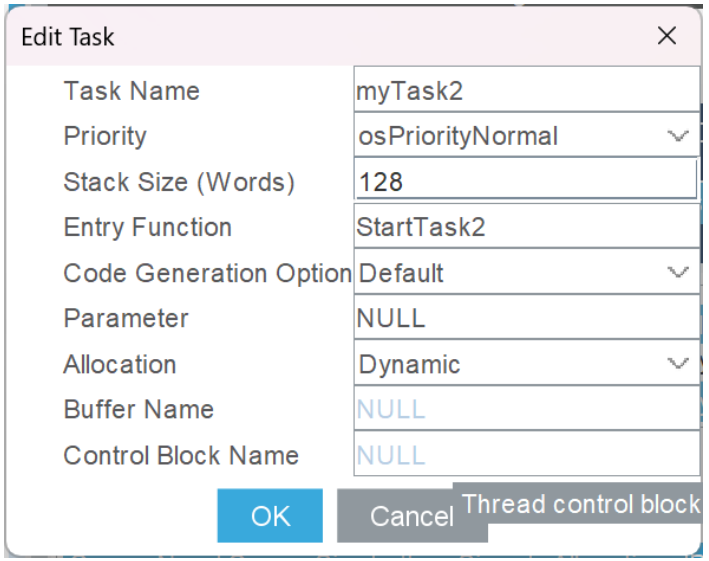
- Context switching
  - CPU ngừng thực thi 1 task, chuyển sang thực thi task khác.
  - Xảy ra khi có sự kiện phù hợp: Systick, osDelay() call, thread yield, interrupt, semaphore block,...
  - Chú ý: hàm HAL_Delay() là hàm polling, sẽ chạy liên tục với độ ưu tiên của task.



Ví dụ 2: Lịch thực thi của task và độ ưu tiên

- Tạo project mới (hoặc cấu hình lại project 1)
  - Bật RCC + cấu hình clock cho CPU chạy ở 90 MHz.
  - Đặt Timebase source trong SYS là TIM6.
  - Enable USART1, đặt PG13 và PG14 là GPIO_Output.
  - Bật FreeRTOS trong Middleware and Software pack.
    - Tạo 2 task myTask1 và myTask2.
    - Chú ý: độ ưu tiên của myTask1 là osPriorityNormal1 (cao hơn so với myTask2).





Ví dụ 2: Lịch thực thi của task và độ ưu tiên

- Sinh code và lập trình thêm các hàm

```
71  /* USER CODE BEGIN PFP */
72  void SendChar(char c)
73  {
74      uint8_t str[] = {c, '\r', '\n'};
75      HAL_UART_Transmit(&huart1, str, 3, 10);
76  }
77  /* USER CODE END PFP */
```

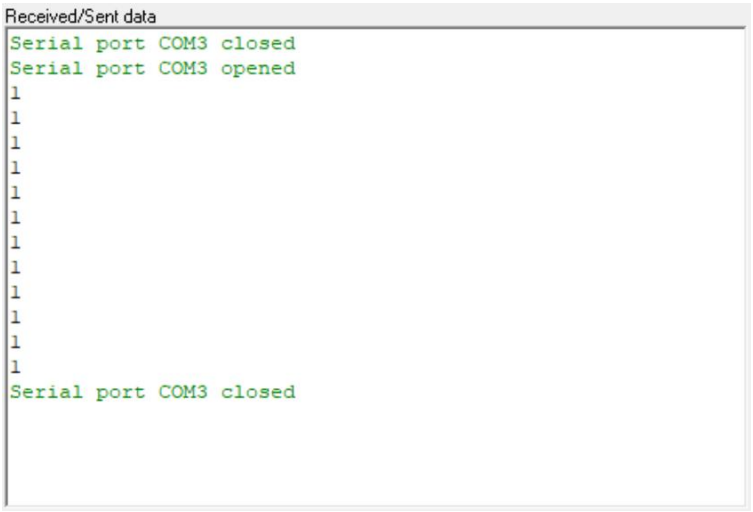
```
294 void StartTask1(void *argument)
295 {
296   /* USER CODE BEGIN 5 */
297   /* Infinite loop */
298   for(;;)
299   {
300       SendChar('1');
301       HAL_Delay(200);
302       //osDelay(200);
303   }
304   /* USER CODE END 5 */
305 }
```

```
314 void StartTask2(void *argument)
315 {
316   /* USER CODE BEGIN StartTask2 */
317   /* Infinite loop */
318   for(;;)
319   {
320       SendChar('2');
321       osDelay(200);
322   }
323   /* USER CODE END StartTask2 */
324 }
```

Chú ý Task1 routine dùng HAL_Delay()

Ví dụ 2: Lịch thực thi của task và độ ưu tiên

- Nạp code lên board và chạy. Quan sát kết quả trên Hercules.
  - Nội dung hiện trên Hercules toàn là ký tự 1, chứng tỏ Task 2 không được thực thi.
  - Lý do: Task 1 độ ưu tiên cao hơn và lại dùng hàm HAL_Delay(), nên sẽ chiếm 100% thời gian của CPU.
- Để task 2 có thể được lập lịch thực thi.
  - Giữ nguyên độ ưu tiên của Task 1, và dùng osDelay() thay cho HAL_Delay().
  - Hoặc giữ nguyên code của Task 1, đổi độ ưu tiên của Task 1 xuống nhỏ hơn hoặc bằng Task 2.



➔*Chú ý: task loop luôn phải có osDelay()*

Ví dụ 3: Tạo/xóa task

- Lập trình để các Task hoạt động như sau.
  - Task 1: lần lượt gửi các ký tự ‘a’, ‘b’,... về PC qua UART với chu kỳ 1000 ms một ký tự.
  - Task 2: nhận sự kiện bấm nút PA0, nếu nút được bấm thì Task 1 chuyển sang gửi chuỗi ký tự ‘A’, ‘B’,...
  - Chú ý: thay đổi cần được thực hiện ngay lập tức mà không cần chờ hết chu kỳ 1000 ms của Task 1.

Ví dụ 3: Tạo/xóa task

- Kế thừa project trên, thêm cấu hình chân PA0 là GPIO_Input.
- Biến global task1Count: giá trị đếm.
- Task 1: nhận tham số khi khởi tạo task.
- Task 2: phát hiện sự kiện nút bấm, xóa và tạo lại task 1 khi có nút bấm.

```
71  /* USER CODE BEGIN PFP */
72  void SendChar(char c)
73  {
74      uint8_t str[] = {c, '\r', '\n'};
75      HAL_UART_Transmit(&huart1, str, 3, 10);
76  }
77  /* USER CODE END PFP */
78
79  /* Private user code ------------------------
80  /* USER CODE BEGIN 0 */
81  uint8_t task1Count = 'a';
82  /* USER CODE END 0 */
```

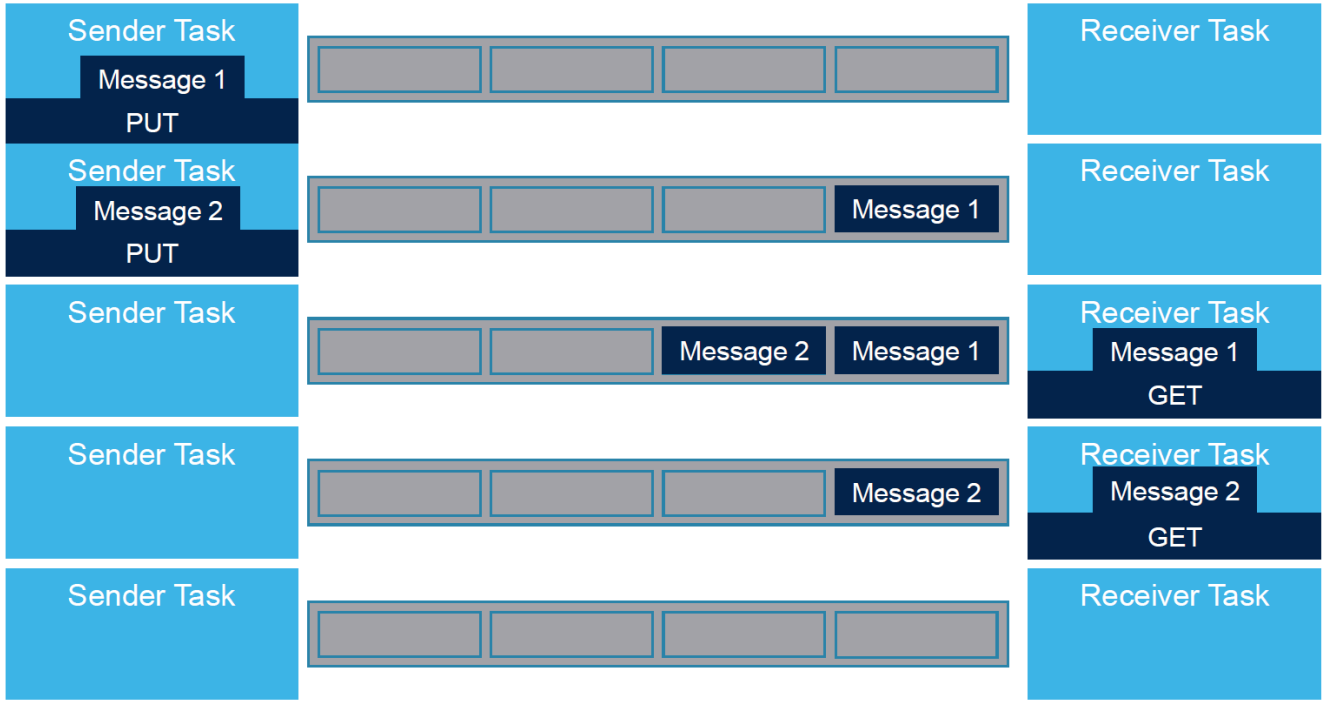
```
288 /* USER CODE END Header_StartTask01 */
289 void StartTask01(void *argument)
290 {
291   /* USER CODE BEGIN 5 */
292     if (argument == NULL)
293         task1Count = 'a';
294     else
295     {
296         uint8_t *p = (uint8_t *)argument;
297         task1Count = *p;
298     }
299   /* Infinite loop */
300   for(;;)
301   {
302       SendChar(task1Count);
303       task1Count++;
304       osDelay(1000);
305   }
306   /* USER CODE END 5 */
307 }
315 /* USER CODE END Header_StartTask02 */
316 void StartTask02(void *argument)
317 {
318   /* USER CODE BEGIN StartTask02 */
319   /* Infinite loop */
320   for(;;)
321   {
322       if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)
323               == GPIO_PIN_SET)
324       {
325           uint8_t c = 'A';
326           osThreadTerminate(myTask01Handle);
327           myTask01Handle = osThreadNew(StartTask01, &c,
328                   &myTask01_attributes);
329           osDelay(500);
330       }
331     osDelay(1);
332   }
333   /* USER CODE END StartTask02 */
334 }
```

RTOS kernel primitives

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

Truyền thông giữa các task

- IPC: inter-process communication, cho phép truyền dữ liệu giữa các tiến trình.
  - Có nhiều phương án: Queue, Queue sets, Buffer, Semaphore...
- 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);`

Ví dụ 4

- Xây dựng application với 2 task
  - Task 1: nháy đèn LED3 tần số 1 Hz
  - Task 2: gửi dữ liệu về PC qua ST-LINK virtual COM port
- Mở rộng:
  - Tạo task riêng liên tục polling trạng thái nút USER_BUTTON.
  - Nếu nút được bấm thì gửi dữ liệu về PC.
  - Việc gửi dữ liệu về PC do task 2 phụ trách.
- Sử dụng message queue gửi dữ liệu giữa các task

Ví dụ 4

- Ví dụ 1 (đã làm): Xây dựng application với 2 task
  - Task 1: nháy đèn LED3 tần số 1 Hz
  - Task 2: gửi dữ liệu về PC qua ST-LINK virtual COM port
- Mở rộng:
  - Tạo task riêng Task 3 liên tục polling trạng thái nút USER_BUTTON.
  - Nếu nút được bấm thì gửi dữ liệu về PC.
  - Việc gửi dữ liệu về PC do task 2 phụ trách.
- Tạo thêm message queue gửi dữ liệu giữa các task.

```c
void StartTask03(void *argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  for(;;)
  {
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET)
    {
        uint8_t data = 'X';
        osMessageQueuePut(myQueue01Handle, &data, 0, 10);
        osDelay(200);
    }
    osDelay(1);
  }
  /* USER CODE END StartTask03 */
}
```

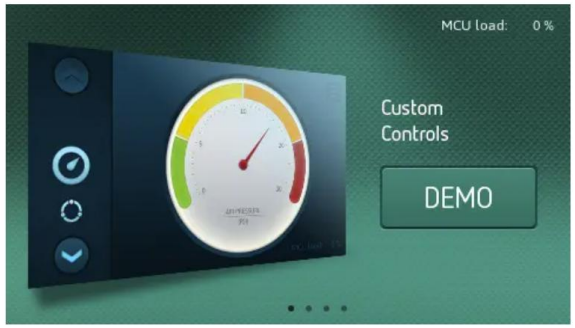
```c
void StartTask02(void *argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
      uint8_t res;
      if (osMessageQueueGetCount(myQueue01Handle) > 0)
      {
          osMessageQueueGet(myQueue01Handle, &res, NULL, osWaitForever);
          if (res == 'X')
              HAL_UART_Transmit(&huart1, "Button pressed\n", 15, 10);
      }

    osDelay(1);
  }
  /* USER CODE END StartTask02 */
}
```

Embedded graphics với 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: 4.25.0

Embedded graphics với TouchGFX

- Đặc điểm embedded graphics
  - Giao diện đồ họa trên các hệ nhúng ít tài nguyên, không có GPU.
  - Độ sâu màu thấp: grayscale, RGB 8 bit hoặc RGB 16 bit.
  - Độ phân giải màn hình nhỏ, kích thước RAM nhỏ.
- TouchGFX:
  - Pixel format tiêu chuẩn RGB565.
  - Single hoặc double frame buffer.
  - Sử dụng FreeRTOS để quản lý task.
  - Các bước phát triển ứng dụng.

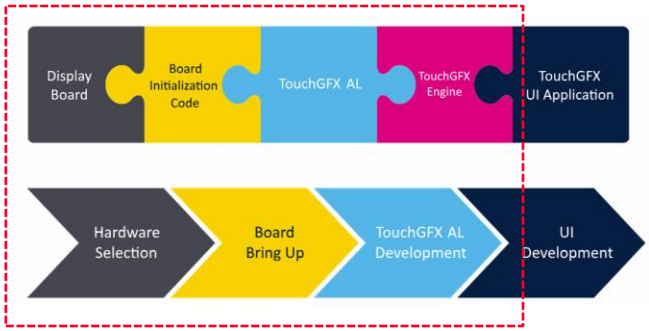


Original RGB888 image

Converted RGB565 images with and without dithering

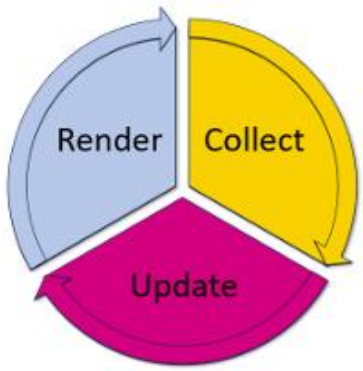
Chất lượng hình ảnh theo pixel format

Được cấu hình sẵn cho các development board



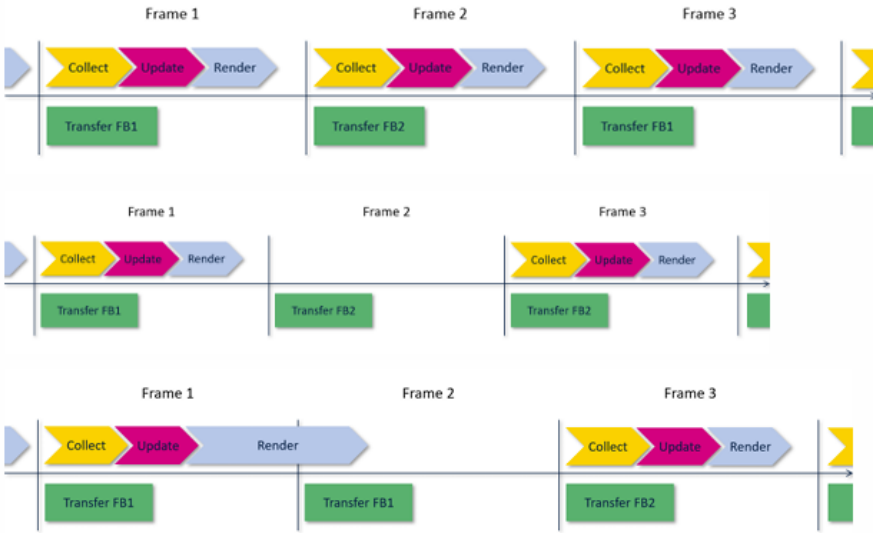
Embedded graphics với TouchGFX

- Main loop



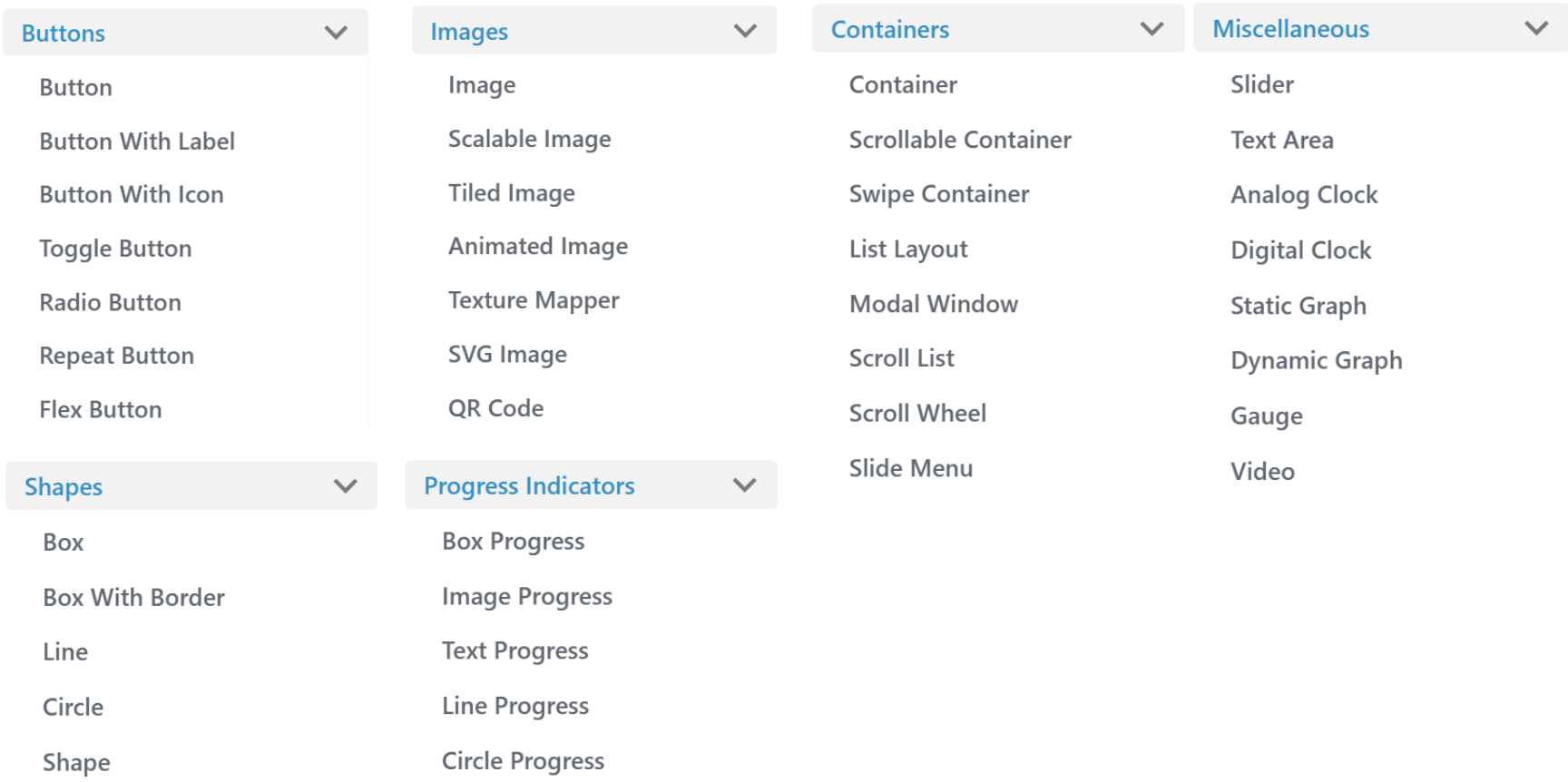
1. Collecting events
   - Touchscreen and buttons
   - Backend (Ticks)
2. Updating
   - Changes the “state” of the widgets based on the events collected
3. Rendering
   - TouchGFX draws relevant updates in the framebuffer

- Refresh rate
  - Theo thông số của màn hình (thường ~60 Hz).
  - Sự kiện tick() trên UI.
  - Tính toán chậm sẽ làm giảm framerate thực.



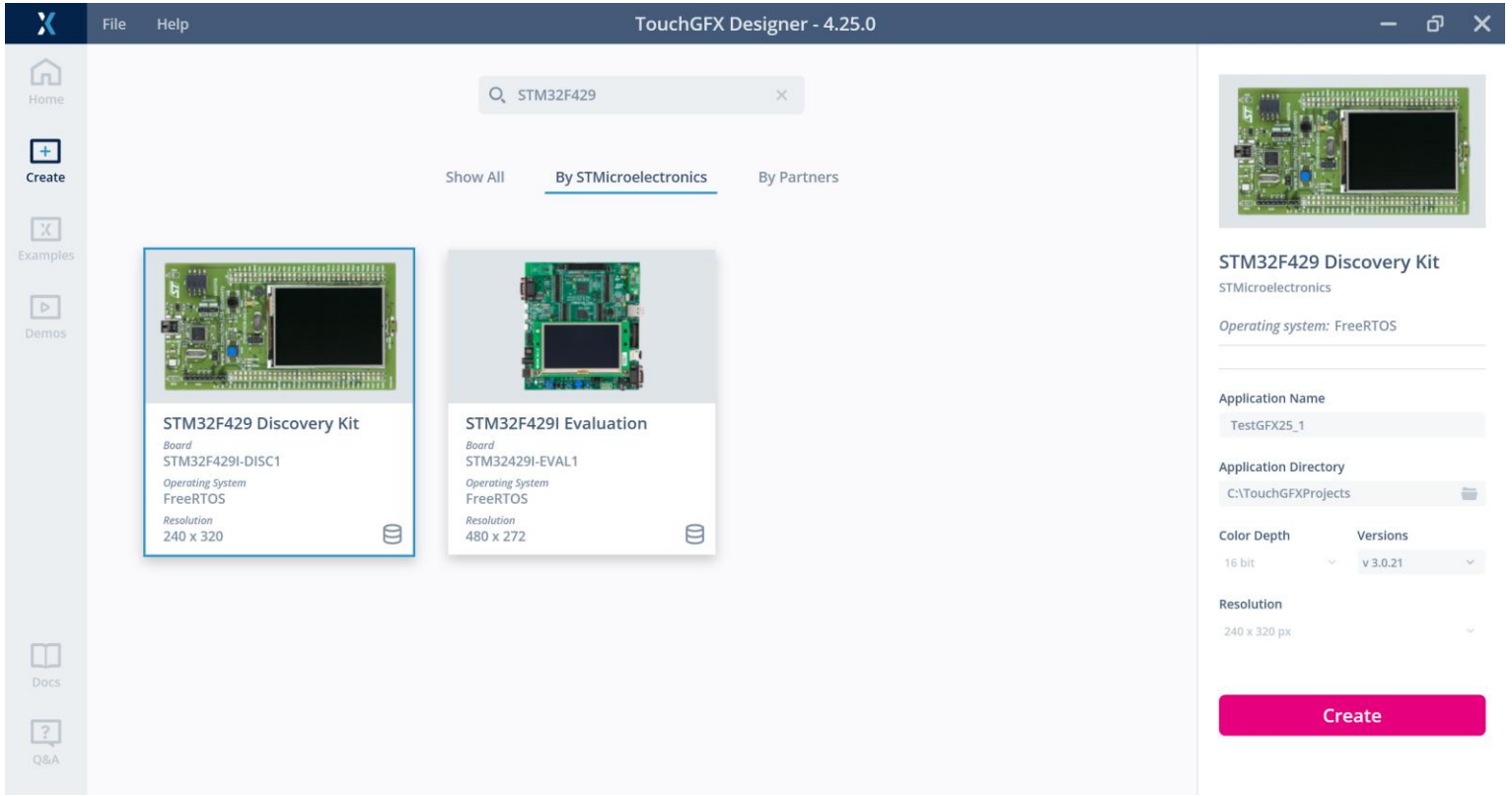
Embedded graphics với TouchGFX

- Các UI components/widgets: tham khảo UI Components | TouchGFX Documentation



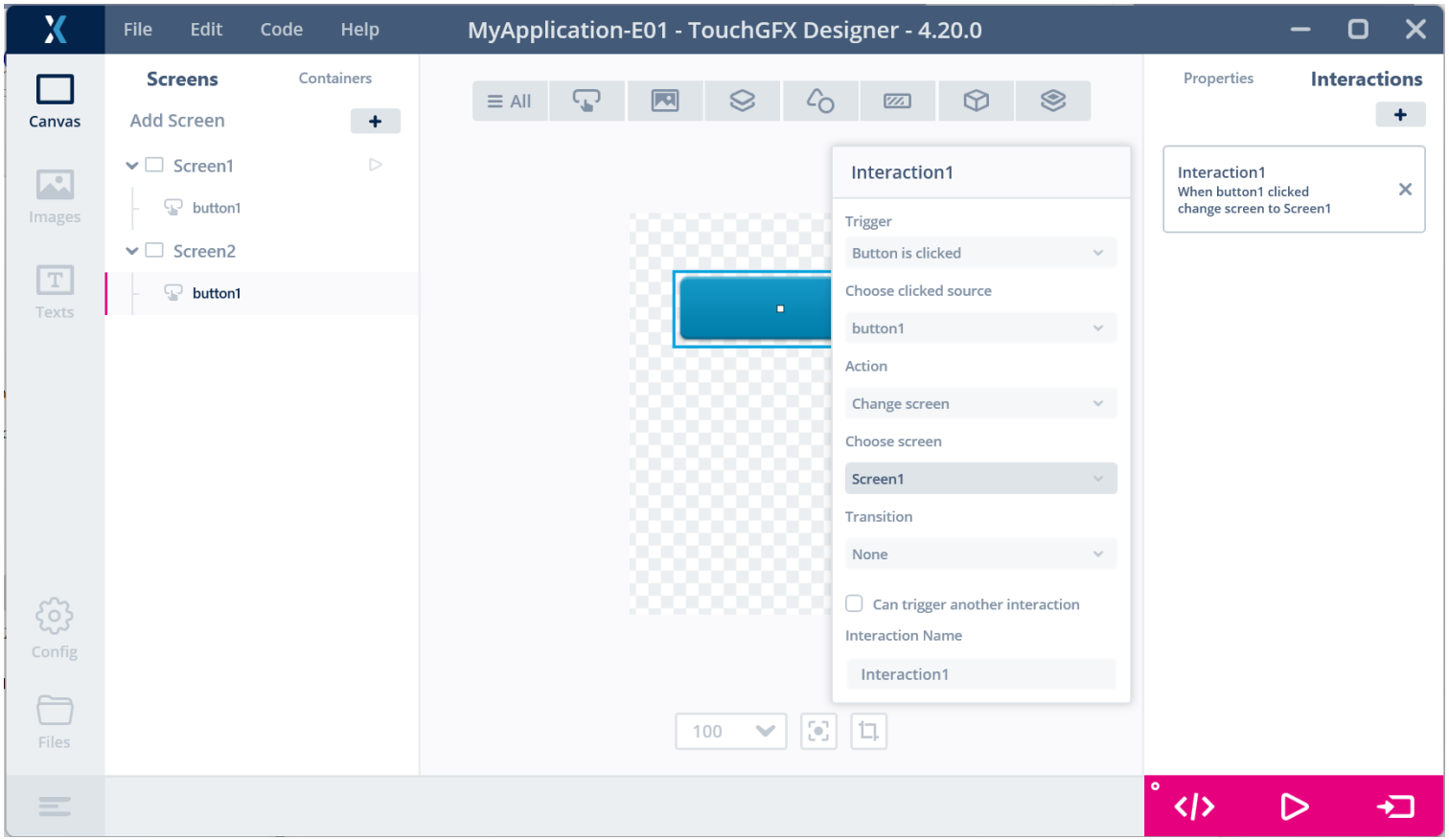
Ví dụ: Tạo project với TouchGFX Designer

- Tạo project cho STM32F429I-DISCO



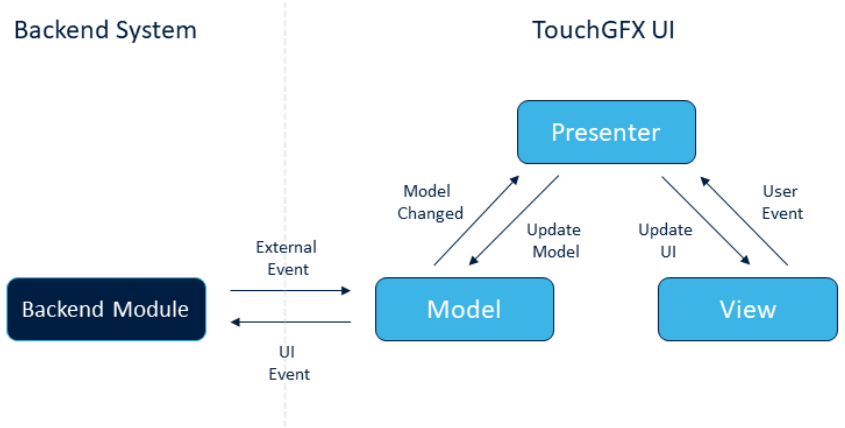
Tạo project với TouchGFX Designer

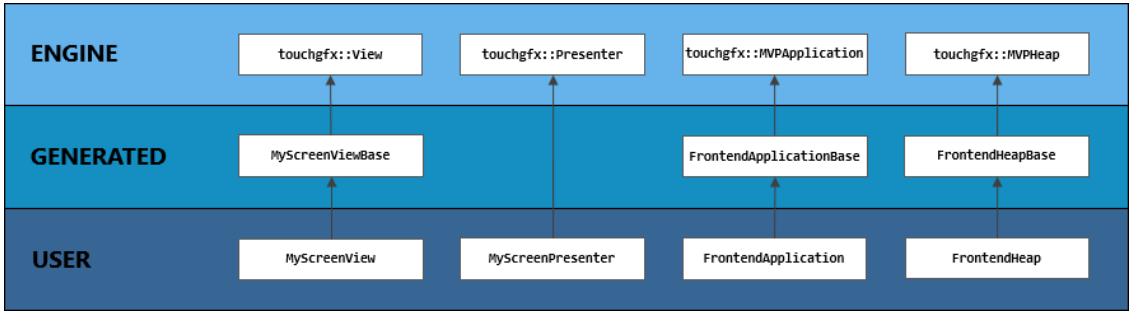
- Tạo giao diện, interaction
- Generate code, chạy thử trên board



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





Bài tập

- Hãy tìm hiểu các đoạn code khởi tạo trong hàm main

Animation với TouchGFX

- Timebased update (giống game loop)
- Vd:

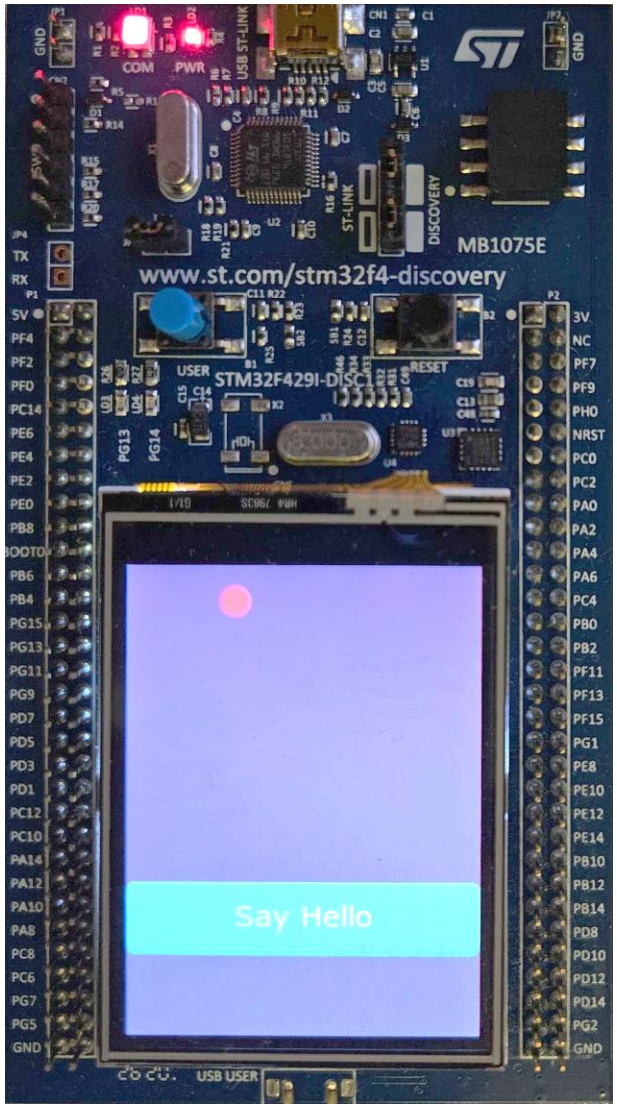
```
void handleTickEvent()
{
    Screen1ViewBase::handleTickEvent();     // Call superclass eventhandler
    tickCounter += 1;
    if (tickCounter == 600)
    {
        myWidget.startFadeAnimation(0, 20); // Fade to 0 = invisible in 20 frames
    }
}

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.

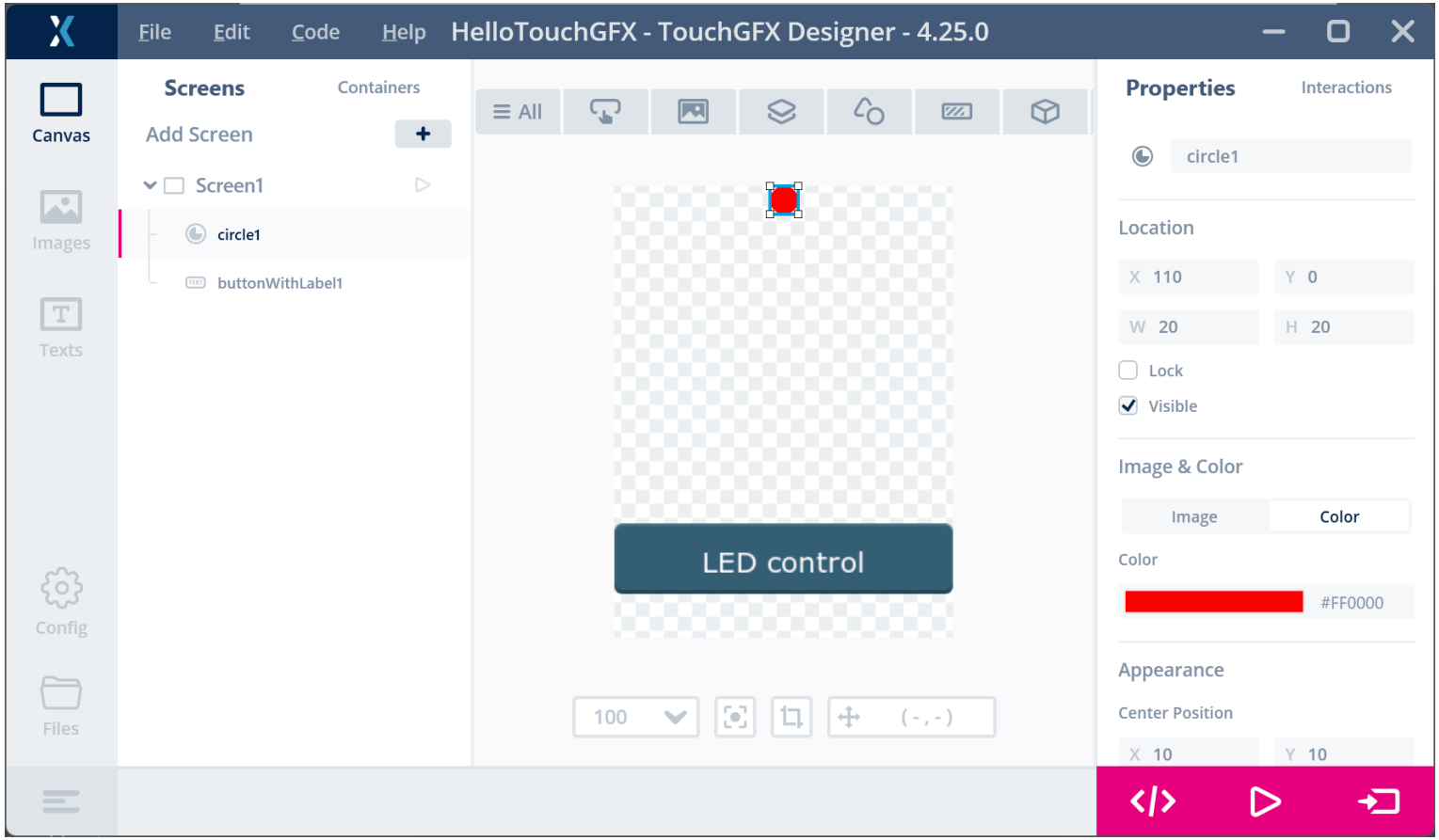
Bài tập

- Thiết kế giao diện với 2 widget như hình.
- Lập trình thực hiện
  - Khi bấm nút mềm trên giao diện thì đèn LED bật/tắt.
  - Khi bấm nút cứng PA0 (nút xanh) thì hình tròn đỏ di chuyển.



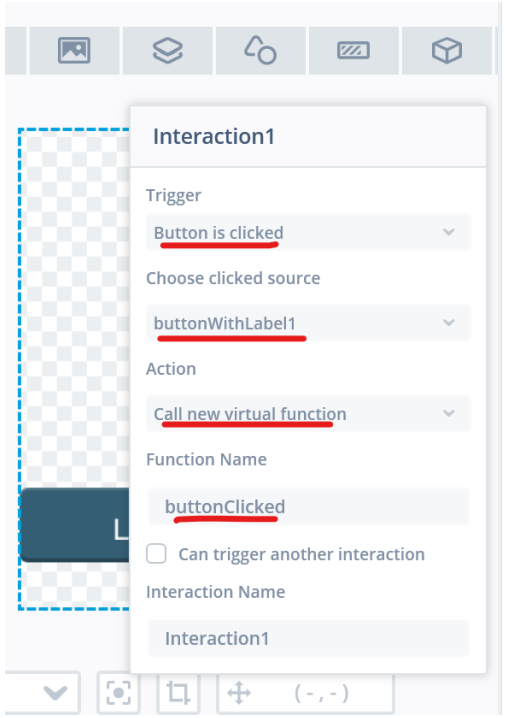
Bài tập

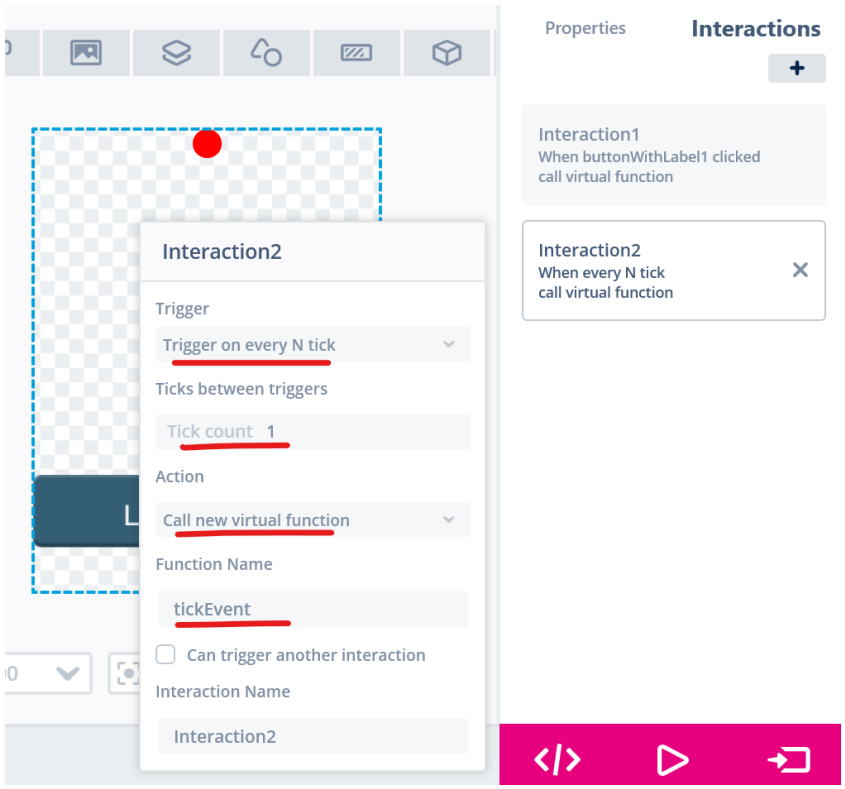
- Tạo project mới trong TouchGFX Designer (chọn STM32F429I-DISCO Rev E).
- Project có 1 screen với 2 widget buttonWithLabel1 và circle1.



Bài tập

- Tạo các Interactions (sự kiện).
- Sinh code (bấm F4).





Bài tập

- Cập nhật hàm MX_GPIO_Init() trong hàm main để khởi tạo chân PA0 là GPIO_Input, và PG13, PG14 là GPIO_Output (tham khảo code các tuần trước).

```
/* USER CODE BEGIN MX_GPIO_Init_2 */
  /*Configure LED pins : PG13 PG14 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
/* USER CODE END MX_GPIO_Init_2 */
```

Bài tập

- Lập trình hàm của default task để bắt sự kiện bấm nút PA0 và gửi dữ liệu vào queue khi nút được bấm.

```
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
        if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET)
        {
            uint8_t data = 'X';
            osMessageQueuePut(myQueue01Handle, &data, 0, 10);
        }

    osDelay(1);
  }
  /* USER CODE END 5 */
}
```

Bài tập

- Thêm biến tickCount vào khai báo lớp Screen1View

```
class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();

    void buttonClicked();
    void tickEvent();
protected:
    uint32_t tickCount = 0;
};
```

Bài tập

- Xử lý các hàm buttonClicked() và tickEvent() trong lớp Screen1View

```
void Screen1View::buttonClicked()
{
    HAL_GPIO_TogglePin(GPIOG, GPIO_PIN_13);
}

extern osMessageQueueId_t myQueue01Handle;
void Screen1View::tickEvent()
{
    tickCount += 2;
    tickCount = tickCount % 240;

    float x = tickCount / 55.0f;
    float y = sin(x)+sin(2*x)+sin(3*x)+1;

    uint8_t res;
    if (osMessageQueueGetCount(myQueue01Handle) > 0)
    {
      osMessageQueueGet(myQueue01Handle, &res, NULL, osWaitForever);
      if (res == 'X')
      {
          circle1.moveTo((int16_t)floor(x*55),200-(int16_t)floor(y*50));
          circle1.invalidate();
      }
    }
}
```

Bài tập

- Dịch và chạy trên kit, quan sát kết quả.
- Giải thích hoạt động của chương trình.