Bài thực hành 3

XÂY DỰNG ỨNG DỤNG TRÊN HỆ NHÚNG ARM Cortex-M4

1. Mục tiêu

- Tìm hiểu vi điều khiển STM32F4 và làm quen các công cụ phát triển.
- Tìm hiểu hệ điều hành FreeRTOS.
- Lập trình xây dựng ứng dụng với framework TouchGFX.
- Xây dựng ứng dụng đa chức năng:
    + Ghép nối ngoại vi với STM32F4.
    + Giao diện đồ họa, thực thi đa nhiệm với FreeRTOS và TouchGFX.

2. Chuẩn bị

- Tài liệu thực hành, Kit STM32F429I-DISC1, mini-USB cable.
- Các phần mềm: STM32CubeIDE 1.12.1, TouchGFX 4.22, Hercules.

3. Nội dung

3.0. Làm quen với Kit STM32F429

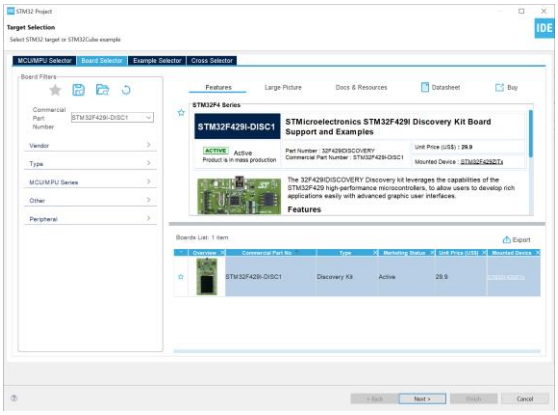
Các nhóm nhận và làm quen với kit STM32F429

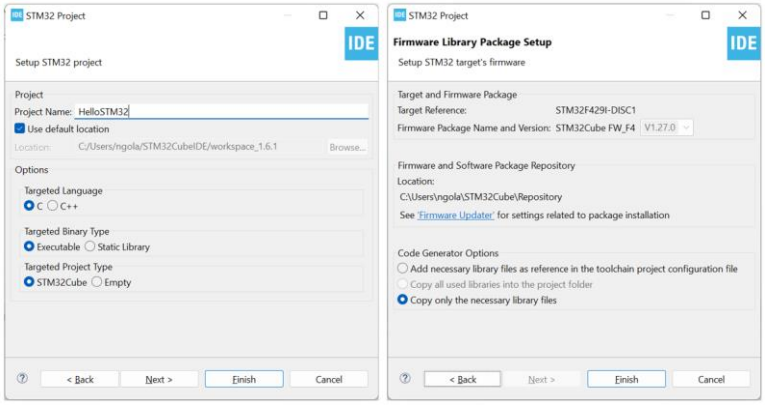
- STM32F429ZIT6 ARM Cortex-M4.
- Tốc độ CPU 180 MHz (max).
- 2 MB flash, 256 KB SRAM, 8MB SDRAM.
- Màn hình LCD 2.4” 240x320, cảm ứng điện trở.
- 6 LEDs (2 user LEDs).
- 3 axis gyros.

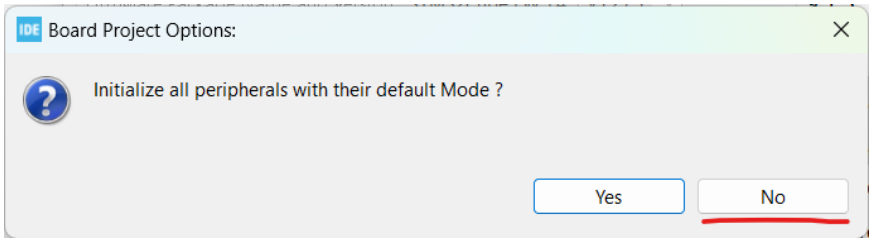
3.1 Làm quen với STM32CubeIDE

- Mở STM32CubeIDE, chọn New → STM32 Project → Board Selector. Nhập tên board STM32F429I-DISC1.

- Đặt tên project và thiết lập sử dụng thư viện firmware FW_F4 1.27. Nếu chưa có thư viện này thì STM32CubeIDE có thể tự tải về.







- Chọn No khi được hỏi khởi tạo ngoại vi ở chế độ mặc định. Project sẽ đơn giản hơn.

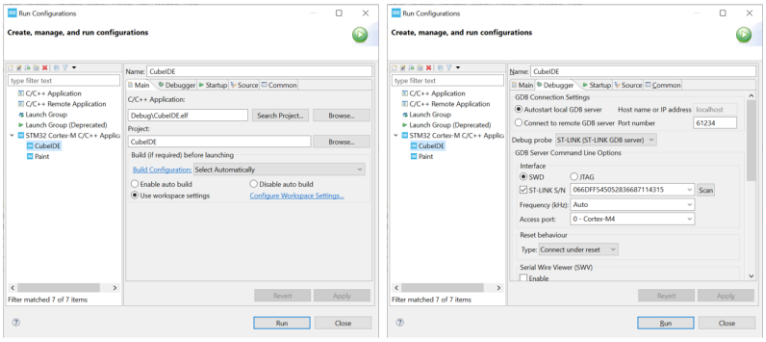
- Mở file main.c trong Project Explorer, thêm 2 dòng code điều khiển chân LD3 và LD4 xuống dưới dòng `MX_GPIO_Init();` trong hàm main().

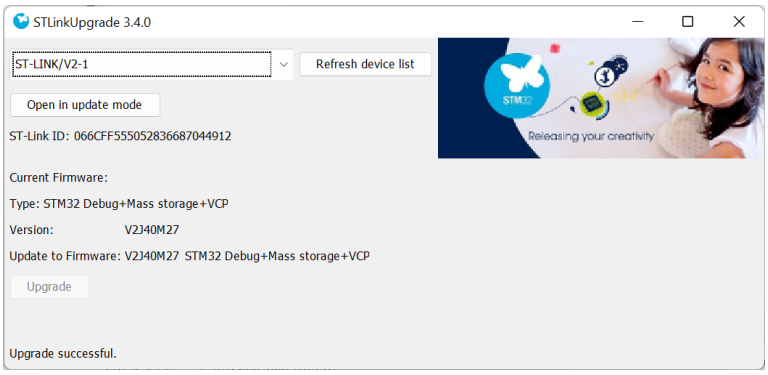
```
HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_SET);  //turn on LED3
HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);  //turn on LED4
```

- Chọn Run → Run as STM32 CortexM C/C++ Application.

- Cắm mạch vào cổng USB của máy tính, chọn Debugger là SWD, ST-LINK, bấm Scan để tìm S/N, chọn S/N tương ứng.

Chú ý: có thể sẽ cần update firmware của mạch nạp ST-LINK trên kit STM32F429. Khi đó làm theo hướng dẫn, rút cable khỏi kit rồi cắm lại trước khi nạp lại chương trình.





- Quan sát 2 đèn LED màu xanh và cam cùng bật sáng.

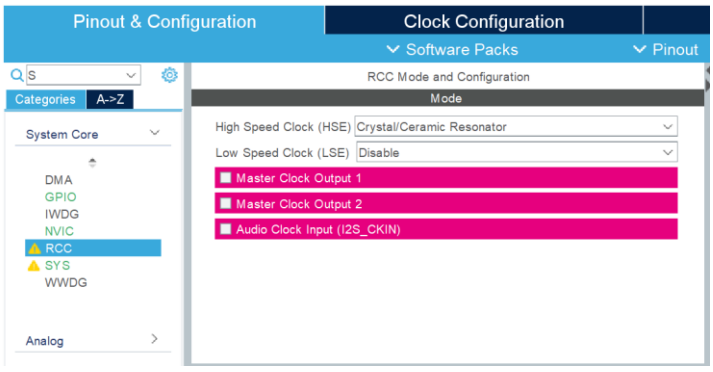
- Tìm hiểu hàm main(), các công việc được thực hiện tại hàm main(), ý nghĩa của 2 dòng code được thêm vào ở trên.

3.2 Giao tiếp với ngoại vi

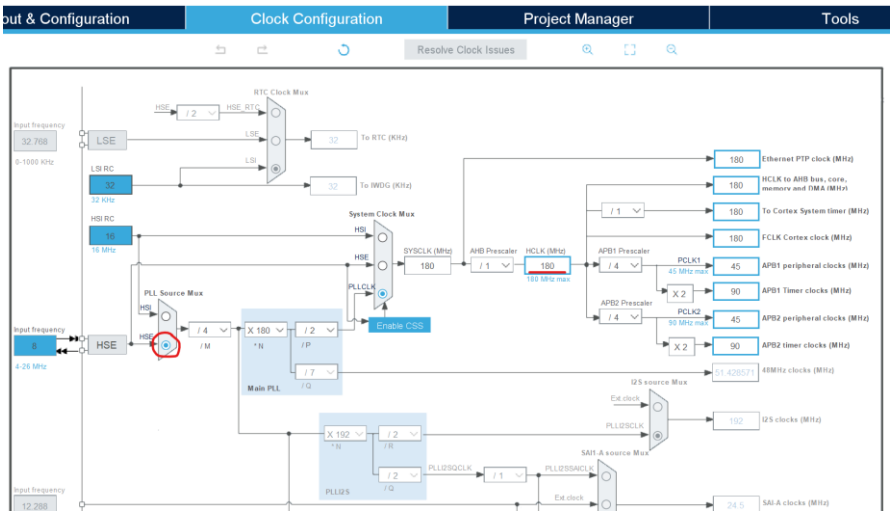
- Mở file **ioc** trong project.

- Cấu hình clock cho CPU

    - Chọn RCC, đặt HSE là Crystal/Ceramic Resonator

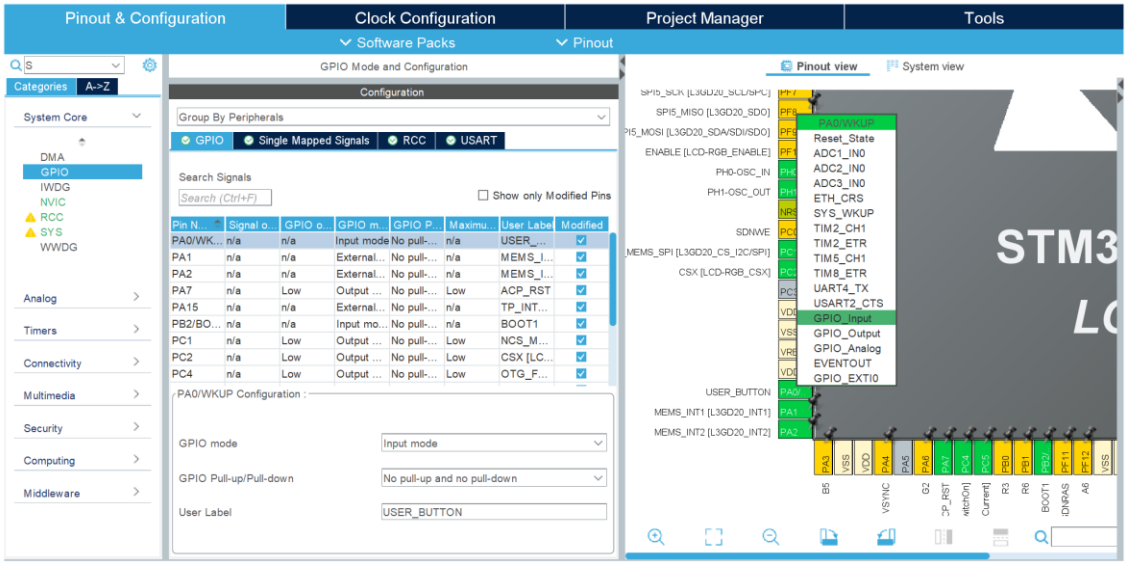


- Mở tab Clock Configuration, chọn HSE, đặt tốc độ clock là 180 MHz (bấm Enter).



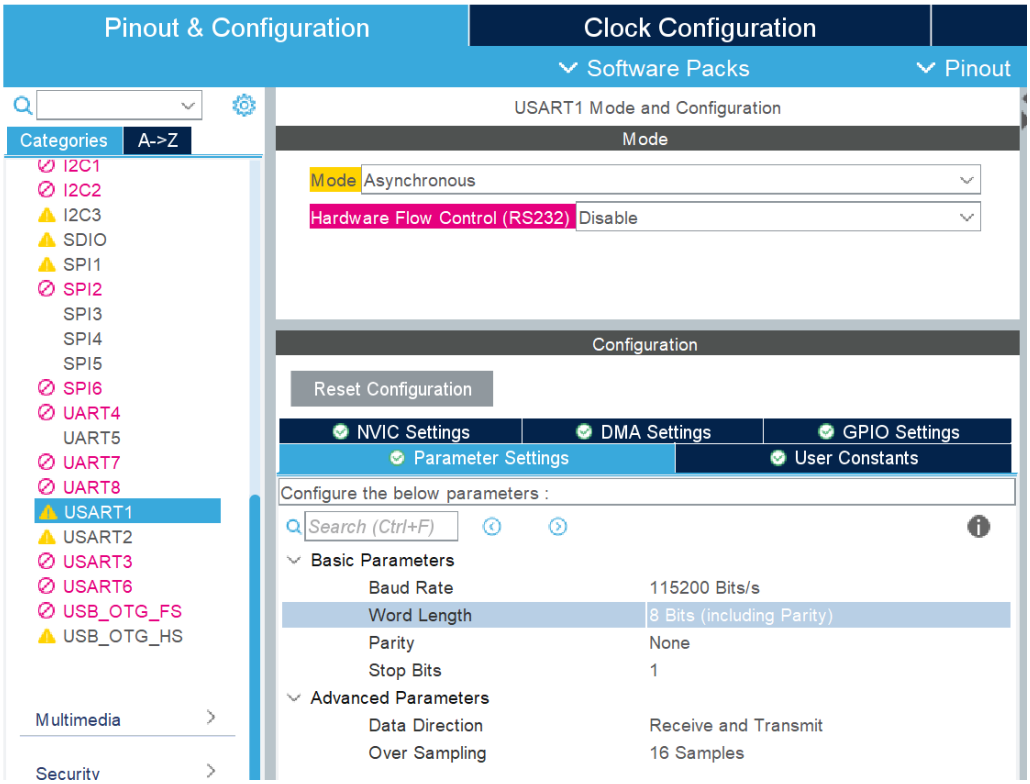
- Cấu hình chân PA0 ở chế độ input để ghép nối nút bấm:

  - Quay lại Pinout & Configuration, chọn GPIO.
  - Bấm chọn chân PA0 rồi đặt cấu hình là GPIO_Input.
  - Chọn chế độ trong phần GPIO setting: Input mode, No pull-up and no pull-down.
  - Đặt tên cho tín hiệu là USER_BUTTON.



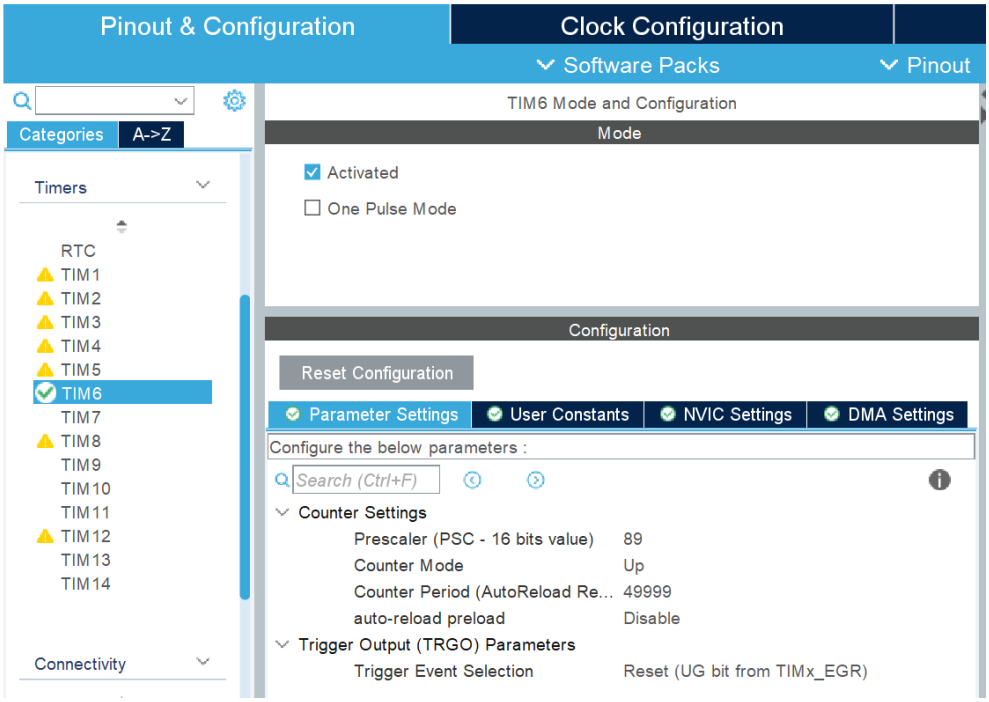
- Cấu hình cho cổng USART1 (được nối với máy tính qua virtual COM port của mạch ST-LINK):

  - Vào Connectivity chọn USART1.
  - Đặt cấu hình: Mode Asynchronous, Baudrate 115200 bps, Word length 8 bits, Parity None.



- Cấu hình cho Timer 6 với chu kỳ 25 ms, có sử dụng ngắt.

- Chọn Timers → TIM6.
- Đặt cấu hình: Mode Activated, Prescaler = 89, Counter period = 49999.
- Enable TIM6 global interrupt trong NVIC Settings.



- Lập trình file main.c: ghép nối nút bấm, UART.

Thêm lệnh bắt đầu chạy Timer trong hàm main()

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM6_Init();
MX_USART1_UART_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim6);
HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_SET);  //turn on LED3
HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);  //turn on LED4
/* USER CODE END 2 */
```

Polling trạng thái nút bấm và định kỳ gửi dữ liệu qua cổng COM trong main loop

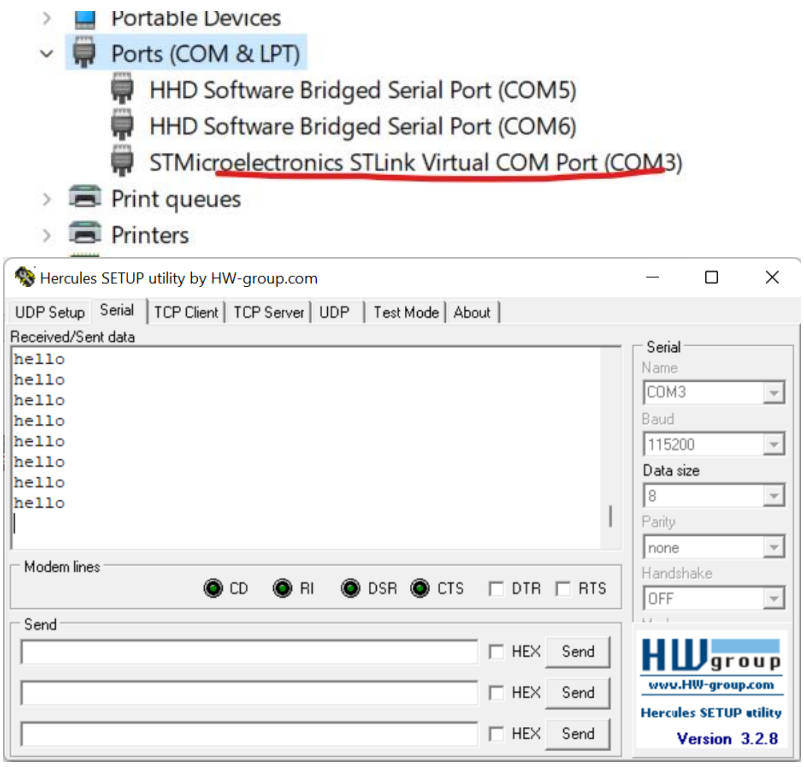
```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    GPIO_PinState UserButtonState = HAL_GPIO_ReadPin(USER_BUTTON_GPIO_Port, USER_BUTTON_Pin);
    if (UserButtonState == GPIO_PIN_SET)
    {
        HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_SET);  //turn on LED4
    }
    else
    {
        HAL_GPIO_WritePin(LD4_GPIO_Port, LD4_Pin, GPIO_PIN_RESET);//turn off LED4
    }
    char buffer [10] = "hello\r\n";
    HAL_Delay(500);
    HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 2);
}
```

- Lập trình file stm32f4xx_it.c, xử lý ngắt của timer với chu kỳ 25 ms. Đèn LED3 (màu xanh) sẽ nháy với tần số 1 Hz.

```
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */
  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */
    TimerCount++;
    if (TimerCount >= 20)
    {
        TimerCount = 0;
        HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin); //toggle LED3
    }
  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

- Biên dịch chương trình, nạp code và quan sát hoạt động của mạch.

  - Đèn LED xanh nháy với tần số 1 Hz.
  - Khi bấm và giữ nút bấm màu xanh thì đèn LED cam sáng, nhả nút thì đèn lại tắt.
  - Mở ứng dụng Hercules, mở cổng Serial ứng với cổng COM ảo của mạch ST-LINK, baudrate 115200 bps, sẽ thấy dòng chữ hello được gửi liên tục với chu kỳ 500 ms.



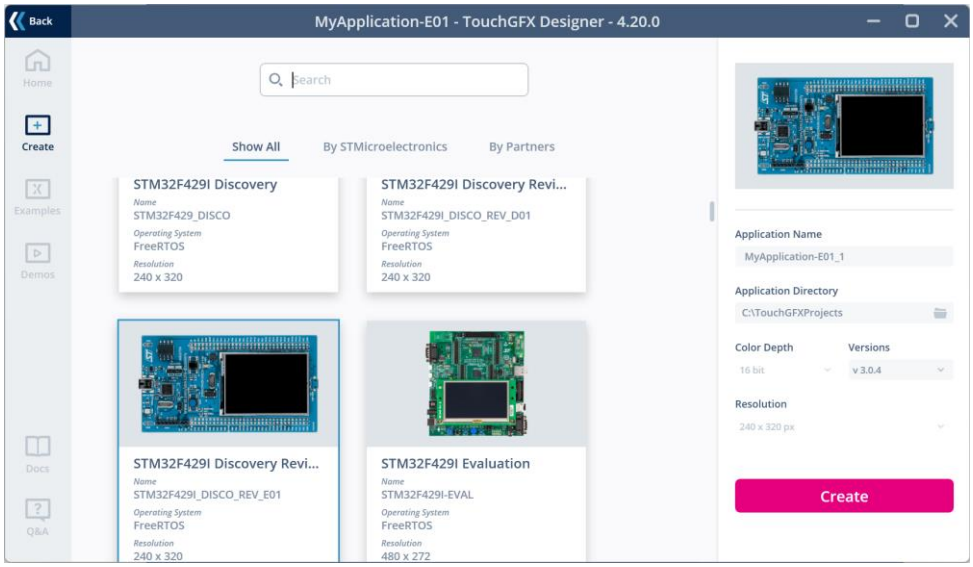
- Hãy giải thích:

  - Vai trò của các đoạn code thêm vào bên trên.
  - Vai trò của đoạn code dưới đây trong hàm MX_GPIO_Init()

```
/*Configure GPIO pin : USER_BUTTON_Pin */
GPIO_InitStruct.Pin = USER_BUTTON_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(USER_BUTTON_GPIO_Port, &GPIO_InitStruct);
```

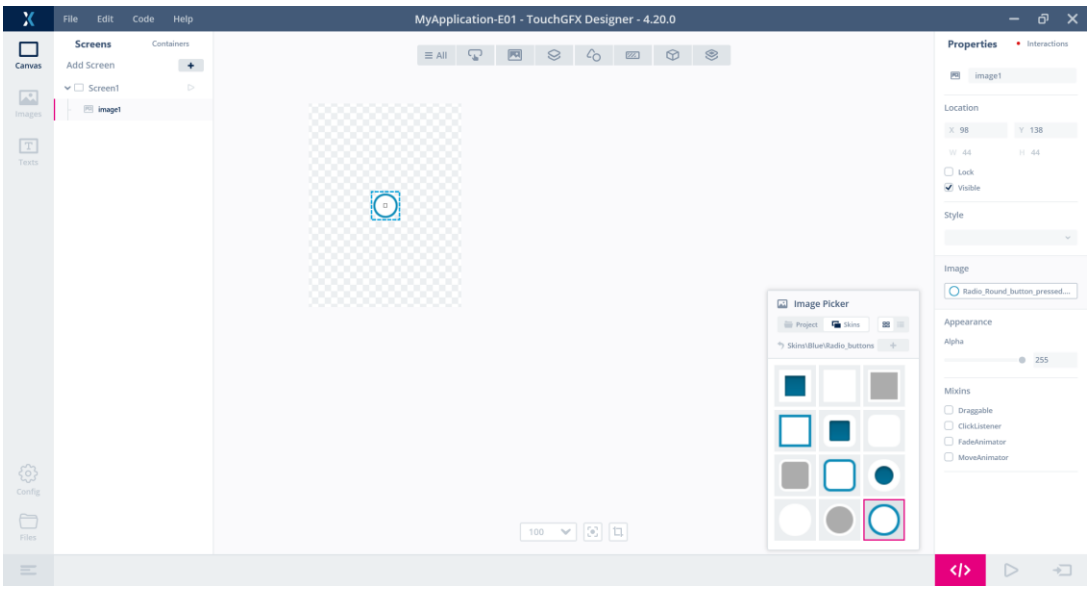
3.3 Xây dựng ứng dụng với TouchGFX và FreeRTOS

- Mở TouchGFX 4.22, tạo project mới với board STM32F429 Revision E01 (xanh dương). Đặt tên project là MyApplication-E01 (hoặc tùy chọn, nhưng tên MyApplication-E01 sẽ được sử dụng để tham chiếu trong tài liệu này).

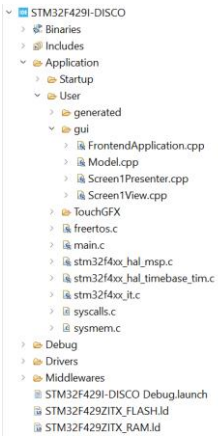


- Trong phần Canvas để thiết kế giao diện, đặt 1 đối tượng Image lên Screen1. Chọn ảnh cho đối tượng trong mục Image.

- Bấm nút Generate code (F4) màu đỏ góc dưới phải.



- Double click file **.project** trong thư mục **MyApplication-E01\STM32CubeIDE** để import project vào STM32CubeIDE.



- Dịch và nạp chương trình lên kit STM32F429. Kết quả thu được là hình tròn viền xanh hiện giữa màn hình.

- Tìm hiểu cấu trúc project.

- Lập trình để cho hình tròn chạy liên tục:

Khai báo thêm thành viên cho Screen1View

Screen1View.cpp 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

```

Xử lý sự kiện tick() để cập nhật vị trí hình tròn

Screen1View.cpp Screen1View.hpp

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

Screen1View::Screen1View()
{
    tickCount = 0;
}

void Screen1View::setupScreen()
{
    Screen1ViewBase::setupScreen();
}

void Screen1View::tearDownScreen()
{
    Screen1ViewBase::tearDownScreen();
}

void Screen1View::handleTickEvent()
{
    Screen1ViewBase::handleTickEvent();     // Call superclass eventhandler
    tickCount += 1;
    image1.setY(tickCount % 320);
    invalidate();
}

```

Dịch và nạp chương trình lên kit, quan sát chuyển động theo chiều dọc của hình tròn.

- Lập trình ghép nối nút bấm USER_BUTTON để di chuyển hình tròn theo chiều ngang.

Cấu hình chân vào ra trong hàm MX_GPIO_Init() ở main.c

```
/*Configure GPIO pin : USER_BUTTON is A0 pin */
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Khai báo queue trong phần Private Variable file main.c (sau khai báo tạo thread ID đã có)

```
/* USER CODE BEGIN PV */
osMessageQueueId_t queue1;
osMessageQueueId_t Queue1Handle;
const osMessageQueueAttr_t Queue1_attributes = {
  .name = "Queue1"
};
osStatus_t r_state;
/* USER CODE END PV */

```

Tạo queue trước khi tạo task trong file main.c

```
  /* USER CODE BEGIN RTOS_QUEUES */
  /* creation of Queue1 */
  Queue1Handle = osMessageQueueNew (8, sizeof(uint8_t), &Queue1_attributes);
  /* USER CODE END RTOS_QUEUES */
```

Liên tục polling trạng thái nút bấm trong default task, gửi dữ liệu vào queue

```
/* 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)
    {
        uint32_t count = osMessageQueueGetCount(Queue1Handle);
        if (count < 2)
        {
            uint8_t x = 'R';
            osMessageQueuePut(Queue1Handle, &x, 0, 200);
        }
    }
    osDelay(10);
  }
  /* USER CODE END 5 */
}
```

Hiệu chỉnh hàm **handleTickEvent** của Screen1View để nhận dữ liệu từ queue và cập nhật animation. Chú ý: #include <cmsis_os.h>

```
extern osMessageQueueId_t Queue1Handle;
void Screen1View::handleTickEvent()
{
    Screen1ViewBase::handleTickEvent();     // Call superclass eventhandler
    tickCount += 1;
    image1.setY(tickCount % 320);

    uint8_t res = 0;
    uint32_t count = osMessageQueueGetCount(Queue1Handle);
    if (count > 0)
    {
        //note: the below function may block this thread
        osMessageQueueGet(Queue1Handle, &res, NULL, osWaitForever);
        if (res == 'R')
        {
            int16_t x = image1.getX();
            x += 2;
            if (x > 240)
            {
                x = 0;
            }
            image1.setX(x);
        }
    }

    invalidate();
}
```

Dịch và chạy chương trình trên kit. Quan sát hoạt động của mạch:

- Hình tròn rơi từ trên xuống.
- Khi bấm nút thì hình tròn đồng thời di chuyển sang phải cùng với chuyển động rơi xuống.

Giải thích hoạt động của chương trình.

3.4 Bài tập tự làm

Sử dụng mã nguồn mẫu đã cho “MyApplication-D01”, xây dựng giao diện và phần hoạt hình cơ bản cho 1 game đua xe.

- Đường đua di chuyển theo chiều từ trên xuống.
- Xe chỉ có thể di chuyển theo chiều ngang.
- Ghép nối thêm 2 nút bấm (sử dụng chân PG2 và PG3), để điều khiển xe dịch chuyển sang trái/phải. Chú ý: cần tham khảo hàm MX_GPIO_Init() trong bài 3.2 để tự viết code cấu hình PG2 và PG3 ở chế độ.

Các file asset mẫu được để trong cùng thư mục.

- Hết -