Bài 21 :Low Power Mode trên STM32F4

Kiến thức cần chuẩn bị

1.Power Suplies

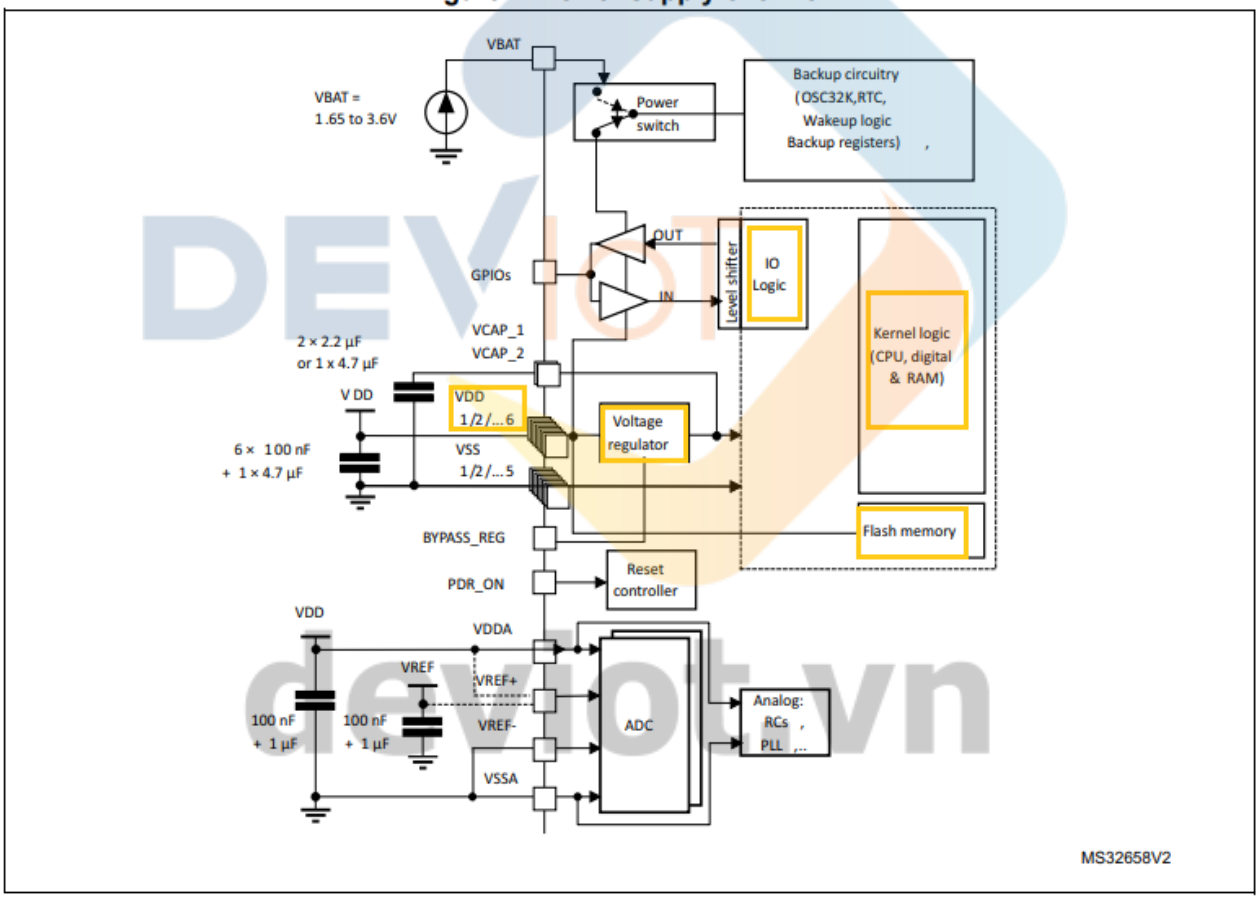
Có hai miền cấp điện áp chính:

• VDD = 1.7 đến 3.6 V: nguồn điện bên ngoài cho các chân I/O với bộ điều chỉnh bên trong bị tắt , được cung cấp bên ngoài thông qua các chân VDD. Yêu cầu sử dụng nguồn điện bên ngoài được kết nối với các chân VDD và PDR_ON (Rese0t).

• 1,8 đến 3,6 V: nguồn điện cấp cho Core Memory và digital peripherals và bộ điều chỉnh bên trong (khi được bật), được cung cấp bên ngoài thông qua các chân VDD.

Thông thường khối thời gian thực và các thanh ghi backup có thể được cấp nguồn VDD nhưng nếu khi bộ nguồn VDD bị mất đi thì lúc này Chip sẽ tự động chuyển mạch sang VBAT để cấp cho khối này

Figure 7. Power supply overview



Tham khảo tại trang của ST
https://www.st.com/content/ccc/resource/technical/document/reference_manual/9b/53/39/1c/f7/01/4a/79/DM00119316.pdf/files/DM00119316.pdf/jcr:content/translations/en.DM00119316.pdf

Voltage Regulator (Bộ hiệu chỉnh điện áp)

Bộ hiệu chỉnh điện áp luôn được bật sau khi Reset. Mục đích nó dùng để cung cấp nguồn ở các chế độ hoạt động của Chip, cụ thể nó bộ hiệu chỉnh này có 3 chế độ hoạt động khác nhau tùy thuộc vào ứng dụng của Chip:

- Run mode: Bộ hiệu chỉnh này sẽ cung cấp full power 1.2V (core, memories and digital peripherals)
- Stop mode: Bộ hiệu chỉnh sẽ cung cấp low power 1.2V để bảo quản nội dung của các thanh ghi và SRAM
- Standby mode: Bộ hiệu chỉnh này sẽ hoàn toàn bị tắt, nội dung trong các thanh ghi và SRAM sẽ bị mất đi. Nếu trước lúc vào chế độ Standby mode bạn có ghi các dữ liệu vào Backup domain thì khi vào chế độ Standby mode, hệ thống sẽ tự chuyển mạch sáng VBAT để lưu trữ dữ liệu của bạn.

  https://www.st.com/content/ccc/resource/technical/document/reference_manual/9b/53/39/1c/f7/01/4a/79/DM00119316.pdf/files/DM00119316.pdf/jcr:content/translations/en.DM00119316.pdf
  (Tham khảo tại mục 5.13 trong link)

3. Low-power modes

Mặc định, Chip ở Run mode sau khi thiết lập lại hệ thống hoặc bật nguồn. Khi đó CPU được xung nhịp bởi HCLK và chương trình code của ta được thực thi. Một số Low Power ở chế độ có sẵn để tiết kiệm năng lượng khi CPU không cần phải tiếp tục chạy (ví dụ khi chờ một sự kiện bên ngoài). Tùy thuộc vào người dùng chọn chế độ cung cấp tốt nhất giữa mức tiêu thụ điện năng thấp, thời gian khởi động ngắn và nguồn có sẵn . Các thiết bị có ba chế độ Low-Power:

• Sleep (Cortex®-M4 với Core FPU dừng, các thiết bị ngoại vi tiếp tục chạy)

• Stop (tất cả clocks đều dừng)

• Standby (tắt nguồn 1,2 V, Các PLL, HSI RC và HSE cũng có thể bị tắt.)

Source :Datasheet mục Low Power(https://www.st.com/resource/en/datasheet/stm32f411ce.pdf)

Với bài viết này ta sẽ làm với Sleep mode

4. Sleep mode

Entering sleep mode

Để có thể đưa Chip vào chế độ sleep mode thì chúng ta cần thi hành câu lệnh WFI(wait for interrupt) hoặc WFE(wait for event). Có sẵn hai sự lựa chọn để chúng ta chọn cơ chế vào chế độ Sleep mode, việc chọn option này hay option kia phụ thuộc vào bit SLEEPONEXIT trong thanh ghi System Register Control.

Sleep-now: Nếu bit SLEEPONEXIT bị clear, thì Chip sẽ vào mode Sleep mode ngay khi câu lệnh WFI hoặc WFE được thi hành.

Sleep-on-exit: Nếu bit SLEEPONEXIT được set, Chip sẽ chuyển sang mode Sleep mode ngay khi nó thoát khỏi ISR(Interrupt service routine) ưu tiên thấp nhất.

Trong Sleep mode, tất cả các chân I/O đều giữ trạng thái như trong Run mode và để tiết kiệm năng lượng Flash memory có thể tắt.

Table 15. Sleep-now entry and exit

Sleep-now mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0, and – No interrupt (for WFI) or event (for WFE) is pending. Refer to the Cortex®-M4 with FPU System Control register.
	On Return from ISR while: – SLEEPDEEP = 0 and – SLEEPONEXIT = 1, – No interrupt is pending. Refer to the Cortex®-M4 with FPU System Control register.
Mode exit	If WFI or Return from ISR was used for entry: Interrupt: Refer to *Table 37: Vector table for STM32F411xC/E* If WFE was used for entry and SEVONPEND = 0 Wakeup event: Refer to *Section 10.2.3: Wakeup event management* f WFE was used for entry and SEVONPEND = 1 Interrupt even when disabled in NVIC: refer to *Table 37: Vector table for STM32F411xC/E* or Wakeup event (see *Section 10.2.3: Wakeup event management*).
Wakeup latency	None

Nếu câu lệnh WFI được sử dụng để vào chế độ Sleep mode, bất cứ xác nhận ngắt ngoại vi bởi các bộ điều khiển vector ngắt (NVIC) đều có thể đánh thức Chip dậy từ Sleep mode. Giả sử rằng nếu hệ thống của bạn đang chạy đợi nhận dữ liệu từ ngắt nhận UART, trong quá trình chưa nhận dữ liệu thì chúng ta sẽ cho vào chế độ Sleep mode để tiết kiệm năng lượng cho hệ thống. Khi dữ liệu đến tại chân RX của UART thì sẽ sinh ra ngắt UART từ đó đánh thức Chip dậy hoạt động trở lại bình thường.

Nếu câu lệnh WFE được sử dụng để vào chế độ Sleep mode, thì Chip thoát khỏi Sleep mode ngay khi có 1 Event xảy ra. Event đánh thức có thể được tạo bởi:

- Bật một bit ngắt trong thanh ghi điều khiển ngoại vi, bật bit SEVONPEND trong thanh ghi System Control register. Khi mà MCU trở lại từ WFE, bit cờ ngắt ngắt ngoại vi hoặc bit NVIC IRQ channel phải bị xóa.
- Cấu hình một chân một sự kiện dòng ngắt ngoài hoặc ngắt nội như là timer. Khi mà khi mà MCU trở lại từ WFE, thì không cần phải xóa cờ ngắt hoặc bit chờ xử lý NVIC IRQ channel như là các bit tương ứng khi bit đang chờ xử lý tương ứng với dòng event chưa được set

Chế độ này cung cấp thời gian đánh thức thấp nhất vì không bị lãng phí thời gian trong việc entry/exit

Table 16. Sleep-on-exit entry and exit

Sleep-on-exit	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP = 0, and – No interrupt (for WFI) or event (for WFE) is pending. Refer to the Cortex®-M4 with FPU System Control register.
	On Return from ISR while: – SLEEPDEEP = 0, and – SLEEPONEXIT = 1, and – No interrupt is pending. Refer to the Cortex®-M4 with FPU System Control register.

Table 16. Sleep-on-exit entry and exit (continued)

Sleep-on-exit	Description
Mode exit	Interrupt: refer to *Table 37: Vector table for STM32F411xC/E*
Wakeup latency	None

Để vào mode Sleep mode thì ban đầu set bit SLEEPONEXIT, chọn vào mode sleep mode thông qua WFI hay WFE

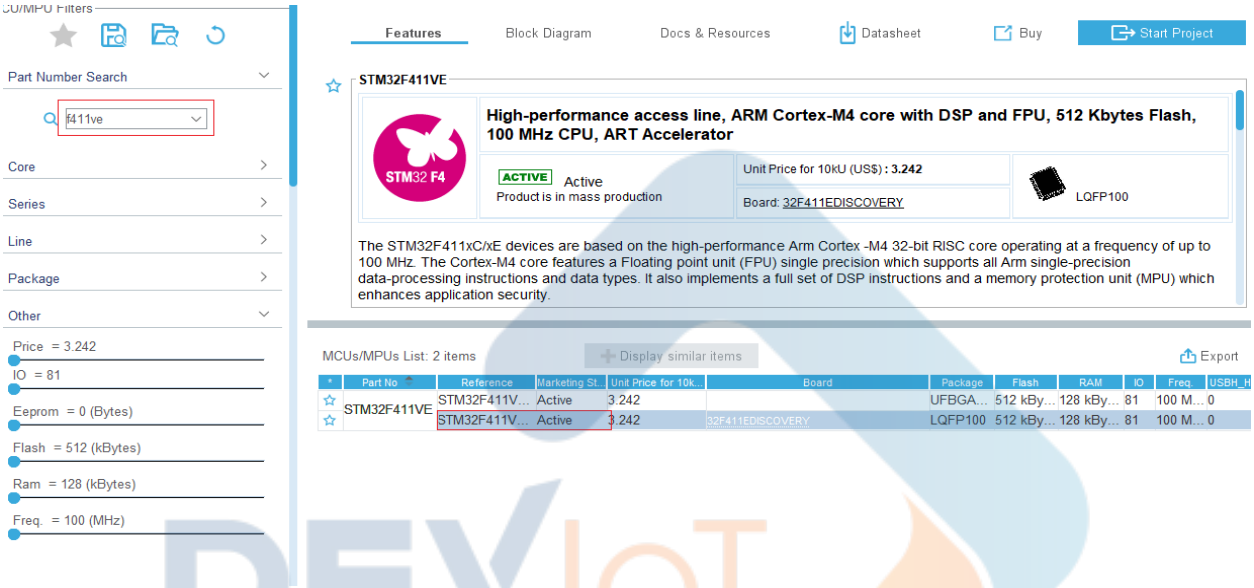
Để thoát khỏi Sleep mode thì nếu WFI được sử dụng thì sẽ đợi cho Interrupt xảy ra, còn nếu WFE được sử dụng thì sẽ đợi cho 1 sự kiện đánh thức xảy ra

Trong ví dụ Sleepmode này chúng ta sẽ sử dụng chân PD12 để thực hiện việc Blink trong vòng lặp while(1) trong hàm main, chân PA0 dùng ngắt ngoài để đánh thức MCU dậy từ Sleepmode.

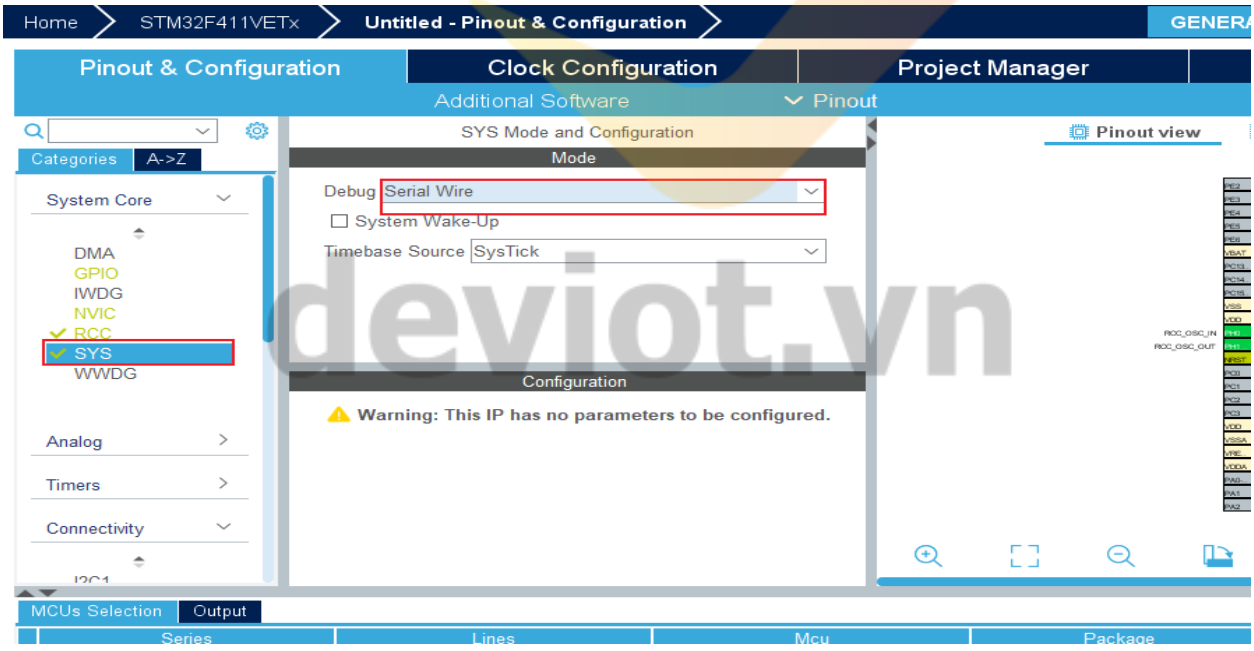
Lập trình

Mở phần mềm STM CubeMX, chọn dòng chip bạn sử dụng. Ở đây mình chọn chip STM32F411VE

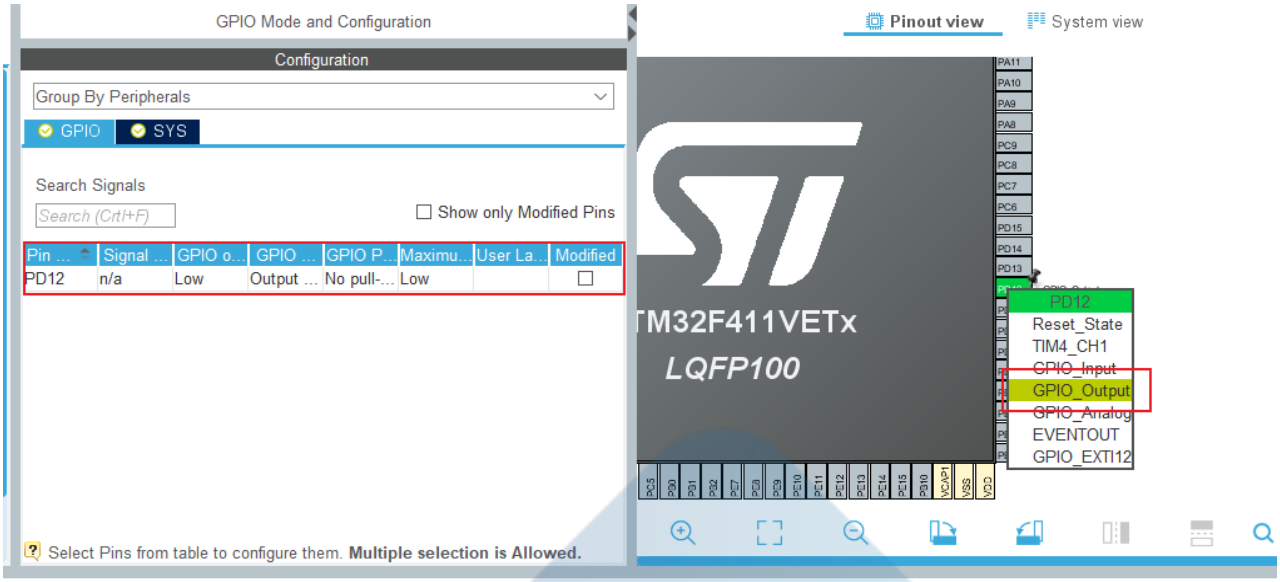
Đối với các dòng chip STM32 đời 4, tất cả mọi câu lệnh khi sử dụng thư viện HAL đều giống nhau. Chỉ khác nhau phần cấu hình Clock phụ thuộc riêng vào mỗi Chip



Chip Debug bằng SWD

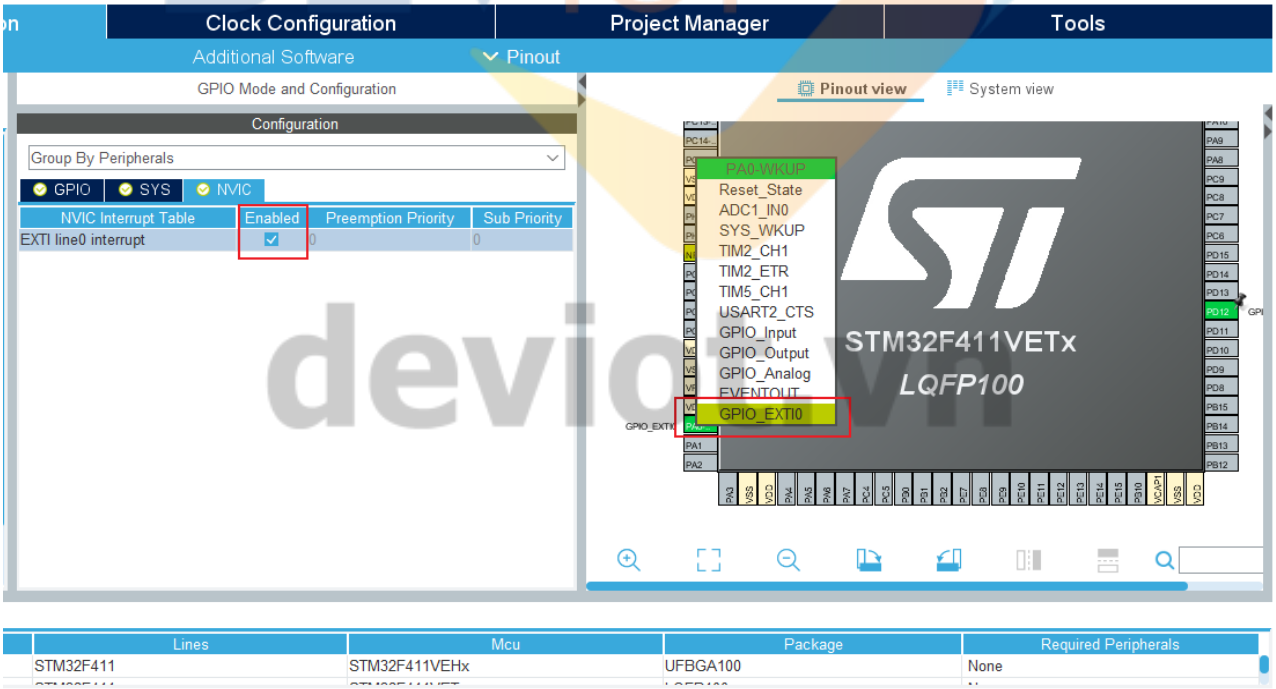


Cấu hình chân PD12 là GPIO Output

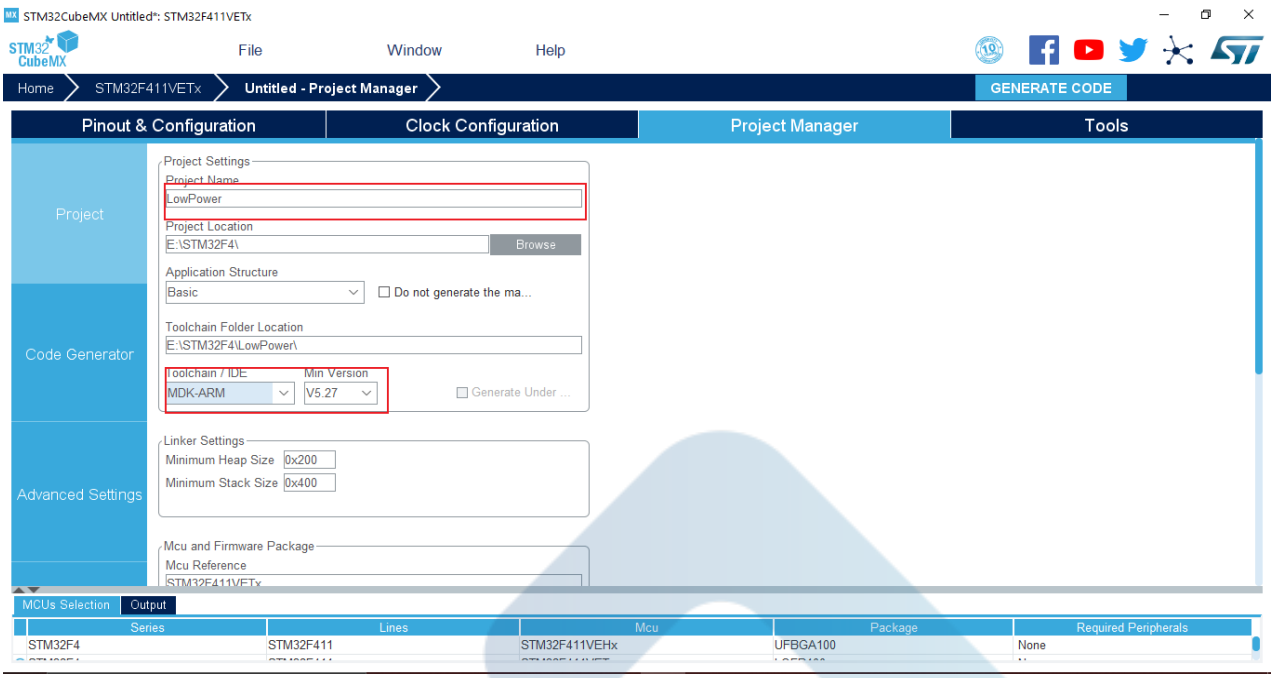


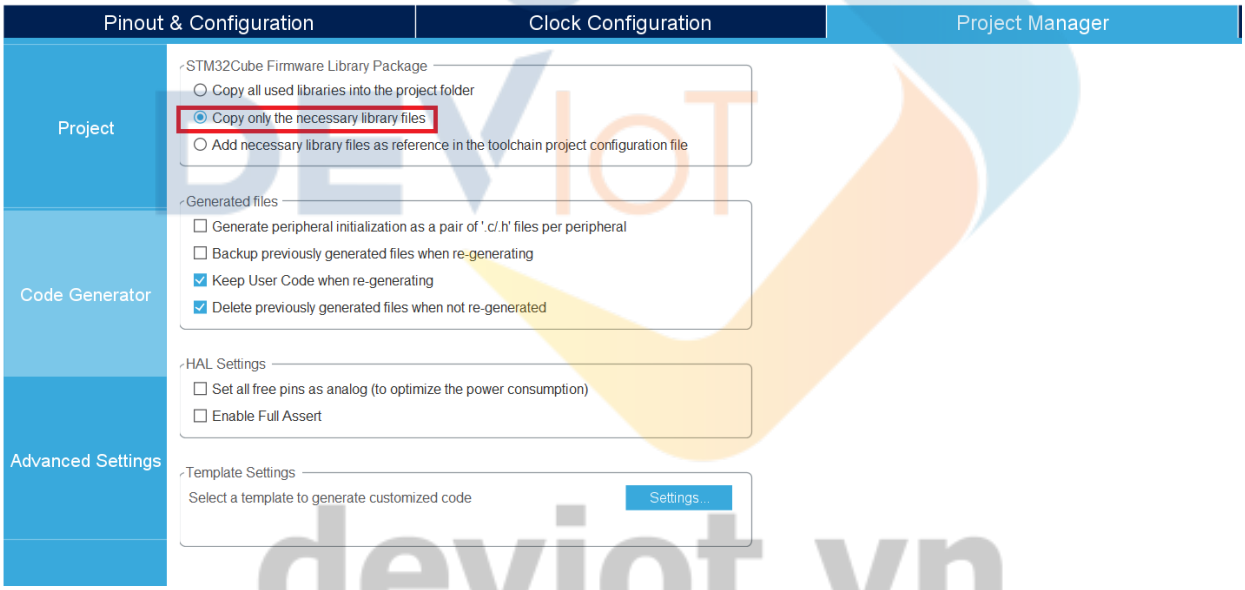
Click vào PA0-WKUP sau đó tại: GPIO Pull-up/Pull-down: Pull-up, ở đây tại chân PA0 sẽ được kéo lên nguồn Vcc = 3.3v

GPIO Mode chúng ta chọn External Interrupt Mode with Falling edge trigger detection, ở đây chân PA0 xảy ra sự kiện ngắt ngoài theo cạnh xuống khi nó được kéo xuống mức 0 thức là từ 3.3v->0v.



Click vào Project sau đó Setting , đặt tên cho project , chọn công cụ lập trình Kiel





Chọn những thư viện cần thiết để sinh code nhanh hơn và giảm dung lượng Project nhé.

Đầu tiên trong Kiel C ta kéo xuống thiết lập ngắt cho PA0 và PD12

```
/*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PD12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);
```

Sau đó tại int main() ta có đoạn code như sau và mình sẽ giải thích ngay

```
for(int i =0;i<10;i++)
        {
            HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
            HAL_Delay(500);
        }
  HAL_SuspendTick();
  HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
  HAL_ResumeTick();
  while (1)
  {
            HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12);
            HAL_Delay(200);
  }
```

Trước hàm while ta sẽ cho đèn thay đổi trạng thái 10 lần rồi mới vào chế độ Sleep mode sau đó chúng sẽ sử dụng hàm *HAL_SuspendTick()*;, mục đích sử dụng hàm này là gì?

Như lý thuyết ở trên đã nói, khi vào chế độ Sleepmode, thì Chip sẽ bị đánh thức dậy với bất cứ sự kiện ngắt nào được sinh ra trong quá trình Chip sleep. Khi sinh code từ CubeMX thì có một bộ Timebase source được tự động sinh ra gọi là SysTick, và sẽ sinh ra ngắt 1ms 1 lần, nếu chúng ta vẫn cứ cho SysTick chạy thì sẽ làm Chip không vào chế độ Sleepmode sau đó thoát ra rất nhanh vì vậy chúng ta phải sử dụng hàm treo SysTick (SysTick rất giống hàm Delay)

Tiếp theo chúng ta sẽ sử dụng hàm HAL_PWR_EnterSLEEPMode để vào chế độ Sleepmode.

Thông số đầu tiên là PWR_MAINREGULATOR_ON: Ở đây chúng ta vẫn bật bộ Voltage Regulator để sinh ra điện áp 1.2 cung cấp cho Core, Memory, Digital pheripherals

Thông số tiếp theo là PWR_SLEEPENTRY_WFI: Đợi cho sự kiện ngắt để Enter Sleepmode

Khi chạy qua hàm HAL_PWR_EnterSLEEPMode, chính thức Chip vào chế độ Sleepmode này, nếu muốn đánh thức Chip dậy thì bạn kéo chân PA0 xuống mức “0”. Lúc này chúng ta sử dụng hàm HAL_ResumeTick(); để tắt việc treo hệ thống.

Khi đã được đánh thức dậy, chương trình sẽ tiếp tục thực thi xuống vòng lặp while(1) và nhấp nháy led PD12