Fundamentals of Embedded and Real Time Systems

MODULE 03

TAMER AWAD

Review Module 02

Module 03

Blinking the LED

- Documentation
- Tracing our LED
- Group Demo
- Memory Map review
- Configuring the GPIO
- Blinking LED in C Code

C Programming (continued)

- Preprocessor
- Volatile
- Bit-wise operators in C

STM32MXCube

- Generating code for eval board
- Demo

Blinking the LED

- Documentation
- Tracing our LED
- Datasheets
- GPIO
- Delay

Documentation

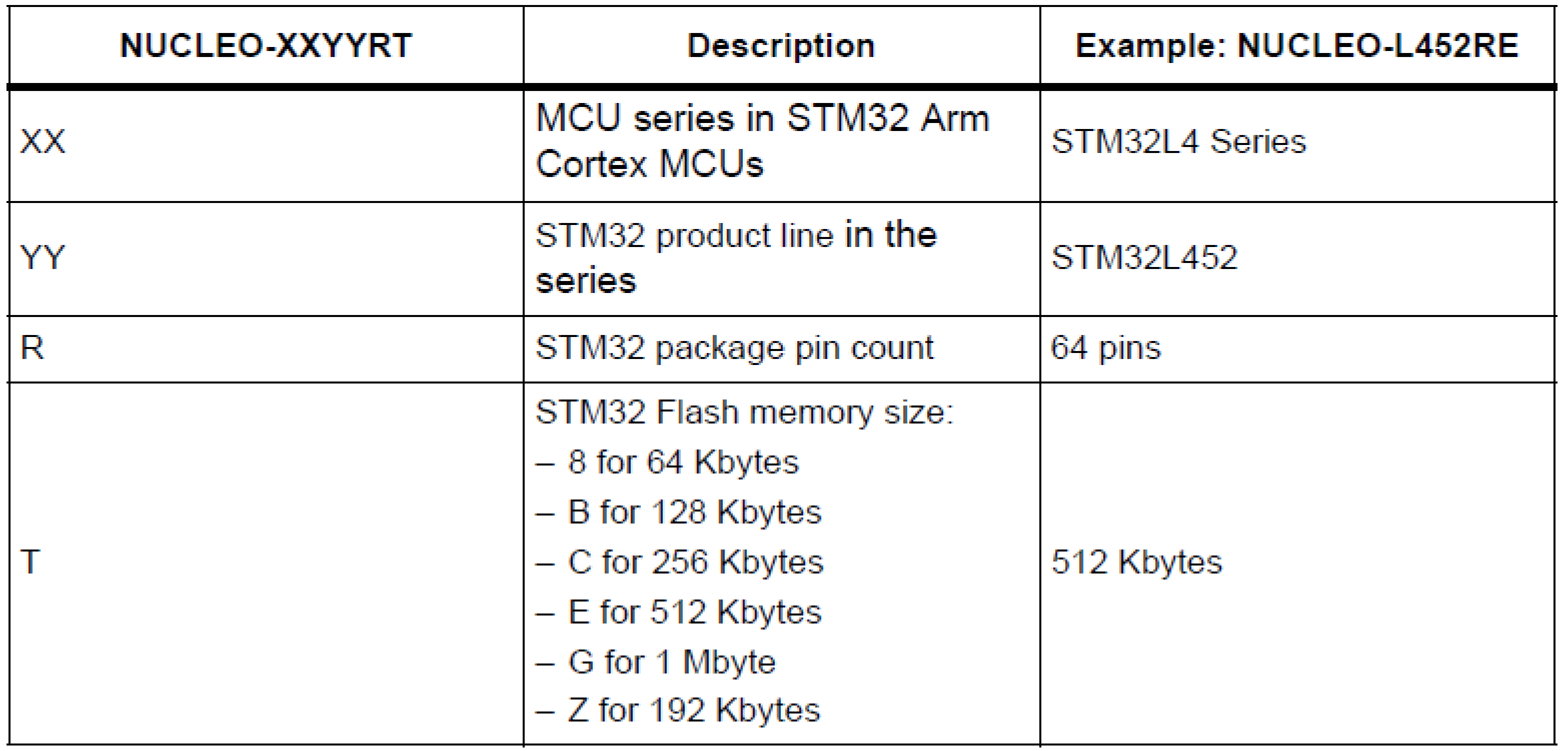
- Search for STM32F401 or browse thru STM website for STM32F01 resources.
- Find the user guide for our board "Nucleo-F401RE"
  - UM1724
- Find the datasheet for our board "Nucleo-F401RE"
  - STM32F401 Datasheet
- Find the reference manual for our STM32F401 Microcontroller (used on our board)
  - RM0368
- Cortex M4 Generic User Guide
  - DUI0553A

Tracing our user LED

DIGGING THRU “UM1724”,

USER MANUAL FOR OUR BOARD

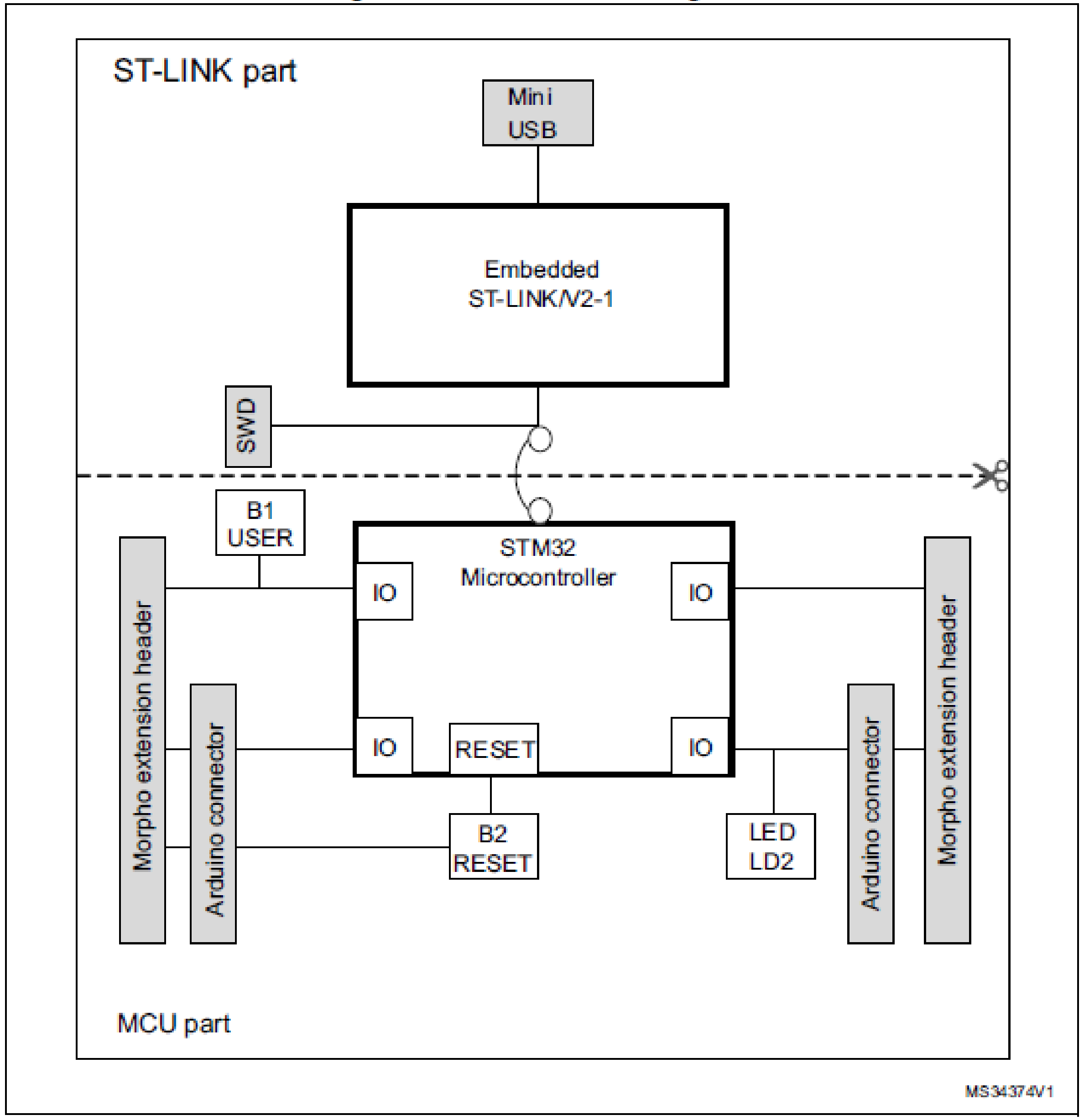
Codification: NUCLEO-F401RE



NUCLEO-XXYYRT	Description	Example: NUCLEO-L452RE
XX	MCU series in STM32 Arm Cortex MCUs	STM32L4 Series
YY	STM32 product line in the series	STM32L452
R	STM32 package pin count	64 pins
T	STM32 Flash memory size: – 8 for 64 Kbytes – B for 128 Kbytes – C for 256 Kbytes – E for 512 Kbytes – G for 1 Mbyte – Z for 192 Kbytes	512 Kbytes

Source: User Manual 1724

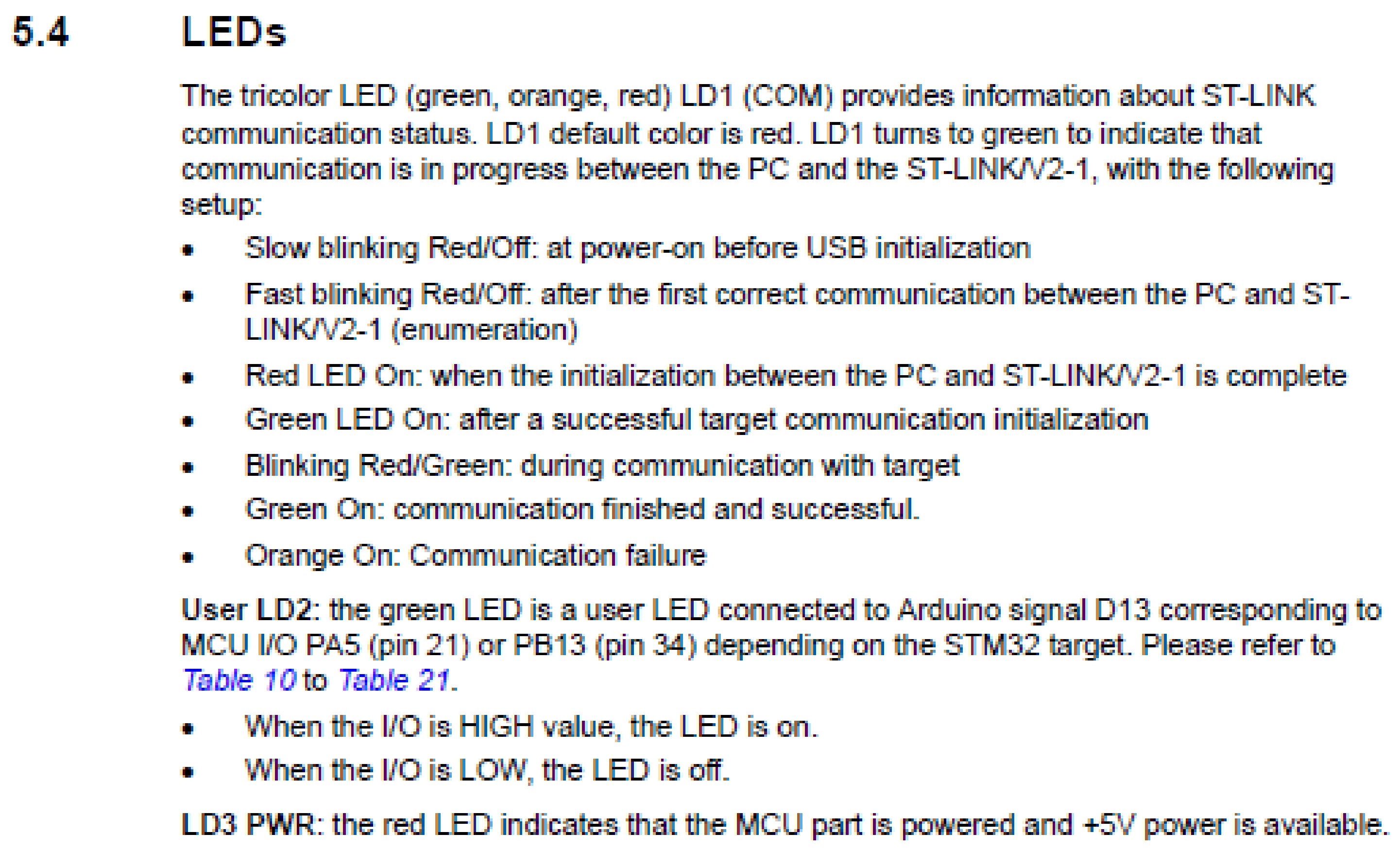
Figure 1. Hardware block diagram



Hardware Block Diagram

Source: User Manual 1724

Look for the LEDs (in UM1724)



5.4 LEDs

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

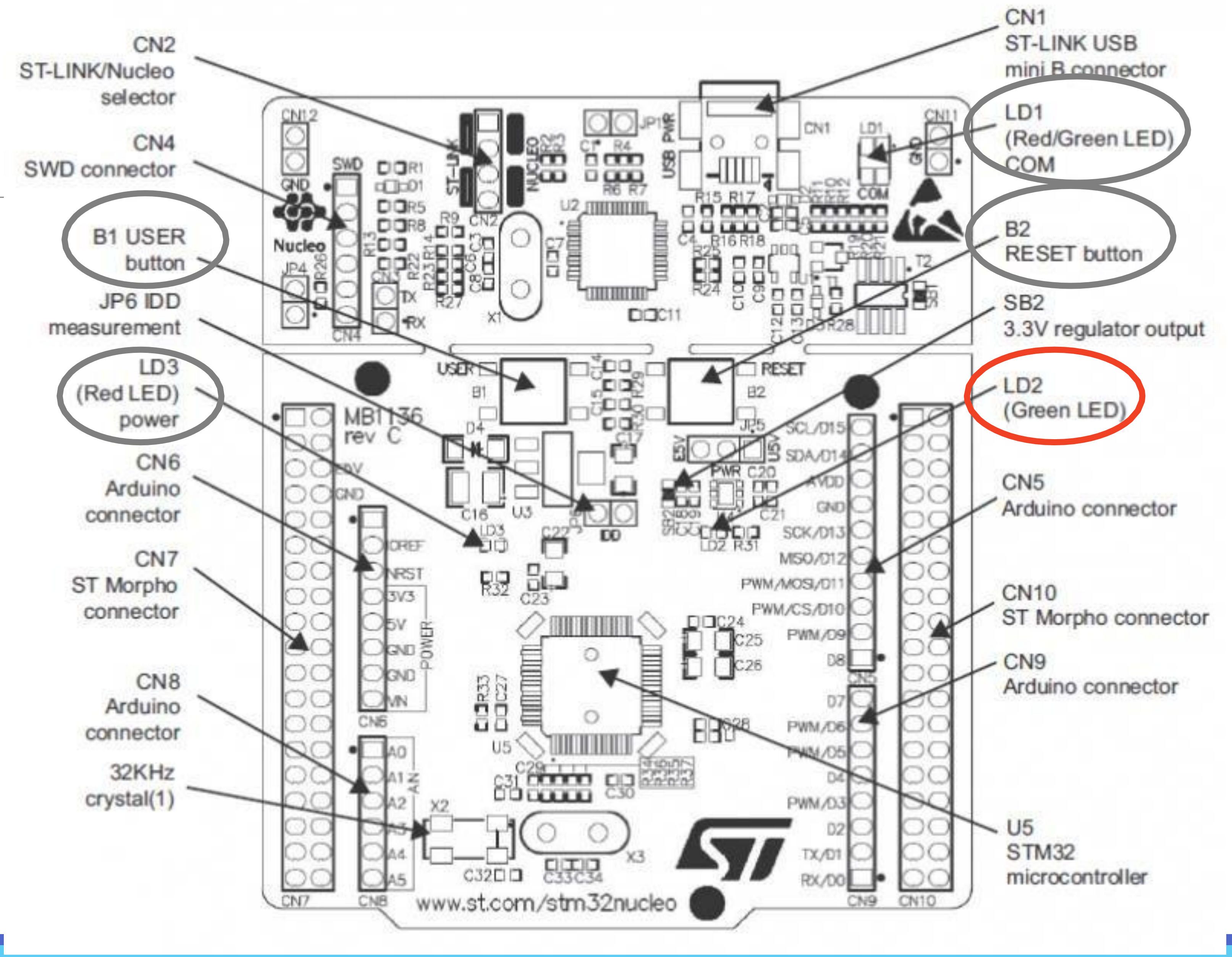
- Slow blinking Red/Off: at power-on before USB initialization
- Fast blinking Red/Off: after the first correct communication between the PC and ST-LINK/V2-1 (enumeration)
- Red LED On: when the initialization between the PC and ST-LINK/V2-1 is complete
- Green LED On: after a successful target communication initialization
- Blinking Red/Green: during communication with target
- Green On: communication finished and successful.
- Orange On: Communication failure

User LD2: the green LED is a user LED connected to Arduino signal D13 corresponding to MCU I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Please refer to *Table 10* to *Table 21*.

- When the I/O is HIGH value, the LED is on.
- When the I/O is LOW, the LED is off.

LD3 PWR: the red LED indicates that the MCU part is powered and +5V power is available.

Board Layout



Source: [User Manual 1724]

Arduino connectors on NUCLEO-F401RE

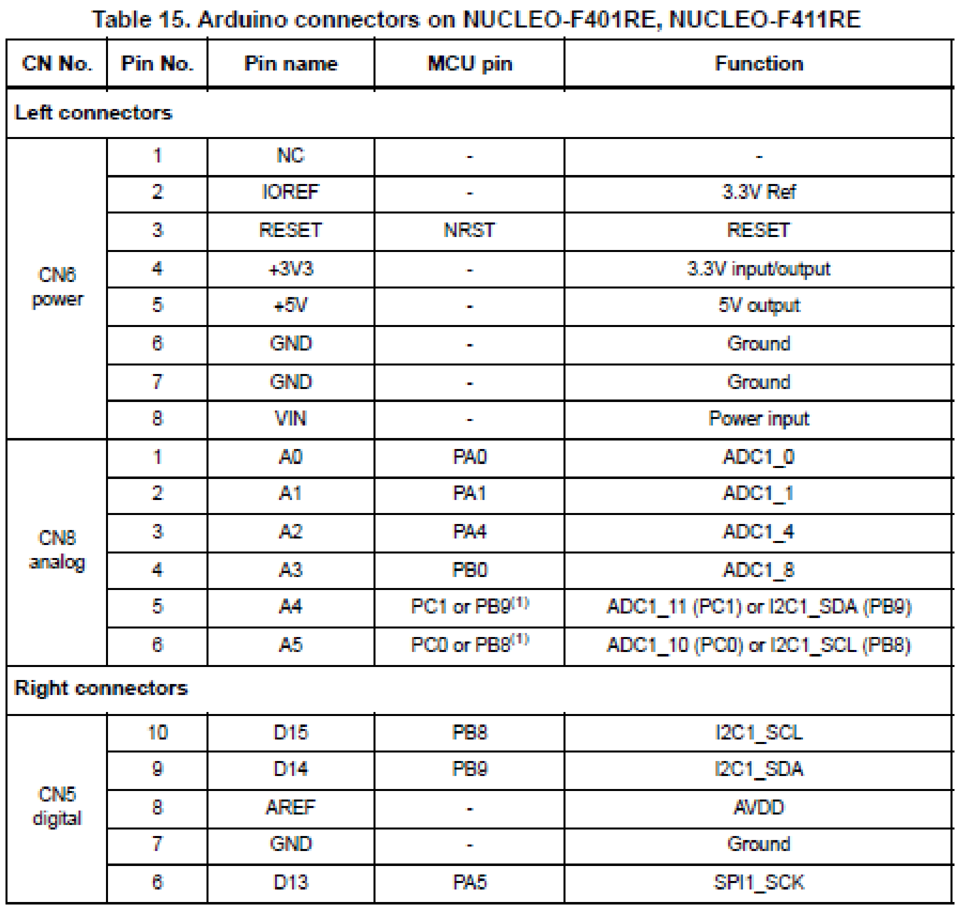


Table 15. Arduino connectors on NUCLEO-F401RE, NUCLEO-F411RE

CN No.	Pin No.	Pin name	MCU pin	Function
Left connectors				
CN6 power	1	NC	-	-
	2	IOREF	-	3.3V Ref
	3	RESET	NRST	RESET
	4	+3V3	-	3.3V input/output
	5	+5V	-	5V output
	6	GND	-	Ground
	7	GND	-	Ground
	8	VIN	-	Power input
CN8 analog	1	A0	PA0	ADC1_0
	2	A1	PA1	ADC1_1
	3	A2	PA4	ADC1_4
	4	A3	PB0	ADC1_8
	5	A4	PC1 or PB9(1)	ADC1_11 (PC1) or I2C1_SDA (PB9)
	6	A5	PC0 or PB8(1)	ADC1_10 (PC0) or I2C1_SCL (PB8)
Right connectors				
CN5 digital	10	D15	PB8	I2C1_SCL
	9	D14	PB9	I2C1_SDA
	8	AREF	-	AVDD
	7	GND	-	Ground
	6	D13	PA5	SPI1_SCK

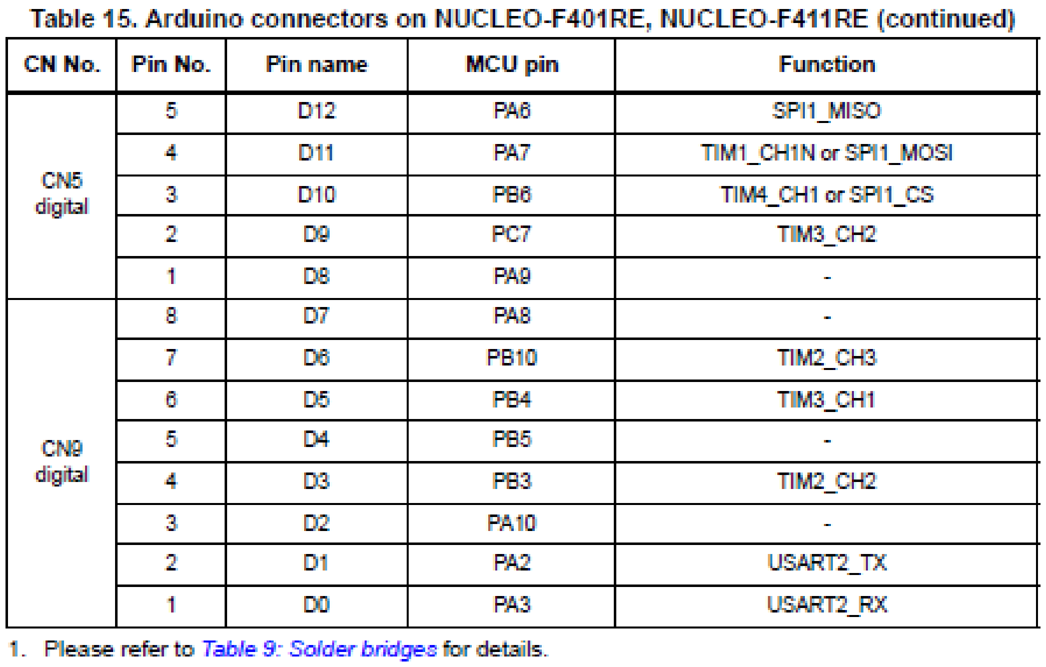


Table 15. Arduino connectors on NUCLEO-F401RE, NUCLEO-F411RE (continued)

CN No.	Pin No.	Pin name	MCU pin	Function
CN5 digital	5	D12	PA6	SPI1_MISO
	4	D11	PA7	TIM1_CH1N or SPI1_MOSI
	3	D10	PB6	TIM4_CH1 or SPI1_CS
	2	D9	PC7	TIM3_CH2
	1	D8	PA9	-
CN9 digital	8	D7	PA8	-
	7	D6	PB10	TIM2_CH3
	6	D5	PB4	TIM3_CH1
	5	D4	PB5	-
	4	D3	PB3	TIM2_CH2
	3	D2	PA10	-
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

1. Please refer to *Table 9: Solder bridges* for details.

Source: User Manual 1724

Schematics

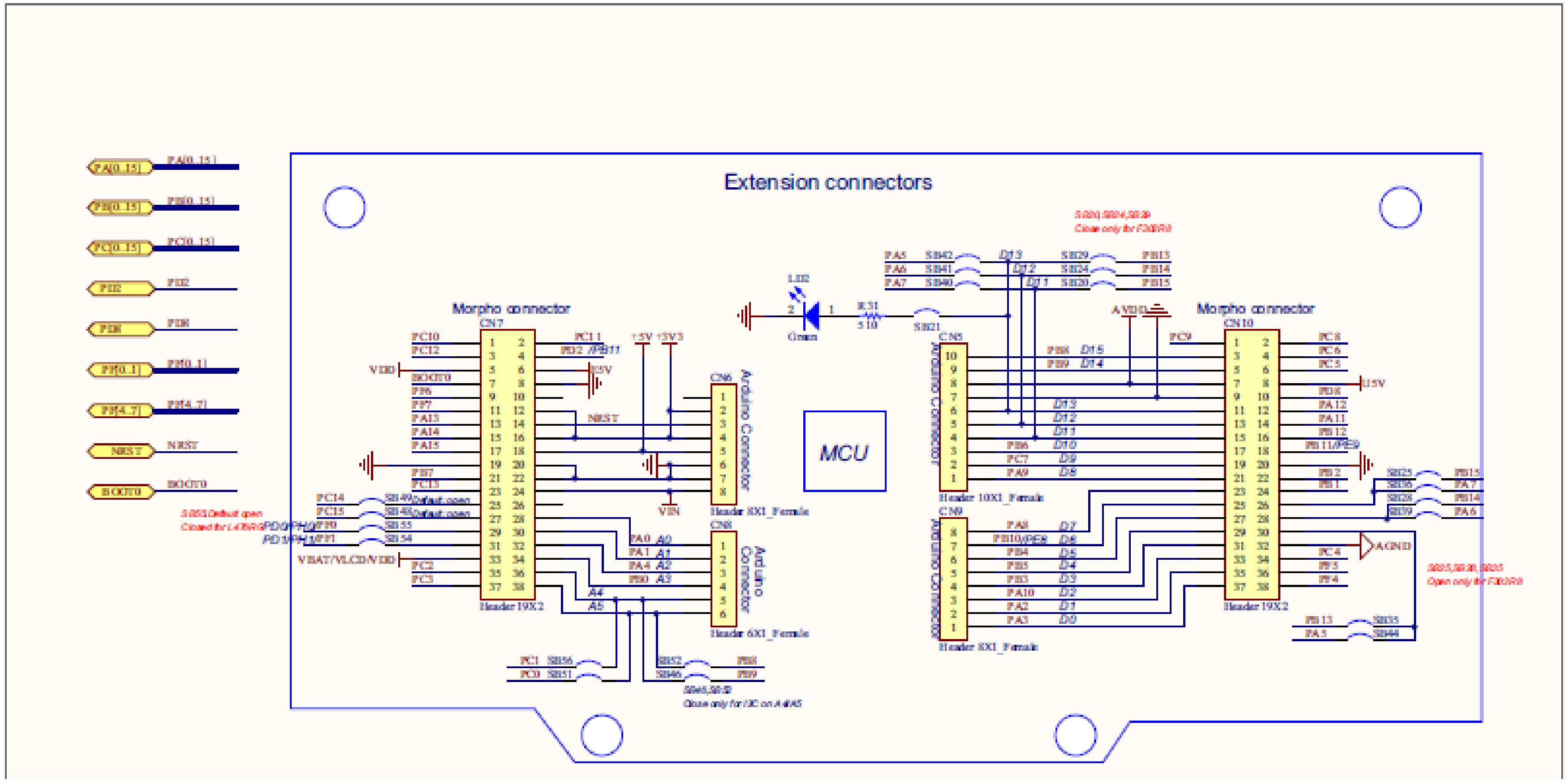


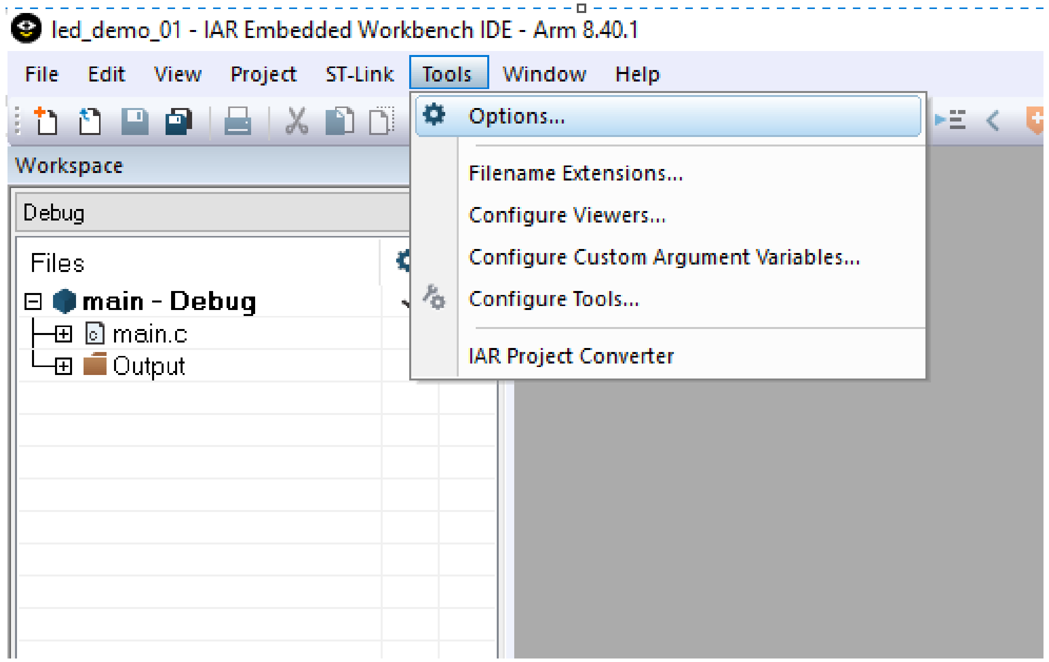
Figure 28. Electrical schematics (4/4)

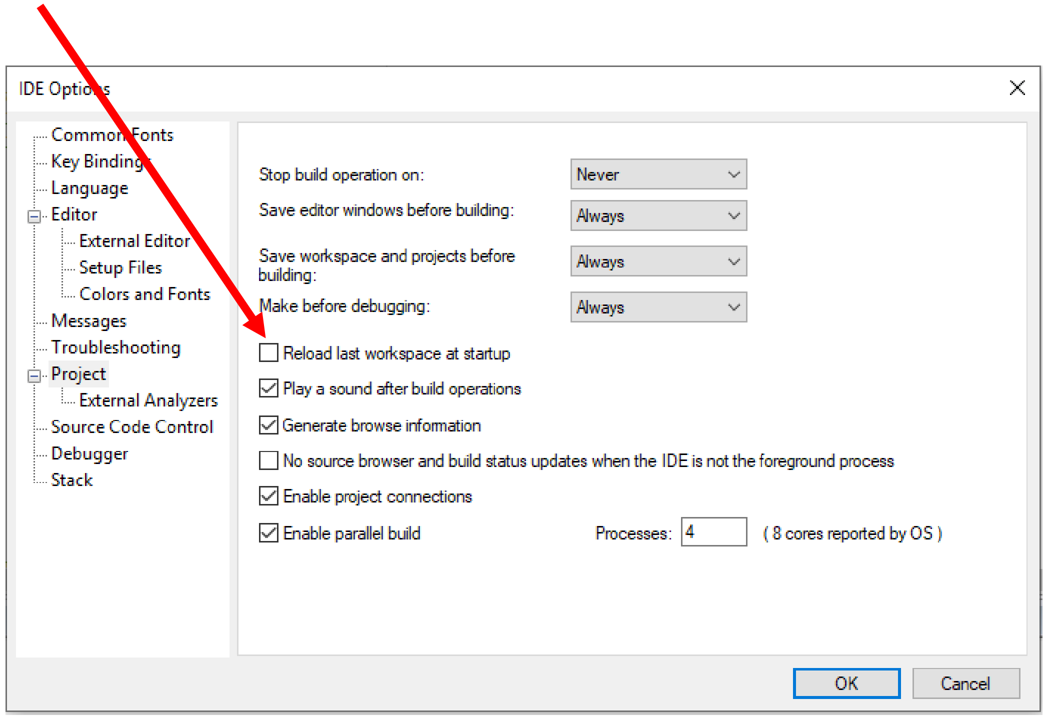
Source: User Manual 1724

Walk thru

- Starting with “counter” example
- Review the Memory Map
- Configure the GPIO to blink the LED
- Blinking the LED using C code
- Blinking the LED in loop

IAR Startup option





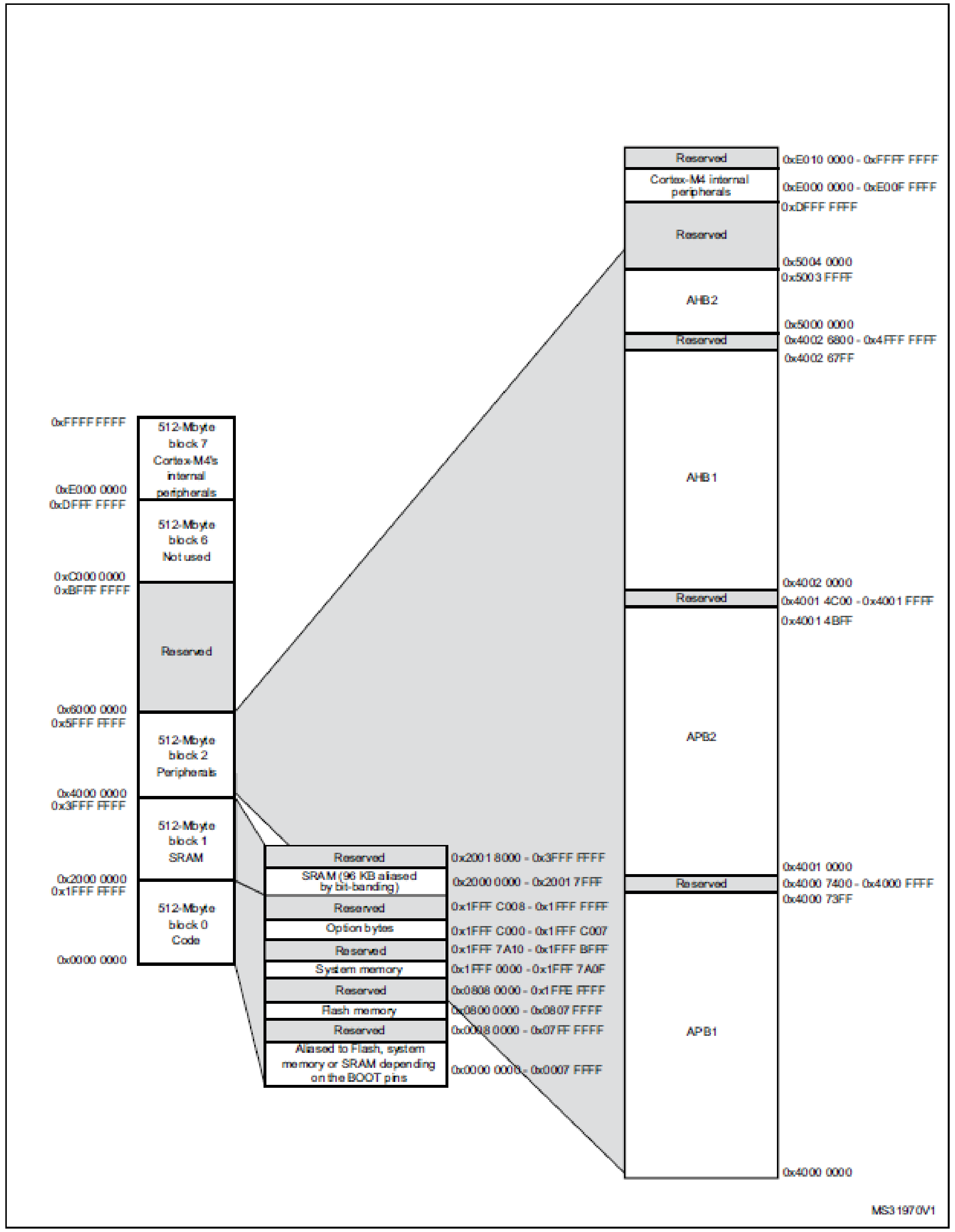
Stop build operation on: Never
Save editor windows before building: Always
Save workspace and projects before building: Always
Make before debugging: Always

- [ ] Reload last workspace at startup
- [✓] Play a sound after build operations
- [✓] Generate browse information
- [ ] No source browser and build status updates when the IDE is not the foreground process
- [✓] Enable project connections
- [✓] Enable parallel build — Processes: 4 (8 cores reported by OS)

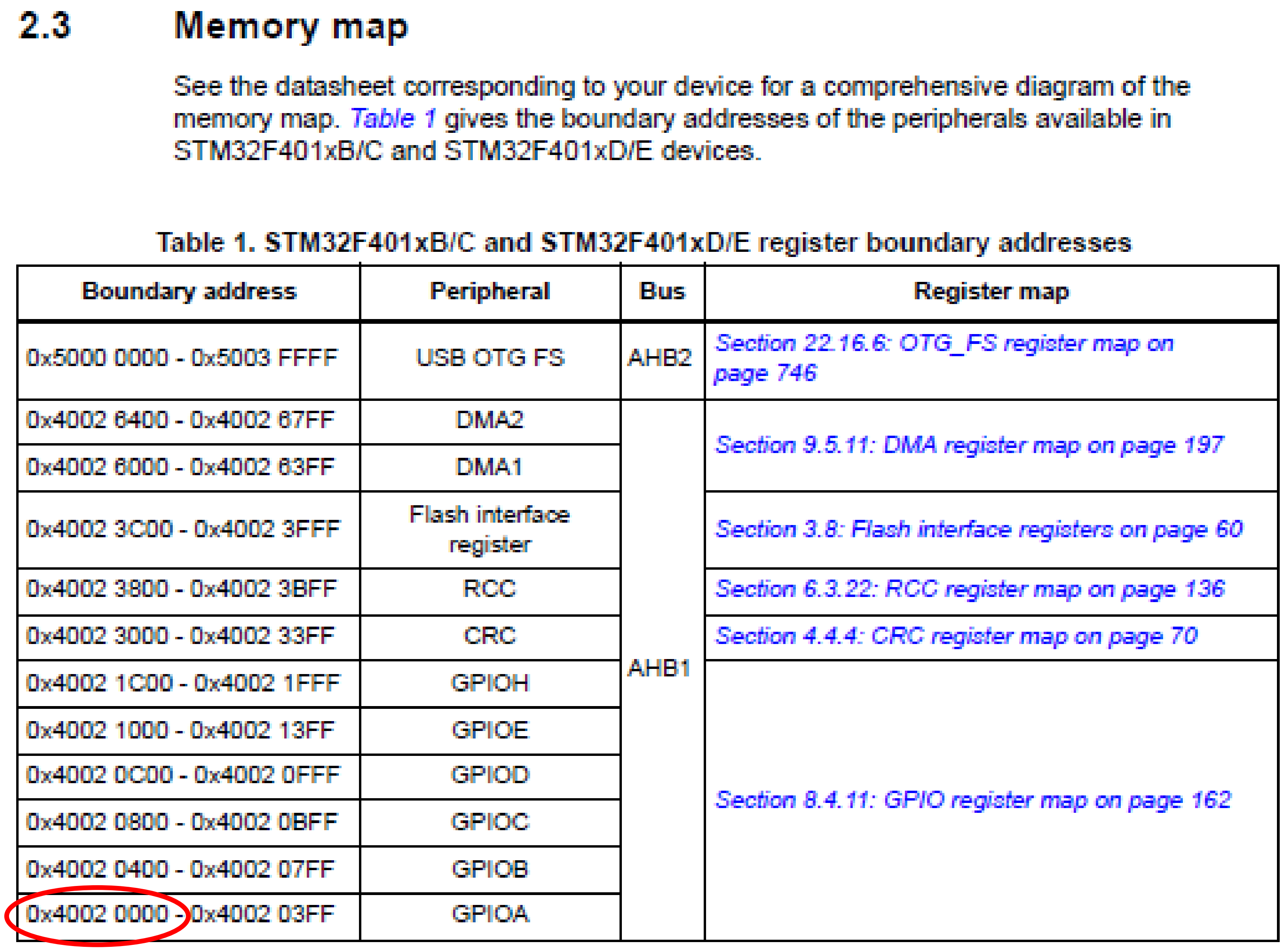
Memory Map review

- Looking back at “counter” demo…
- At the low addresses starting with 0
  - Machine instructions.
  - This is the compiled code of your program
  - Stored permanently inside the Flash
- Addresses starting at 0x2000 0000 are used for variables, such as the counter variable.
  - So 0x2000 0000 is the start of RAM
- RAM ends at 0x20018000
  - So it’s a block of size 0x18000
  - 0x18000 is 96KB in decimal.
  - Our microcontroller has 96KB of RAM.
- In order to blink the LED, we need to know the "map" of the various sections of the address space

Memory Map


0xFFFF FFFF
512-Mbyte block 7 Cortex-M4's internal peripherals
0xE000 0000
0xDFFF FFFF
512-Mbyte block 6 Not used
0xC000 0000
0xBFFF FFFF
Reserved
0x6000 0000
0x5FFF FFFF
512-Mbyte block 2 Peripherals
0x4000 0000
0x3FFF FFFF
512-Mbyte block 1 SRAM
0x2000 0000
0x1FFF FFFF
512-Mbyte block 0 Code
0x0000 0000
Reserved
0x2001 8000 - 0x3FFF FFFF
SRAM (96 KB aliased by bit-banding)
0x2000 0000 - 0x2001 7FFF
Reserved
0x1FFF C008 - 0x1FFF FFFF
Option bytes
0x1FFF C000 - 0x1FFF C007
Reserved
0x1FFF 7A10 - 0x1FFF BFFF
System memory
0x1FFF 0000 - 0x1FFF 7A0F
Reserved
0x0808 0000 - 0x1FFE FFFF
Flash memory
0x0800 0000 - 0x0807 FFFF
Reserved
0x0008 0000 - 0x07FF FFFF
Aliased to Flash, system memory or SRAM depending on the BOOT pins
0x0000 0000 - 0x0007 FFFF
Reserved
0xE010 0000 - 0xFFFF FFFF
Cortex-M4 internal peripherals
0xE000 0000 - 0xE00F FFFF
0xDFFF FFFF
Reserved
0x5004 0000
0x5003 FFFF
AHB2
0x5000 0000
Reserved
0x4002 6800 - 0x4FFF FFFF
0x4002 67FF
AHB1
0x4002 0000
Reserved
0x4001 4C00 - 0x4001 FFFF
0x4001 4BFF
APB2
0x4001 0000
Reserved
0x4000 7400 - 0x4000 FFFF
0x4000 73FF
APB1
0x4000 0000
MS31970V1

Source: STM32F401 Datasheet



2.3 Memory map

See the datasheet corresponding to your device for a comprehensive diagram of the memory map. *Table 1* gives the boundary addresses of the peripherals available in STM32F401xB/C and STM32F401xD/E devices.

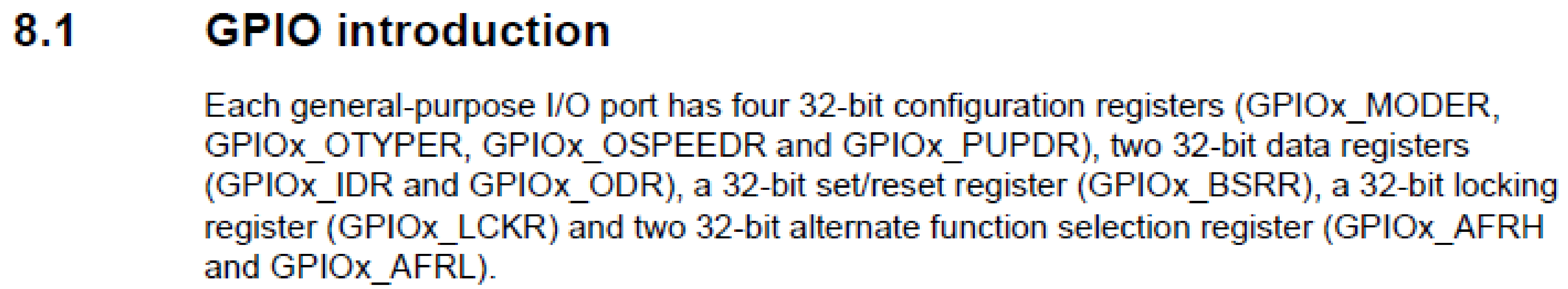
Table 1. STM32F401xB/C and STM32F401xD/E register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	*Section 22.16.6: OTG_FS register map on page 746*
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	*Section 9.5.11: DMA register map on page 197*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.8: Flash interface registers on page 60*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 6.3.22: RCC register map on page 136*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 70*
0x4002 1C00 - 0x4002 1FFF	GPIOH		*Section 8.4.11: GPIO register map on page 162*
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

Base address

Source: RM0368

GPIO registers

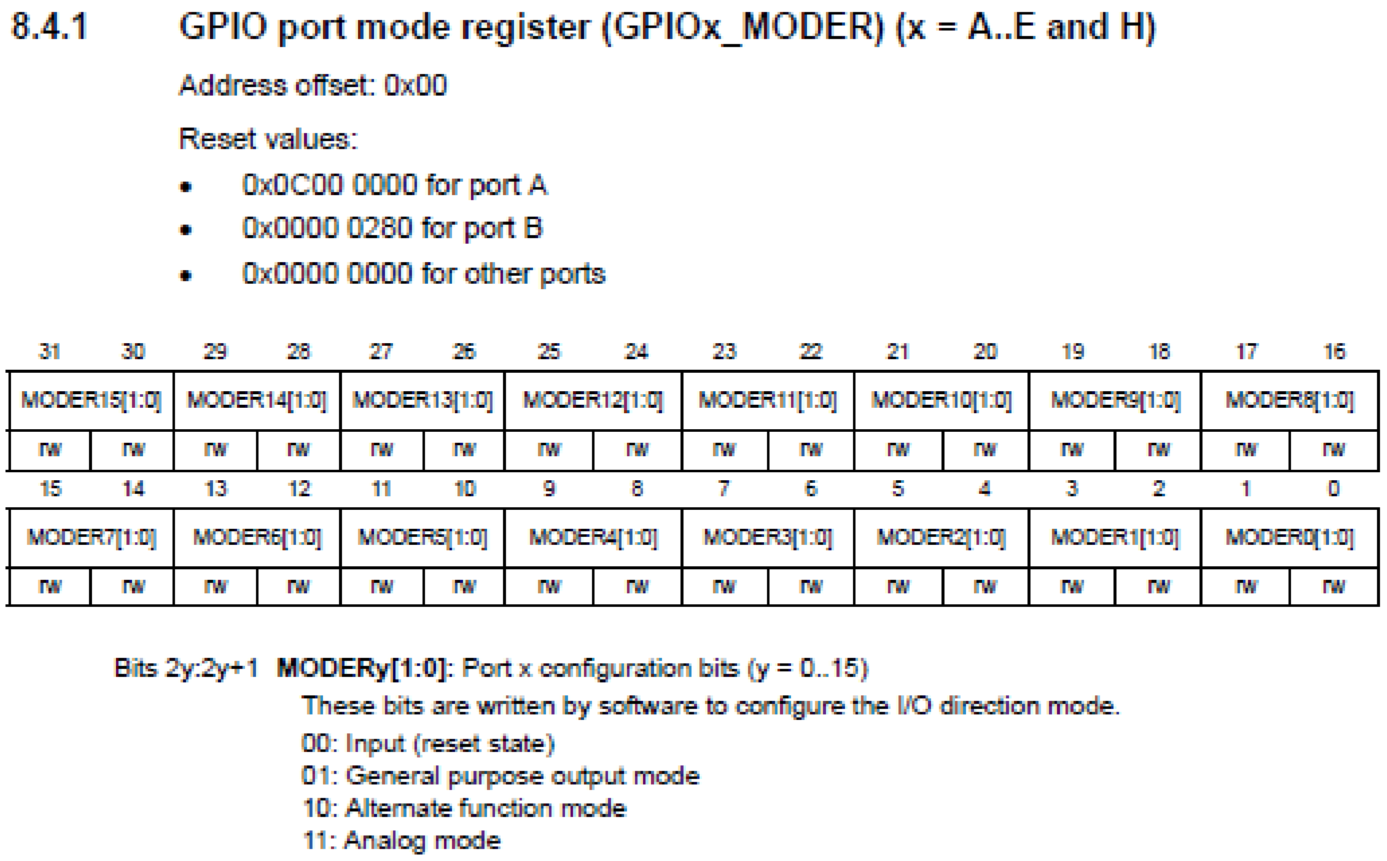


8.1 GPIO introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR), two 32-bit data registers (GPIOx_IDR and GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 32-bit locking register (GPIOx_LCKR) and two 32-bit alternate function selection register (GPIOx_AFRH and GPIOx_AFRL).

Source: RM0368

Setup GPIO Port to “output”



8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0x0C00 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

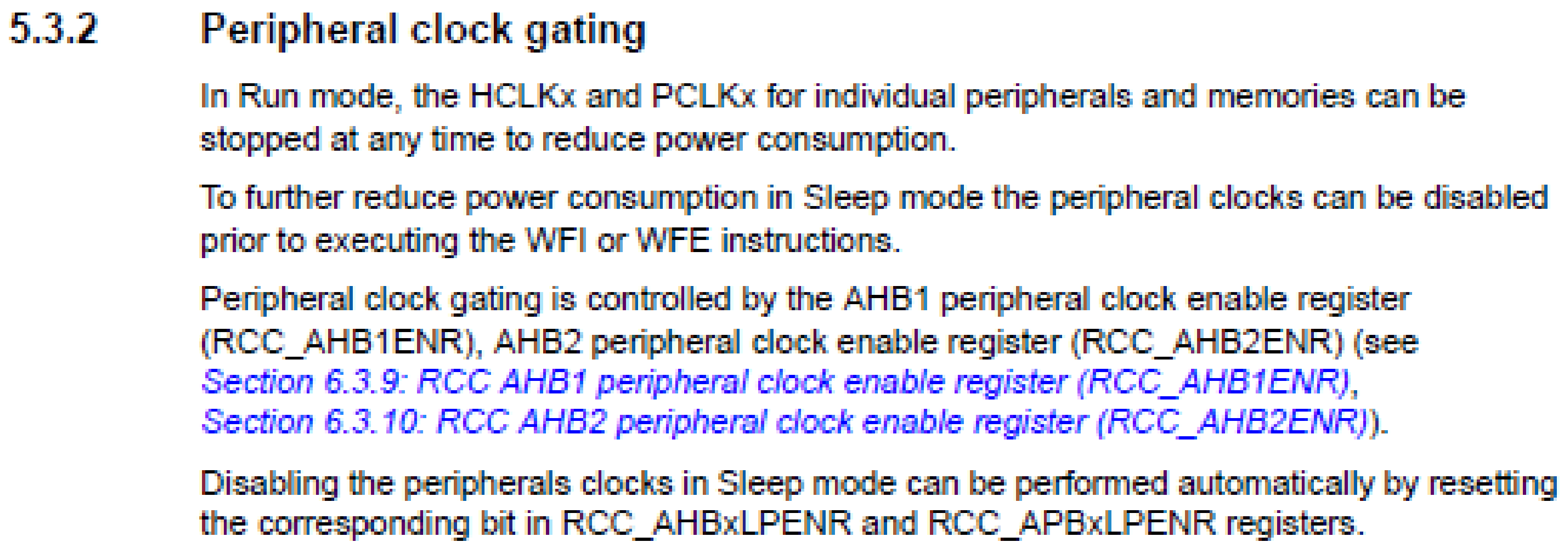
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Source: RM0368

Clock Gating

- The Cortex-M processors provide a number of low power features.
- These include multiple sleep modes defined in the architecture.
- And integrated architectural ***clock gating*** support, which allows clock circuits for parts of the processor to be deactivated when the section is not in use.

Clock Gating



5.3.2 Peripheral clock gating

In Run mode, the HCLKx and PCLKx for individual peripherals and memories can be stopped at any time to reduce power consumption.

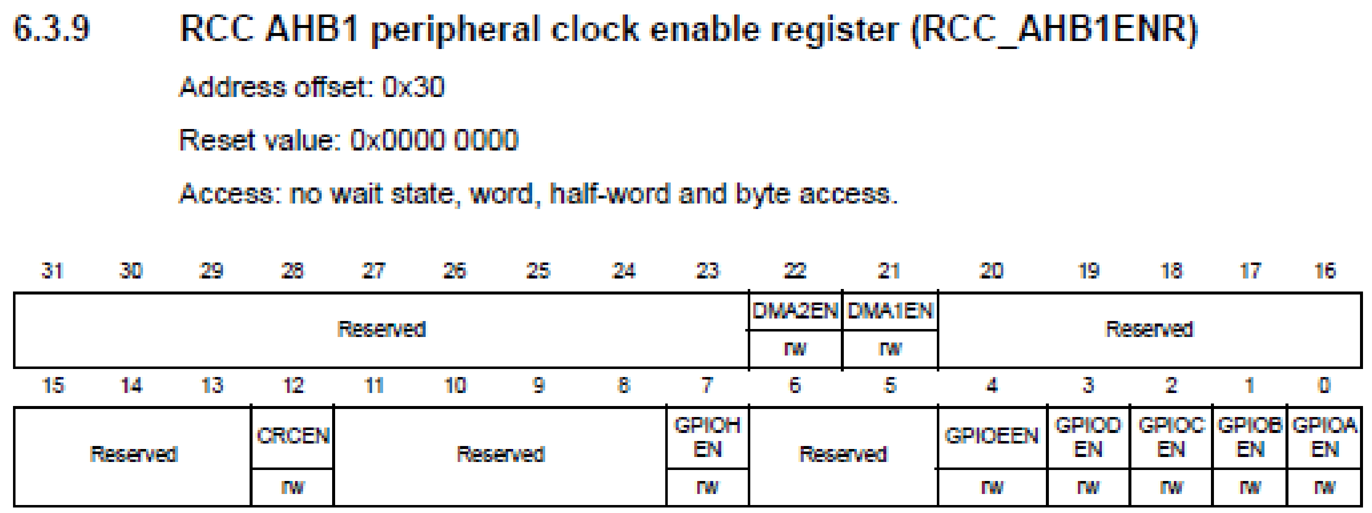
To further reduce power consumption in Sleep mode the peripheral clocks can be disabled prior to executing the WFI or WFE instructions.

Peripheral clock gating is controlled by the AHB1 peripheral clock enable register (RCC_AHB1ENR), AHB2 peripheral clock enable register (RCC_AHB2ENR) (see *Section 6.3.9: RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)*, *Section 6.3.10: RCC AHB2 peripheral clock enable register (RCC_AHB2ENR)*).

Disabling the peripherals clocks in Sleep mode can be performed automatically by resetting the corresponding bit in RCC_AHBxLPENR and RCC_APBxLPENR registers.

Source: RM0368

Peripheral Clock Enable Register



6.3.9 **RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)**

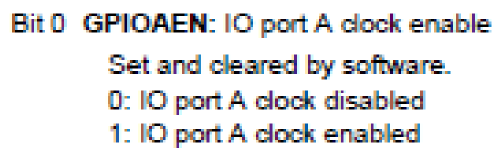
Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOH EN	Reserved		GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw



Bit 0 **GPIOAEN:** IO port A clock enable

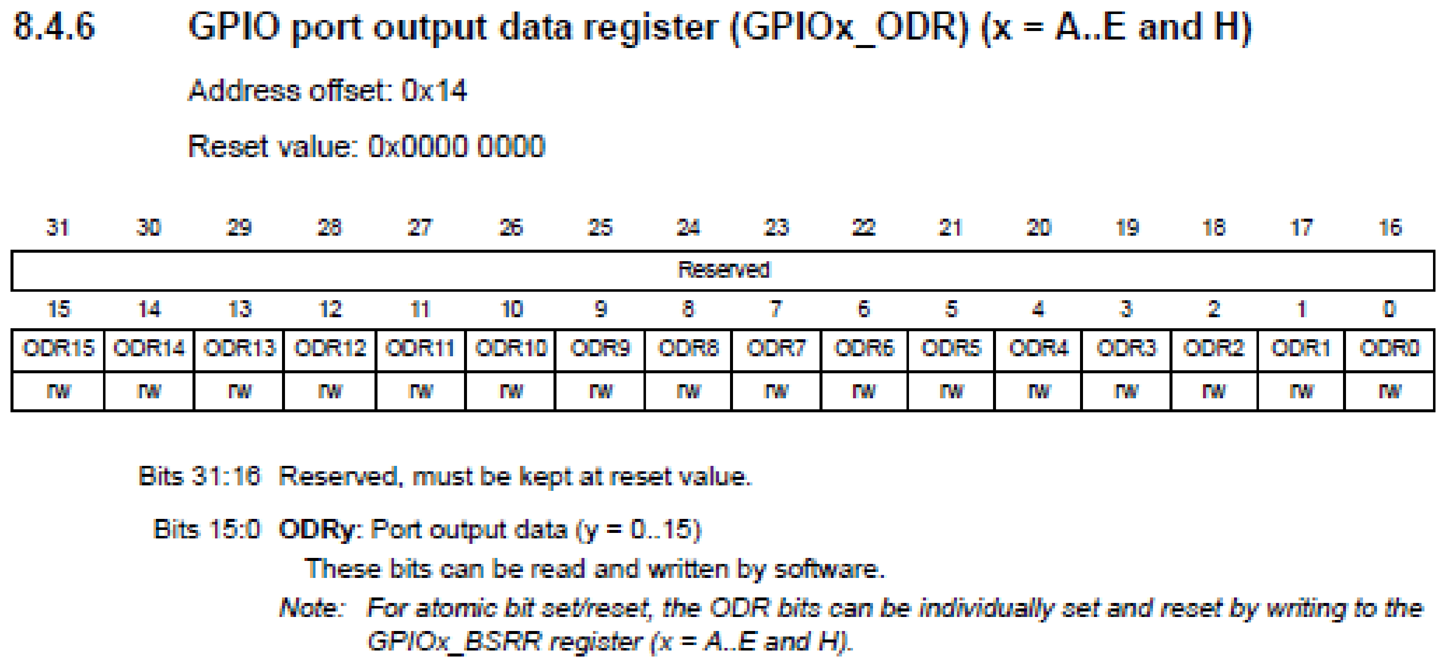
Set and cleared by software.

0: IO port A clock disabled

1: IO port A clock enabled

Source: RM0368

Write to the GPIO output



8.4.6 **GPIO port output data register (GPIOx_ODR) (x = A..E and H)**

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

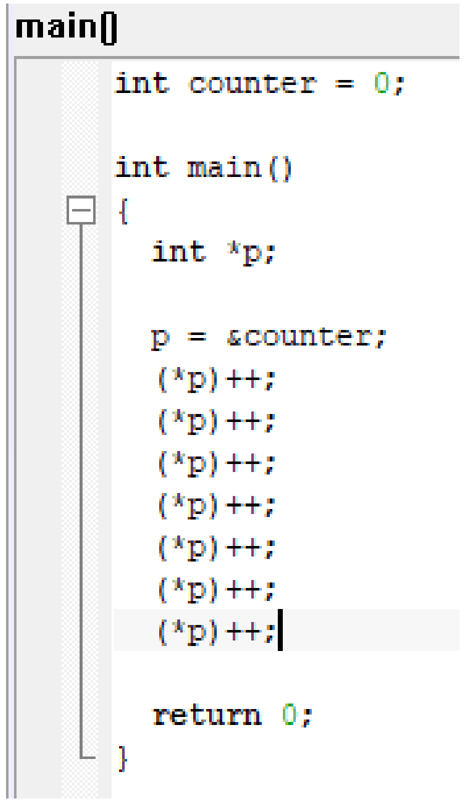
Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..E and H).

Source: RM0368

Blinking the LED using C code

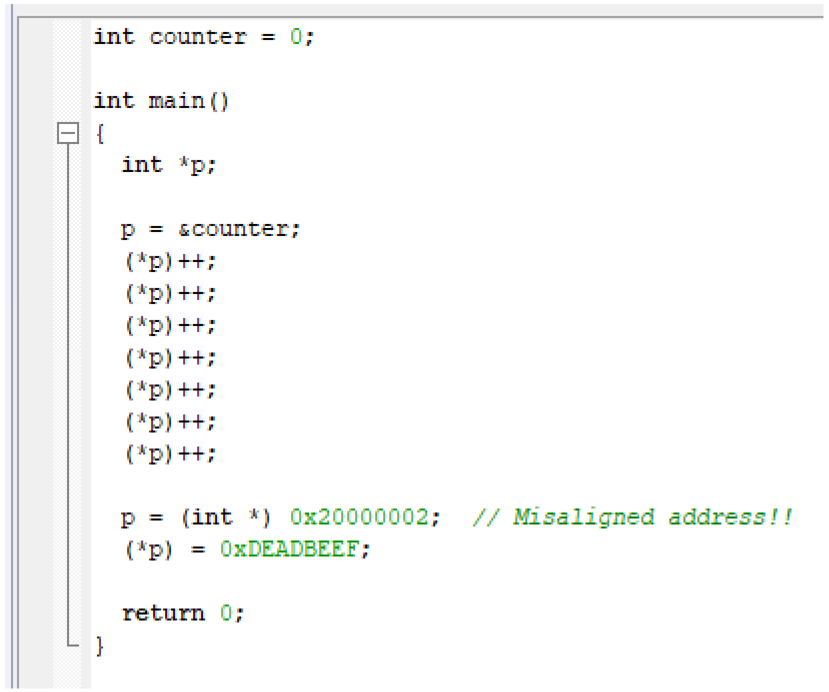
- Remember pointers?
- We can use pointers to point to any memory address and potentially manipulate the value stored in there.
- So our LED program becomes:
  - Writing values to memory addresses

Counter example with Pointers

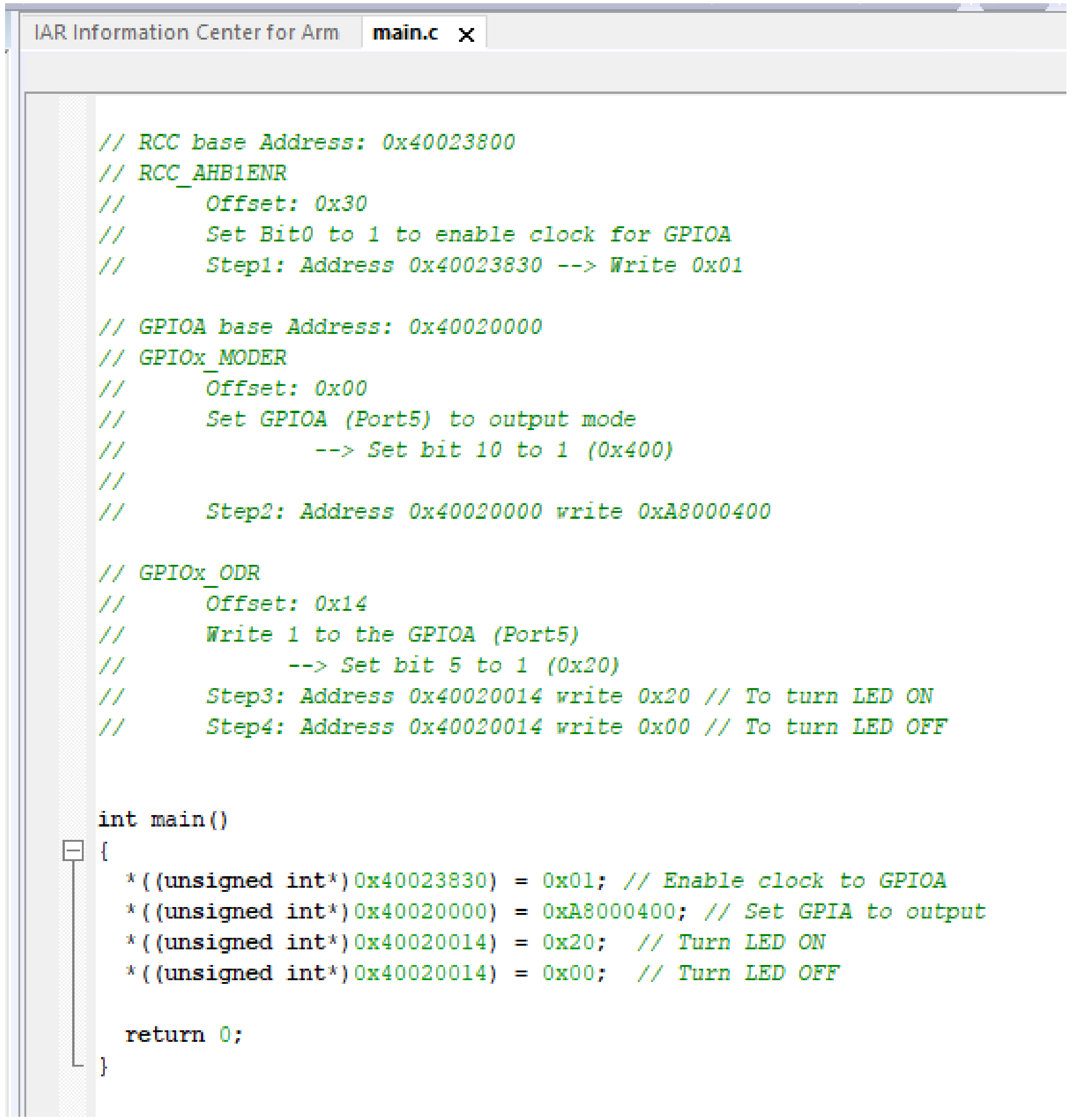


main()

```
int counter = 0;

int main()
{
  int *p;

  p = &counter;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;

  return 0;
}
```



```
int counter = 0;

int main()
{
  int *p;

  p = &counter;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;
  (*p)++;

  p = (int *) 0x20000002;   // Misaligned address!!
  (*p) = 0xDEADBEEF;

  return 0;
}
```



```c
// RCC base Address: 0x40023800
// RCC_AHB1ENR
//      Offset: 0x30
//      Set Bit0 to 1 to enable clock for GPIOA
//      Step1: Address 0x40023830 --> Write 0x01

// GPIOA base Address: 0x40020000
// GPIOx_MODER
//      Offset: 0x00
//      Set GPIOA (Port5) to output mode
//              --> Set bit 10 to 1 (0x400)
//
//      Step2: Address 0x40020000 write 0xA8000400

// GPIOx_ODR
//      Offset: 0x14
//      Write 1 to the GPIOA (Port5)
//            --> Set bit 5 to 1 (0x20)
//      Step3: Address 0x40020014 write 0x20 // To turn LED ON
//      Step4: Address 0x40020014 write 0x00 // To turn LED OFF


int main()
{
  *((unsigned int*)0x40023830) = 0x01; // Enable clock to GPIOA
  *((unsigned int*)0x40020000) = 0xA8000400; // Set GPIA to output
  *((unsigned int*)0x40020014) = 0x20;  // Turn LED ON
  *((unsigned int*)0x40020014) = 0x00;  // Turn LED OFF

  return 0;
}
```

Blinking LED in C

Summary of steps

// 1. Enable clock

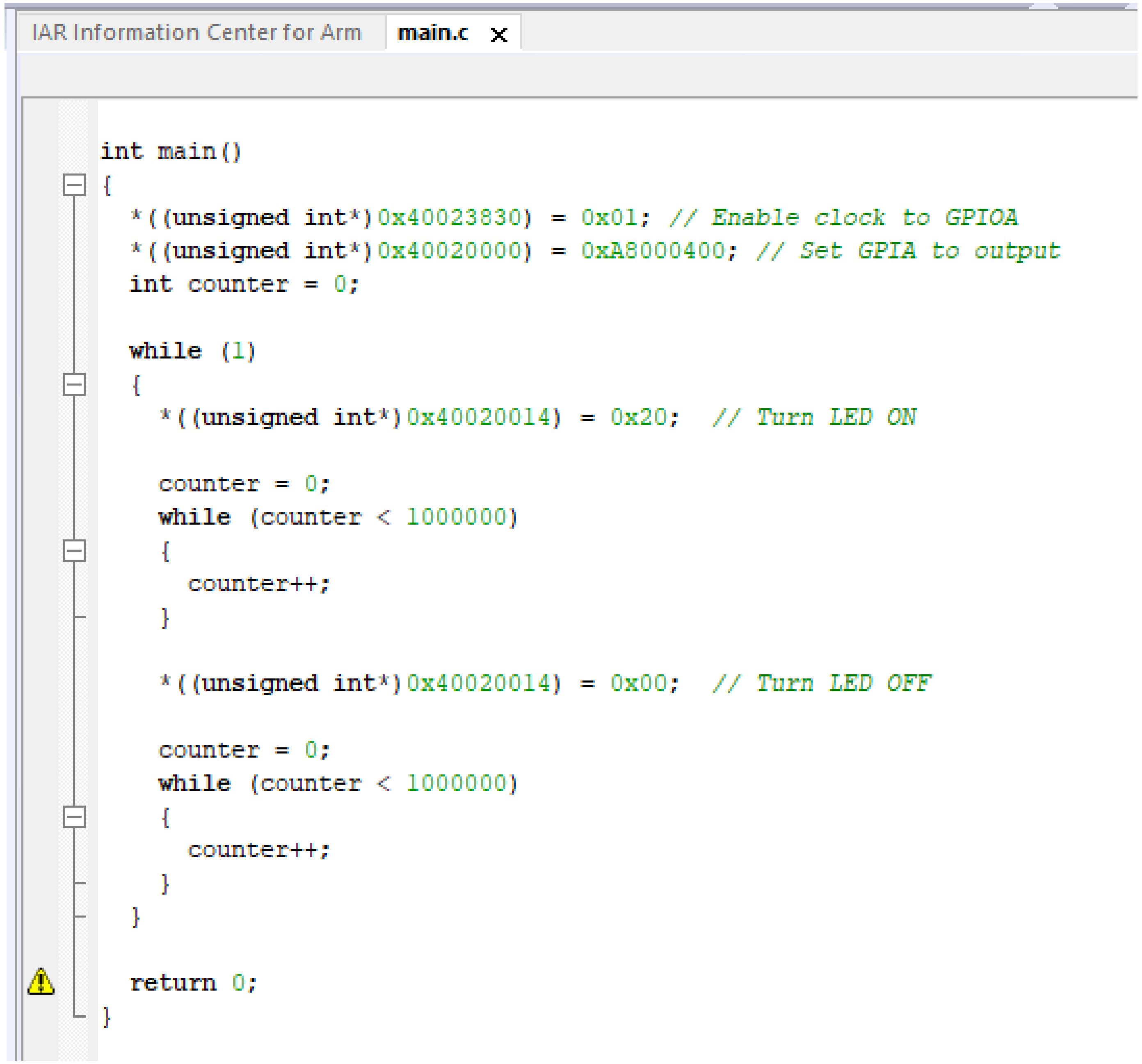
```
    // RCC_BASE = 0x40023800 // Base Address for RCC registers
    // RCC_AHB1_ENR_offset: 0x30 // Peripheral Clock Enable Register
    // Set bit[0] to 1
```

// 2. Set GPIOA to Output mode

```
    // GPIOA_BASE = 0x40020000 // Base Address for GPIO registers
    // GPIOx_MODER offset is 0x00 // To enable port mode (IN, OUT, AF..)
    // Set bit[11:10] to 0x01  so --> 0x400 // To enable Port5 as output
```

// 3. Write to GPIO Data Register to toggle LED

```
     // GPIOA_BASE = 0x40020000 // Base Address for GPIO registers
    // GPIOx_ODR offset: 0x14 // Port output data register
    // Set bit[5] to 1 --> 0x20; // Turn LED ON
    // Set bit[5] to 0 --> 0x00; // Turn LED OFF
```


IAR Information Center for Arm | main.c

```c
int main()
{
  *((unsigned int*)0x40023830) = 0x01; // Enable clock to GPIOA
  *((unsigned int*)0x40020000) = 0xA8000400; // Set GPIA to output
  int counter = 0;

  while (1)
  {
    *((unsigned int*)0x40020014) = 0x20;  // Turn LED ON

    counter = 0;
    while (counter < 1000000)
    {
      counter++;
    }

    *((unsigned int*)0x40020014) = 0x00;  // Turn LED OFF

    counter = 0;
    while (counter < 1000000)
    {
      counter++;
    }
  }

  return 0;
}
```

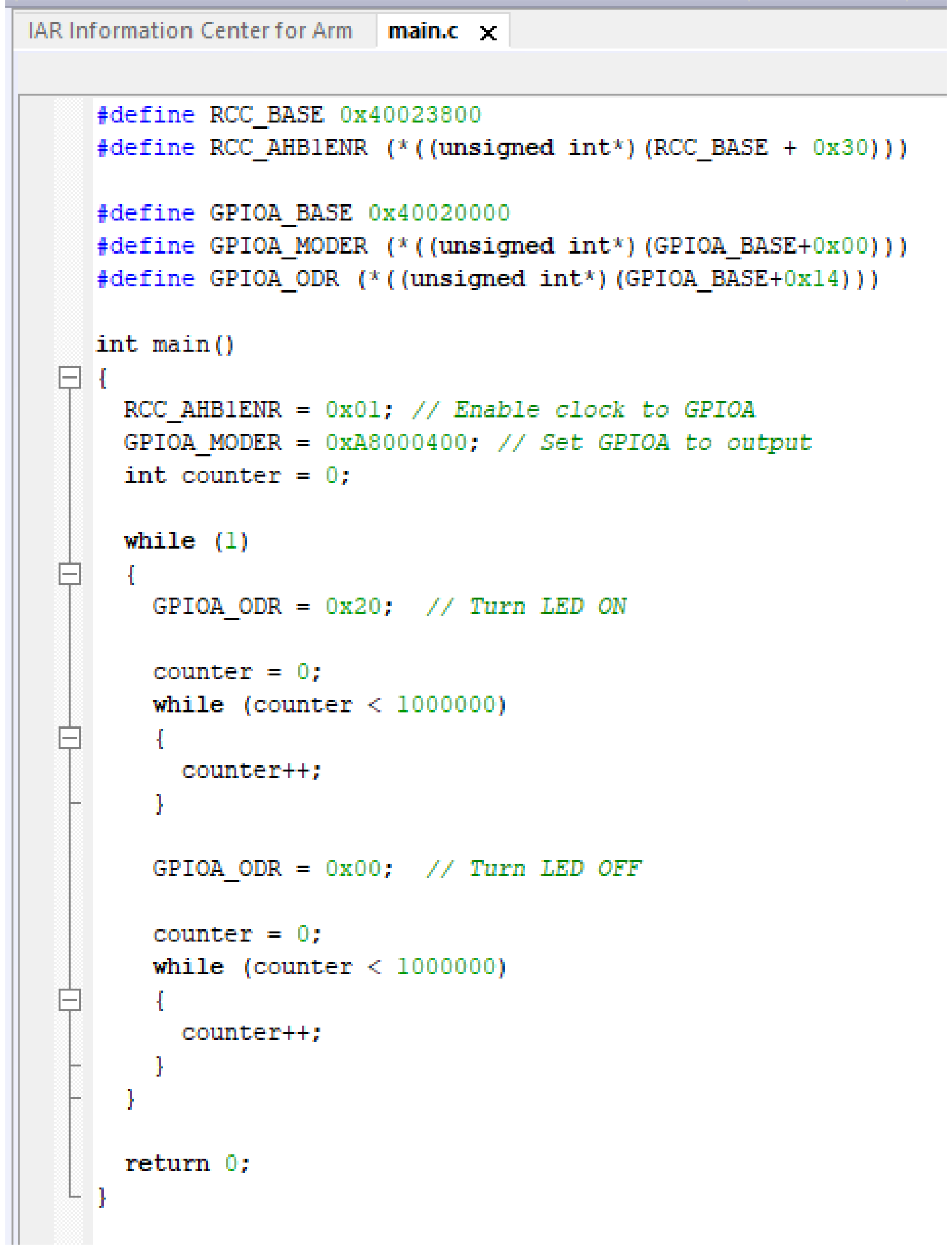
Blinking the LED in a loop

C Programming (Continued)

- Preprocessor
- Volatile
- Bit-wise operators in C

Making the LED code more readable

- Create names for the registers
- Use preprocessors
- Beware of how you define your macros so they have the same effect regardless of where they’re used.
  - EX: *((unsigned int*)0x40023830) vs (*((unsigned int*)0x40023830))
- Let’s make macros
  - … and macros within macros



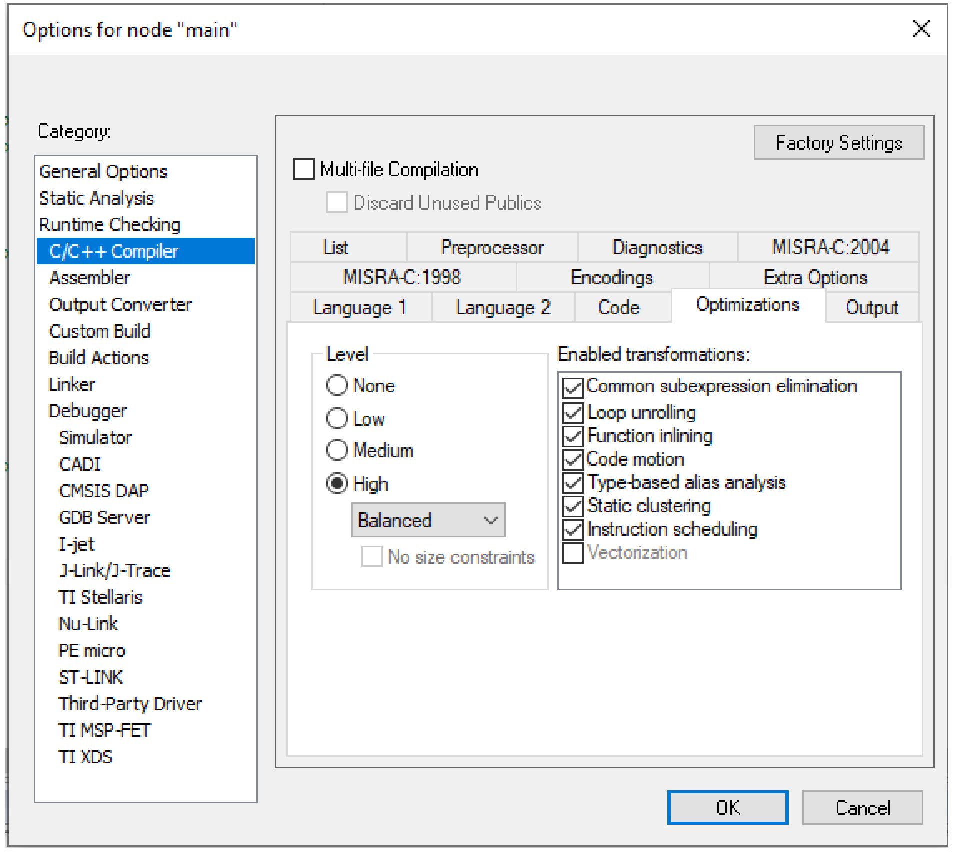
```c
#define RCC_BASE 0x40023800
#define RCC_AHB1ENR (*((unsigned int*)(RCC_BASE + 0x30)))

#define GPIOA_BASE 0x40020000
#define GPIOA_MODER (*((unsigned int*)(GPIOA_BASE+0x00)))
#define GPIOA_ODR (*((unsigned int*)(GPIOA_BASE+0x14)))

int main()
{
  RCC_AHB1ENR = 0x01; // Enable clock to GPIOA
  GPIOA_MODER = 0xA8000400; // Set GPIOA to output
  int counter = 0;

  while (1)
  {
    GPIOA_ODR = 0x20;  // Turn LED ON

    counter = 0;
    while (counter < 1000000)
    {
      counter++;
    }

    GPIOA_ODR = 0x00;  // Turn LED OFF

    counter = 0;
    while (counter < 1000000)
    {
      counter++;
    }
  }

  return 0;
}
```

Using macros

C language Volatile Qualifier

- The “**volatile**” qualifier tells the compiler that a variable might be changed even though there are no statements in the program that appear to change it.
  - EX: User input switch writing to a GPIO register.
- The compiler can optimize access to non-volatile variables by reading their value into a CPU register, working with that register for a while, and eventually writing the value in the register back to the object.
- The compiler is NOT permitted to do this optimization with "volatile" variables. Every time the source program says to read-from or to write-to a "volatile" variable, the compiler will have to do so.
- The "volatile" qualifier is useful for I/O registers
- But it can also be useful for "normal" variables to prevent optimizations that the compiler might do.
  - EX: “counter” variable in our delay loop – From the compiler’s perspective, makes no contribution to any computation.
- [DEMO]

Compiler optimization



Options for node "main"

Category:

- General Options
- Static Analysis
- Runtime Checking
- C/C++ Compiler
- Assembler
- Output Converter
- Custom Build
- Build Actions
- Linker
- Debugger
  - Simulator
  - CADI
  - CMSIS DAP
  - GDB Server
  - I-jet
  - J-Link/J-Trace
  - TI Stellaris
  - Nu-Link
  - PE micro
  - ST-LINK
  - Third-Party Driver
  - TI MSP-FET
  - TI XDS

Factory Settings

☐ Multi-file Compilation

☐ Discard Unused Publics

List | Preprocessor | Diagnostics | MISRA-C:2004 | MISRA-C:1998 | Encodings | Extra Options | Language 1 | Language 2 | Code | Optimizations | Output

Level
- ○ None
- ○ Low
- ○ Medium
- ● High

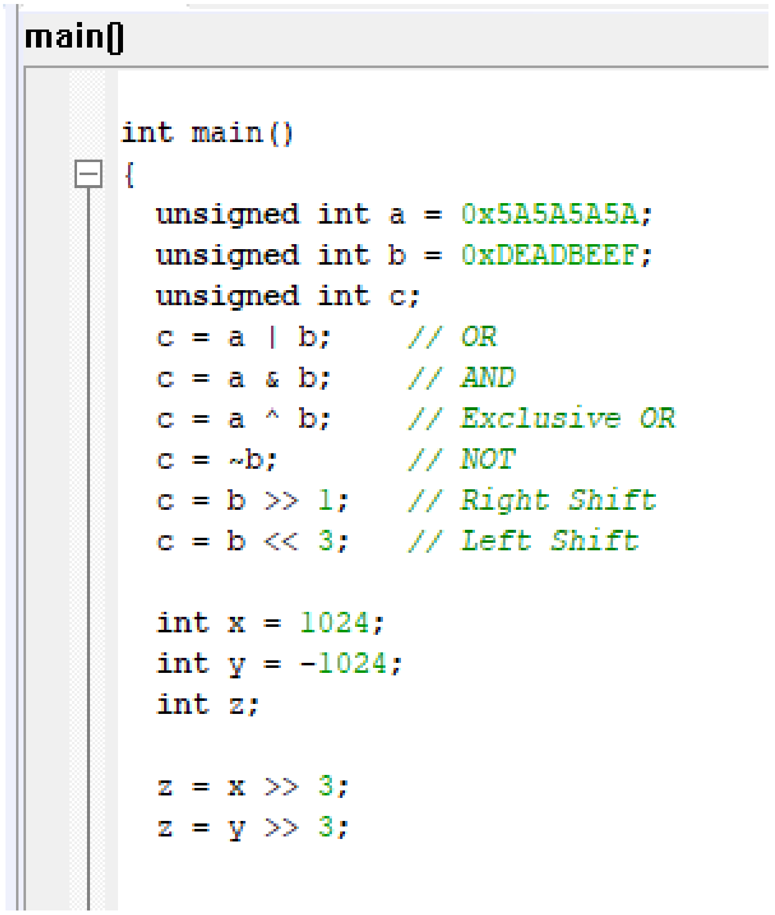
Balanced

☐ No size constraints

Enabled transformations:
- ☑ Common subexpression elimination
- ☑ Loop unrolling
- ☑ Function inlining
- ☑ Code motion
- ☑ Type-based alias analysis
- ☑ Static clustering
- ☑ Instruction scheduling
- ☐ Vectorization

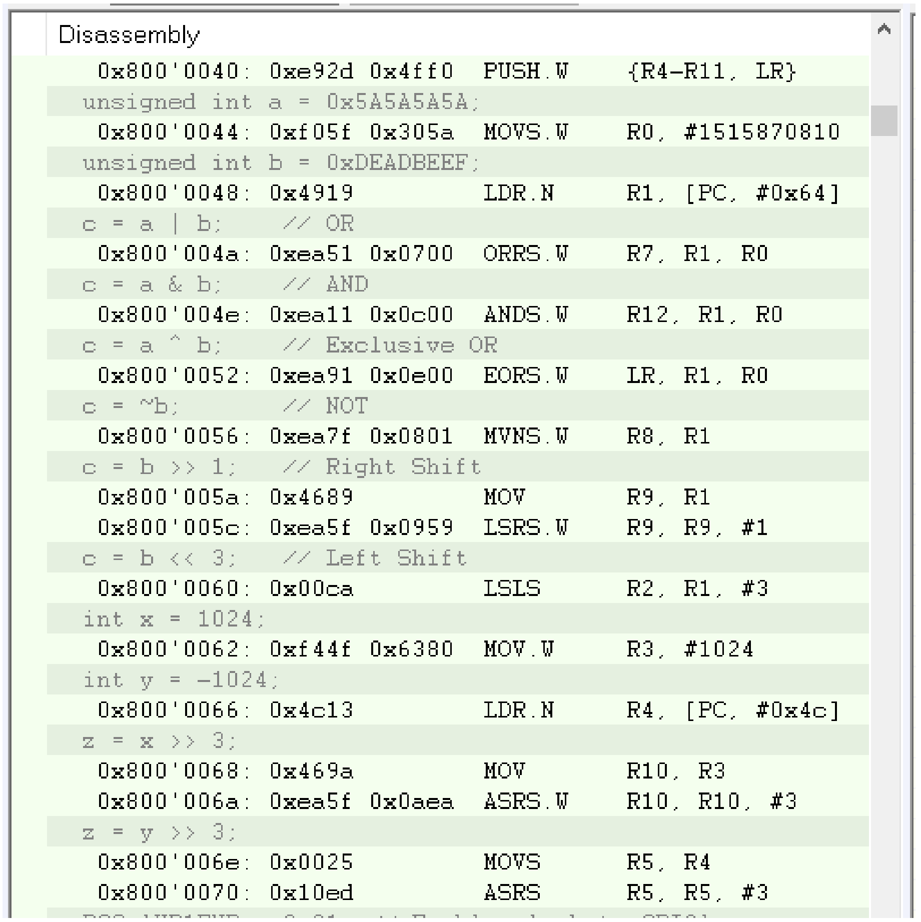
OK | Cancel

Bit-wise operators



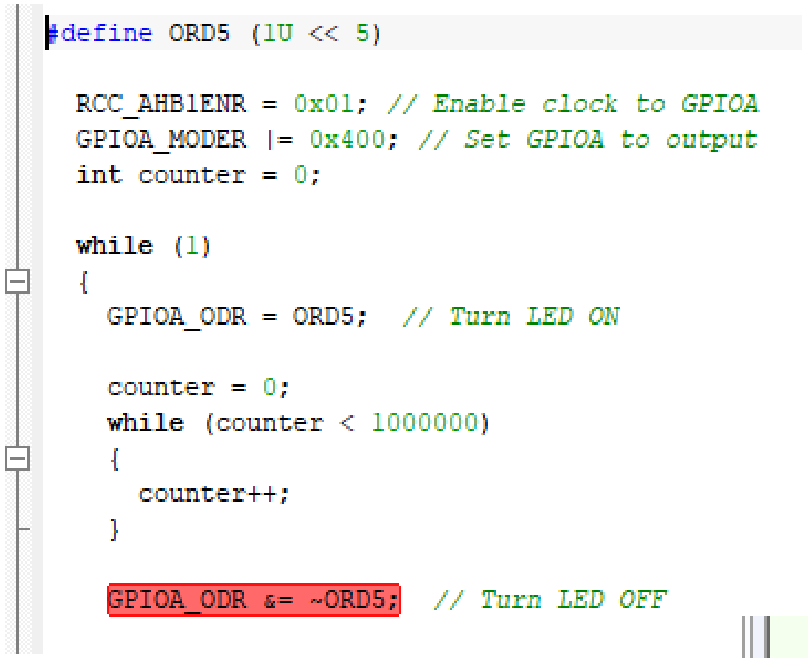
main()

```
int main()
{
  unsigned int a = 0x5A5A5A5A;
  unsigned int b = 0xDEADBEEF;
  unsigned int c;
  c = a | b;     // OR
  c = a & b;     // AND
  c = a ^ b;     // Exclusive OR
  c = ~b;        // NOT
  c = b >> 1;    // Right Shift
  c = b << 3;    // Left Shift

  int x = 1024;
  int y = -1024;
  int z;

  z = x >> 3;
  z = y >> 3;
```

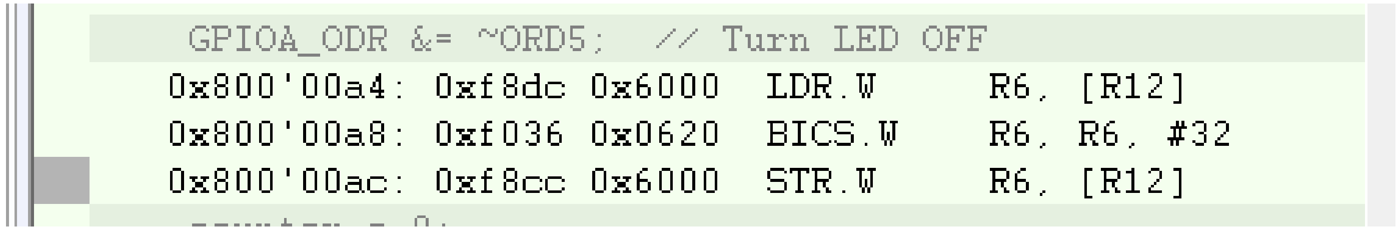


Disassembly

```
  0x800'0040: 0xe92d 0x4ff0  PUSH.W    {R4-R11, LR}
unsigned int a = 0x5A5A5A5A;
  0x800'0044: 0xf05f 0x305a  MOVS.W    R0, #1515870810
unsigned int b = 0xDEADBEEF;
  0x800'0048: 0x4919         LDR.N     R1, [PC, #0x64]
c = a | b;    // OR
  0x800'004a: 0xea51 0x0700  ORRS.W    R7, R1, R0
c = a & b;    // AND
  0x800'004e: 0xea11 0x0c00  ANDS.W    R12, R1, R0
c = a ^ b;    // Exclusive OR
  0x800'0052: 0xea91 0x0e00  EORS.W    LR, R1, R0
c = ~b;       // NOT
  0x800'0056: 0xea7f 0x0801  MVNS.W    R8, R1
c = b >> 1;   // Right Shift
  0x800'005a: 0x4689         MOV       R9, R1
  0x800'005c: 0xea5f 0x0959  LSRS.W    R9, R9, #1
c = b << 3;   // Left Shift
  0x800'0060: 0x00ca         LSLS      R2, R1, #3
int x = 1024;
  0x800'0062: 0xf44f 0x6380  MOV.W     R3, #1024
int y = -1024;
  0x800'0066: 0x4c13         LDR.N     R4, [PC, #0x4c]
z = x >> 3;
  0x800'0068: 0x469a         MOV       R10, R3
  0x800'006a: 0xea5f 0x0aea  ASRS.W    R10, R10, #3
z = y >> 3;
  0x800'006e: 0x0025         MOVS      R5, R4
  0x800'0070: 0x10ed         ASRS      R5, R5, #3
```



```
#define ORD5 (1U << 5)

  RCC_AHB1ENR = 0x01; // Enable clock to GPIOA
  GPIOA_MODER |= 0x400; // Set GPIOA to output
  int counter = 0;

  while (1)
  {
    GPIOA_ODR = ORD5;  // Turn LED ON

    counter = 0;
    while (counter < 1000000)
    {
      counter++;
    }

    GPIOA_ODR &= ~ORD5;  // Turn LED OFF
```



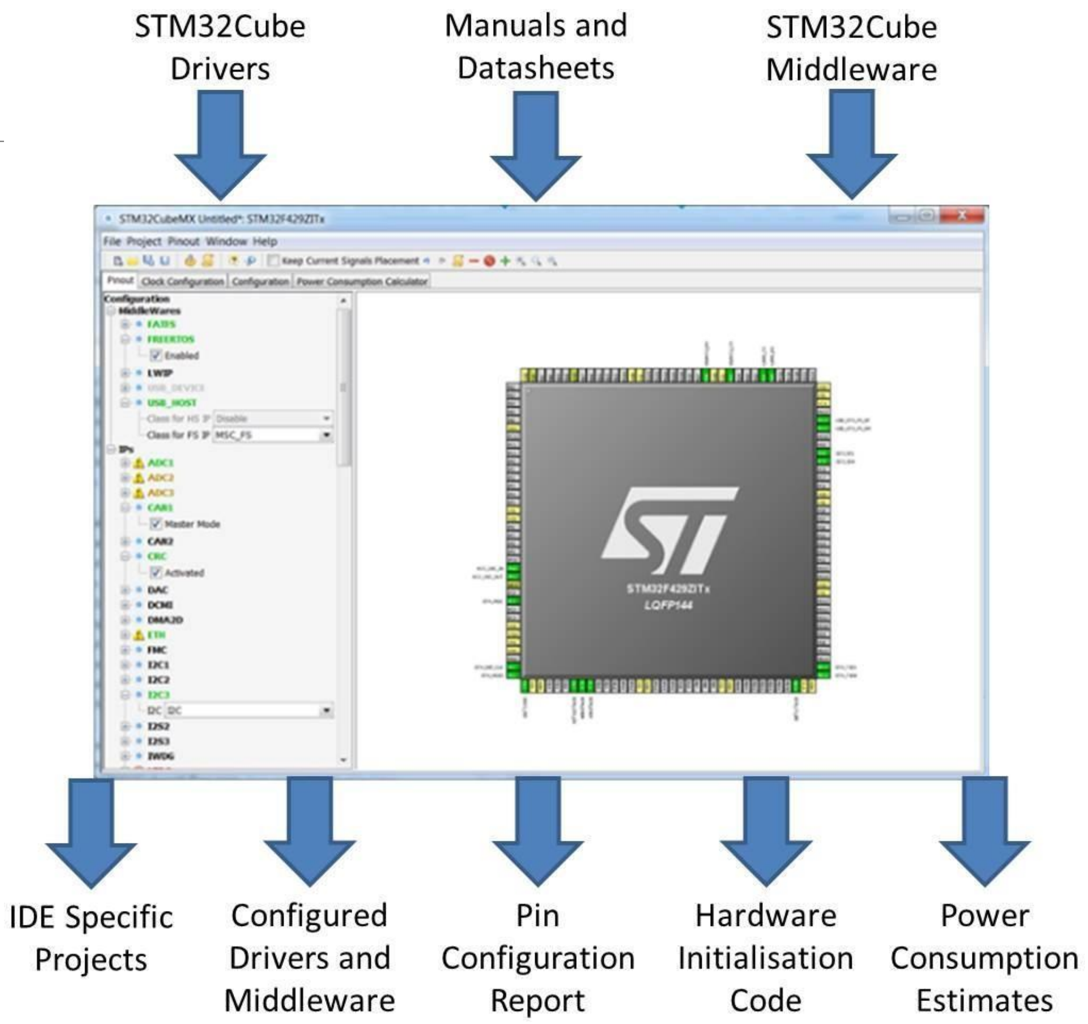
```
      GPIOA_ODR &= ~ORD5;   // Turn LED OFF
    0x800'00a4: 0xf8dc 0x6000  LDR.W     R6, [R12]
    0x800'00a8: 0xf036 0x0620  BICS.W    R6, R6, #32
    0x800'00ac: 0xf8cc 0x6000  STR.W     R6, [R12]
```

STM CubeMx

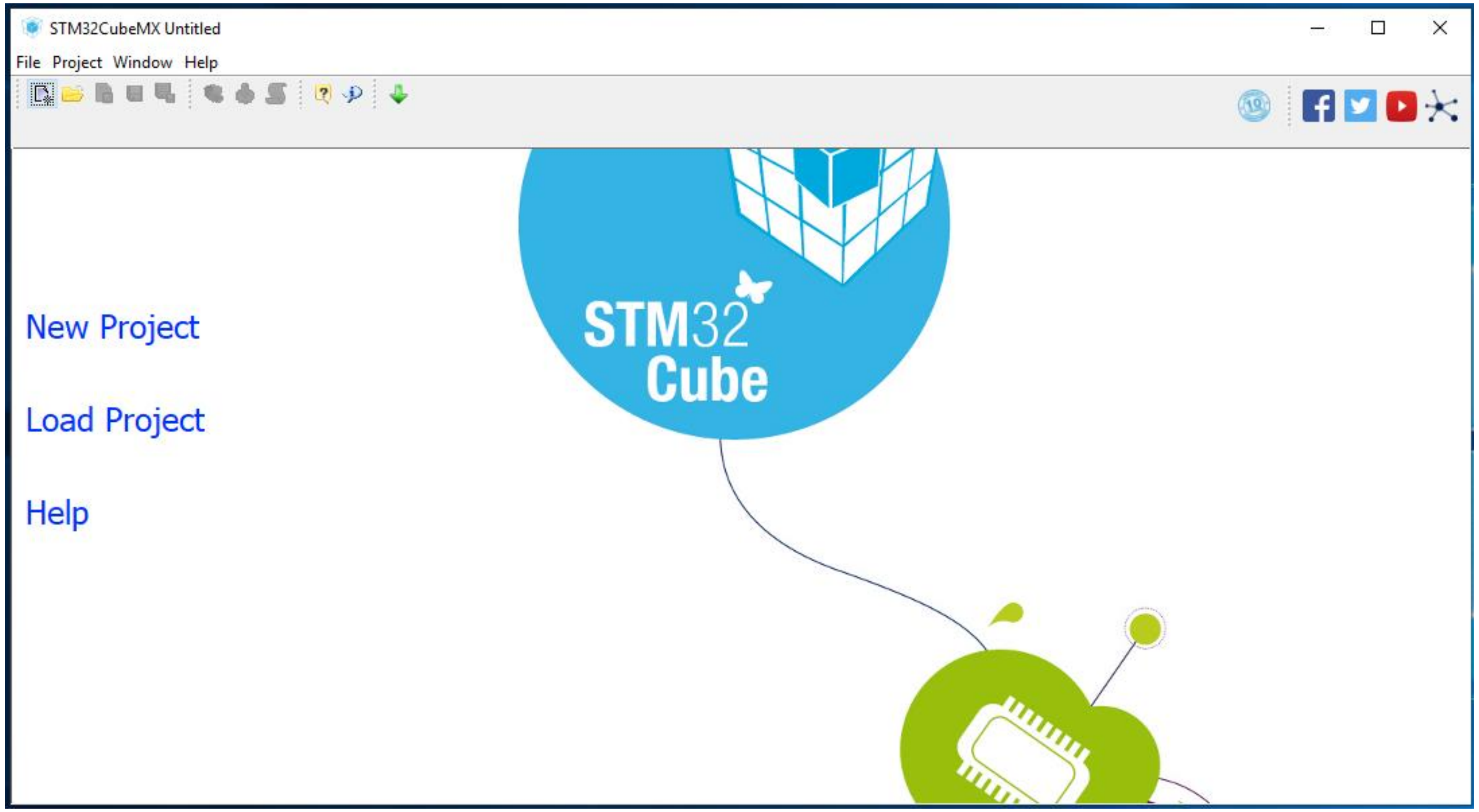
- Blinking the LED

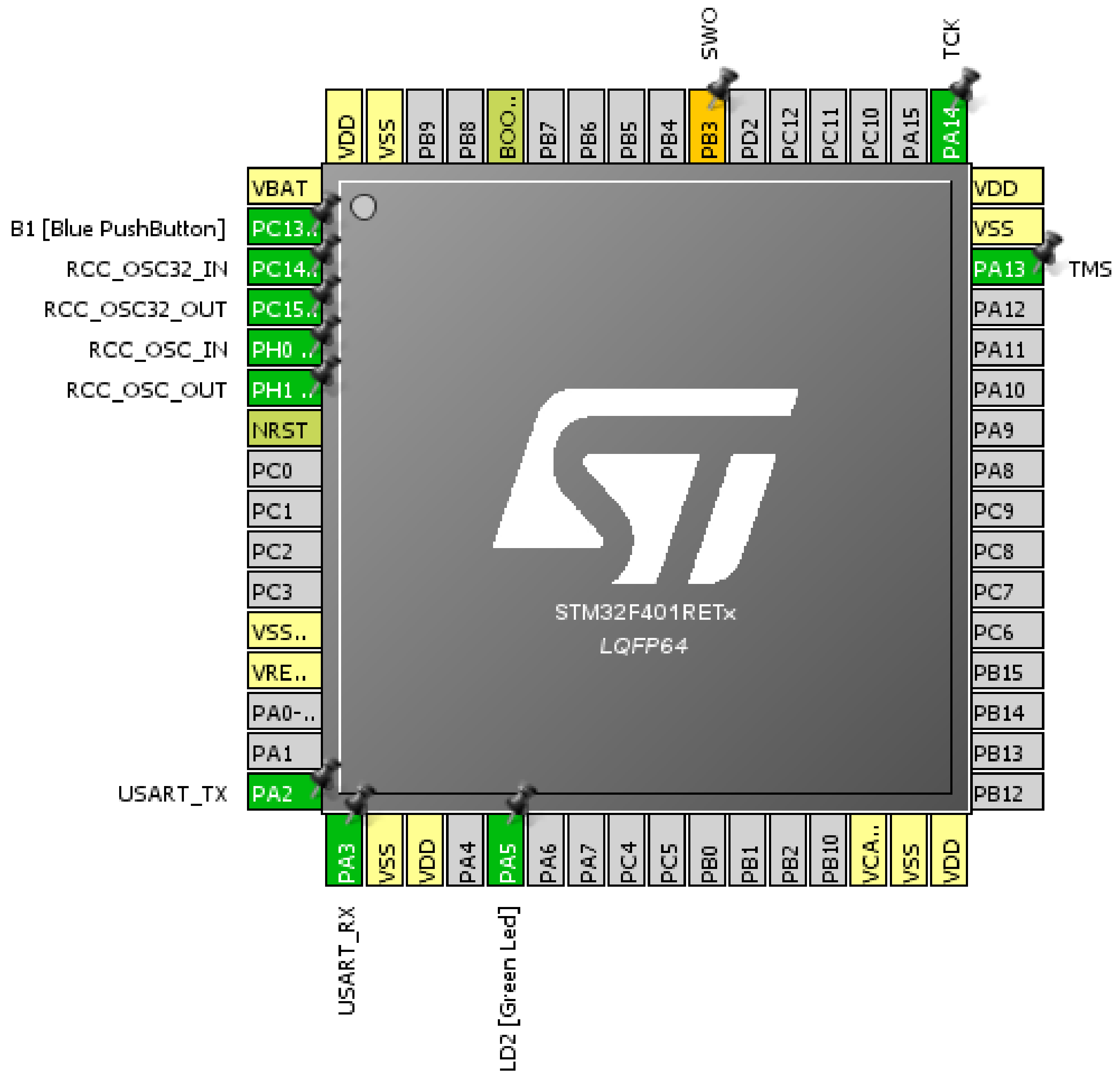
STM32CubeMX

A graphical configuration and low level code generation tool for STM32 ARM Cortex-M microcontrollers



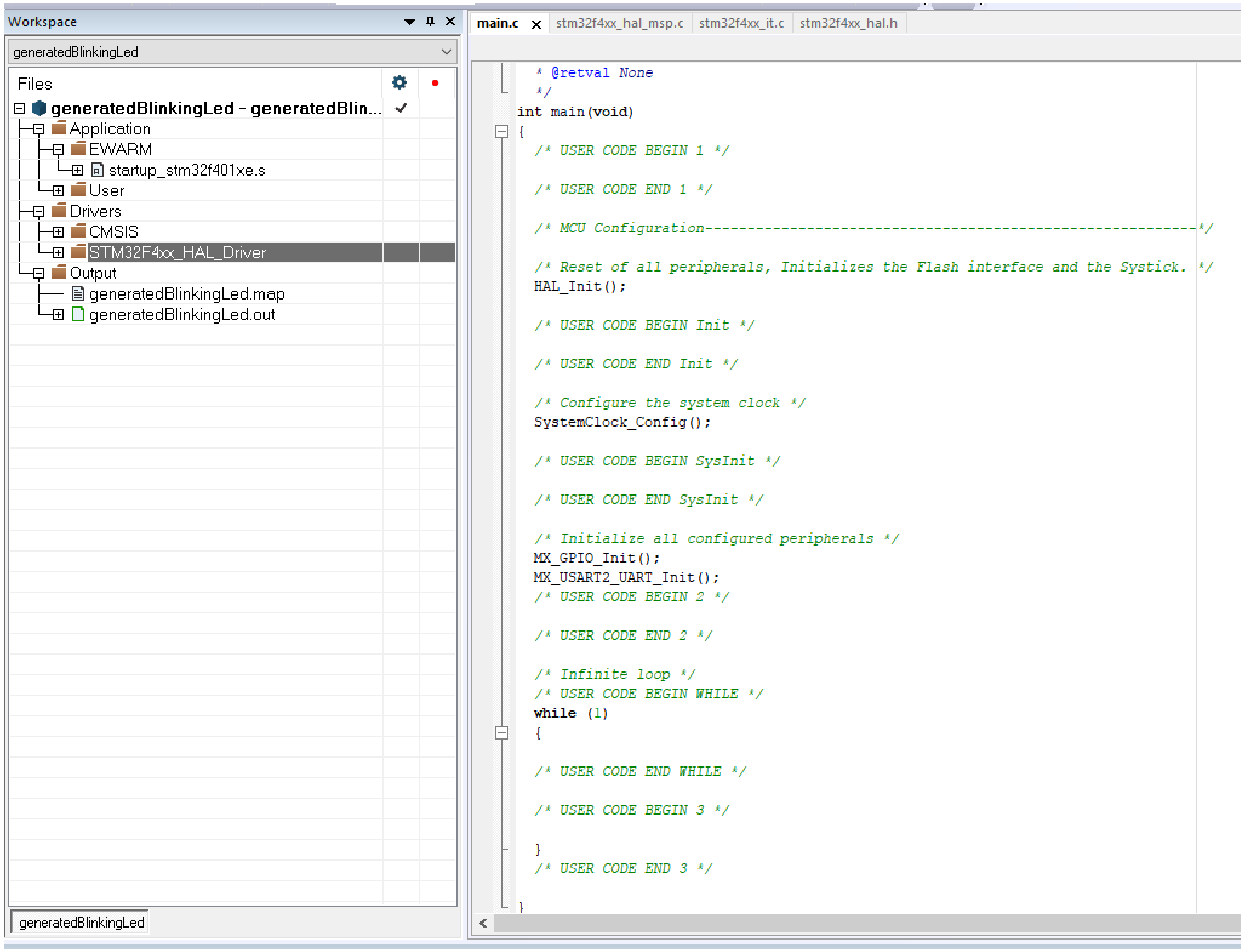
Getting started with CubeMx




SWO
TCK
VDD
VSS
PB9
PB8
BOO..
PB7
PB6
PB5
PB4
PB3
PD2
PC12
PC11
PC10
PA15
PA14
VBAT
B1 [Blue PushButton]
PC13..
RCC_OSC32_IN
PC14..
RCC_OSC32_OUT
PC15..
RCC_OSC_IN
PH0 ..
RCC_OSC_OUT
PH1 ..
NRST
PC0
PC1
PC2
PC3
VSS..
VRE..
PA0-..
PA1
USART_TX
PA2
VDD
VSS
PA13
TMS
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
PC6
PB15
PB14
PB13
PB12
STM32F401RETx
LQFP64
PA3
VSS
VDD
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2
PB10
VCA...
VSS
VDD
USART_RX
LD2 [Green Led]

CubeMx Setup for NUCLEO-F401RE

- New Project
- Board Selector
  - NUCLEO-F401RE
- Start Project
- Initialize all peripherals with their default Mode (Yes)
- Select “Configuration” Tab
- Select “GPIO” under the “System” section
  - Check out the settings for the GPIO there (no need to change anything) --> Cancel
- Project --> Generate Code --> Ok --> (let it download missing files)
- Open Project



```
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */
}
```

Generated Code

Assignment 03

Suggested Reading

- ***"The Definitive Guide to ARM Cortex M3 & M4" by Joseph Yiu (Third Edition)***
  - Chapter 1.1, 1.2, 1.3, & 1.4

ARM Architecture

BY GUEST INSTRUCTOR LAWRENCE LO